Lab 6 Report: Planning and Following Paths around the Stata Basement

Team #13

Kaleb Blake
Lucian Covarrubias
Seth Fine
Jesse George
Ivory Tang

6.141 Robotics, Science, and Systems - Spring 2022

April 16, 2022

Editor: Lucian

1 Introduction

By: Ivory Tang

Throughout this lab, we explored open problem of path planning along with a method to follow set trajectories. We implemented and compared two different types of planning algorithms, search-based via A* and sample-based via Rapidly-Exploring Random Trees (RRT). We worked with a known global map of the Stata basement where we would eventually test our robot. Using A* as a planner, our robot could take in a start and end point and create a near optimal path between them. In addition, we implemented a pure pursuit algorithm which, given the planned trajectory, would issue steering commands to our car to follow the trajectory as closely as possible. After implementing the algorithm and passing unit tests, we first tested our code in simulation. This allowed us to fine tune certain parameters such as look-ahead distance and decide on which planning algorithm was more appropriate for our use case with the necessary optimizations. Once we were confident in the simulation results, our algorithms were ready for real time path planning and tracking on the robot. Throughout our implementation and testing, we needed to optimize for runtime efficiency which led us to utilize and explore downsampling of the search space.

2 Technical Approach

2.1 Overview

By: Ivory Tang

In order to implement a path planning algorithm and then follow it precisely, there are multiple necessary parts.

Firstly, an accurate yet efficient planning algorithm needs to create a trajectory for the robot to follow given a start and end point and the map the robot will be traversing in. This planning algorithm needs to be quite accurate so that the racecar is not crashing into walls or enters unknown spaces such that it is able to reach its goal in a timely fashion. There are many algorithms out there which are useful for different situations, and we carefully considered which algorithm would be best for our situation before implementing it.

Secondly, a pure pursuit module is necessary to issue driving commands to the racecar given a precomputed trajectory. Depending on the look-ahead point and where the robot is relative to the path it should be following, the steering angle published will change automatically.

Finally, we tested the path planning and pure pursuit model in simulation, resulting in the car following the trajectory fairly closely. To test on the real racecar, we published our steering angles to the racecar and adjusted for real world factors, resulting in the racecar successfully navigating between points in the stata basement.

2.2 Path Planning - Search Based

By: Lucian Covarrubias

2.2.1 Method

The search based method consisted of an A* search algorithm operating on the grid space of the map. A* search takes as input the start and goal positions as well as an occupancy grid. For every node it searches, the node is assigned a cost of reaching it plus a heuristic cost. In our case, the heuristic is simply Euclidean distance from the current node to the goal point. Because costs are updated in this manner, A* does not waste time searching for paths that are far from the goal position and is much more efficient than other algorithms such as Dijkstra's, Breadth First Search, or Depth First Search. Upon finding the goal position, A* returns a trajectory built from every grid location that led to the goal.

An important component of search based planning is how we represent the search space. The map provided to the robot is expressed as an occupancy grid, which contains points in pixel coordinates, along with a transformation matrix describing the pose of the map in real world coordinates and a resolution which determines how the map scales to the real world. In order to use pixel coordinates in real world space, a conversion from pixel coordinates (u,v) to real world coordinates (x,y) is necessary. This conversion is constructed from the transformation and resolution features of the occupancy grid.

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_{14} \\ r_{21} & r_{22} & r_{23} & t_{24} \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} u \cdot \text{resolution} \\ v \cdot \text{resolution} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

Given the occupancy grid, it's necessary to clarify where obstacles are and dilate them to account for the size of the car as it would navigate the map. Dilation is the process of making obstacles larger in the map representation by convolving the map with a filter. After dilation, every pixel value is discretized to 1 or 0 via the following function.

$$P(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

While the pathfinding guarantee of A* is valuable, the slower runtime due to an exponential search space is detrimental to a real time path planning system. In order to speed up the path planning time of A*, we tested out methods of reducing our search space via downsampling. Downsampling is the process of taking our original map, and casting it to a reduced space where groups of pixels in the original representation correspond to a single pixel in the reduced representation.

As the search space is reduced, information is lost which means there is no more optimality guarantee for A*. That being said, the performance of A* in these reduced spaces was effective enough to quickly generate valid paths for the robot to traverse. As seen in 1, A* has significantly reduced runtimes when working on downsampled maps, while still generating valid paths for the robot. As paths become more complex, the speed benefits of downsampling become increasingly obvious. In testing, we benchmarked A* performance with downsampling factors ranging from 1 (no downsamplng) to 5 (space reduced by 5^2).

2.2.2 Simulation evaluation

In simulation, the robot was able to plan a path between any arbitrary set of points in under 8 seconds using only a downsampling factor of 2. With higher

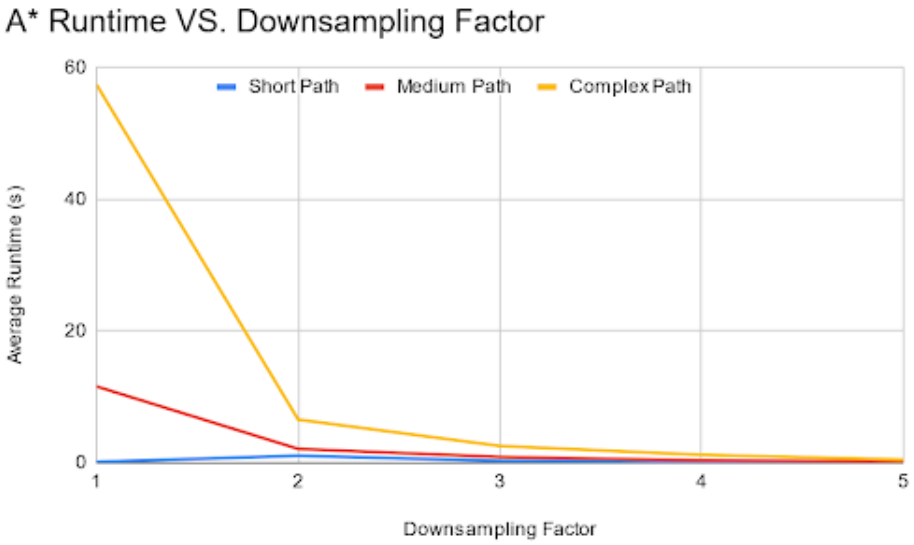

Figure 1: A* Runtimes on similar paths in increasingly reduced search spaces.

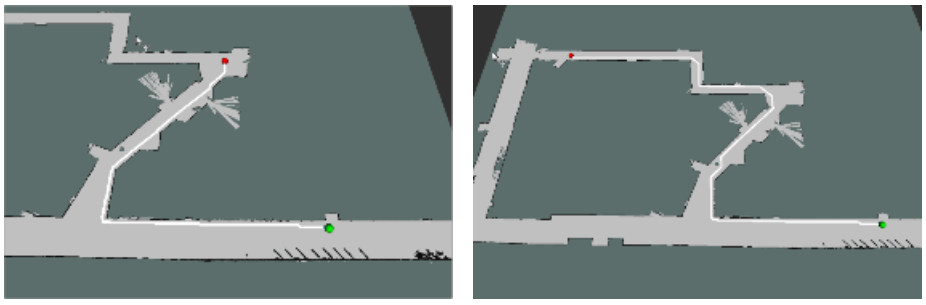

Figure 2: Paths generated by A* with downsampling of 5.

downsampling factors, computation time decreases at a roughly inverse exponential rate. We found a minimum viable computation time of under .5 seconds for any possible path using a downsampling factor of 5, while higher factors would occasionally fail to find paths. As seen in figure 2, which were generated using a downsampling factor of 5, generated paths are still viable for the car to travel.

2.2.3 Strengths/weaknesses

One strength of A* search is that it is a complete algorithm, meaning it will find a solution if one exists. This sacrifices speed to some extent, as our algorithm must search a larger space to find a solution. That being said, the use of a proper heuristic and a downsampling of the search space allows A* search to perform well in the context of the problem posed to our robot. In the given problem, the map is static, so we don't need to quickly react to sudden changes in the environment. With this in mind, the .5s worst case runtime in the Stata basement environment using a downsampling of 5 is fast enough for anything our robot may need to do. This would account for changing destinations, or adding objectives to travel to while still being able to navigate potentially narrow areas without failing to find a viable path.

2.3 Path Planning - Sampling Based

By: Seth Fine

2.3.1 Method

We implemented the rapidly-exploring random trees algorithm (RRT). This algorithm is apart of the sampling based path planning family. In this algorithm, the robot has a known map. From this known map, the search space and its bounds are determined. Additionally, the robot is given occupancy information in the form of a discretized grid of probabilities (same as search based planning above). Using this map and occupancy information, when the algorithm receives initial and goal poses, it undergoes the following iterative process. Randomly select a new node from the search space. This randomness is slightly optimized through a parameter controlling the bias of random sampling towards the goal point. After the new node is chosen, the nearest existing node is identified. Then, using raycasting to conclude that no obstacle blocks the straight line path from the existing nearest node to the new node, the new node is added to the tree with the existing nearest node as its parent.

2.3.2 Simulation evaluation

In simulation, the implementation of RRT behaves as expected. It quickly is able to identify a viable obstacle-free path from the initial pose to the goal pose. However, this path is jagged and often takes wide turns. RRT is governed by

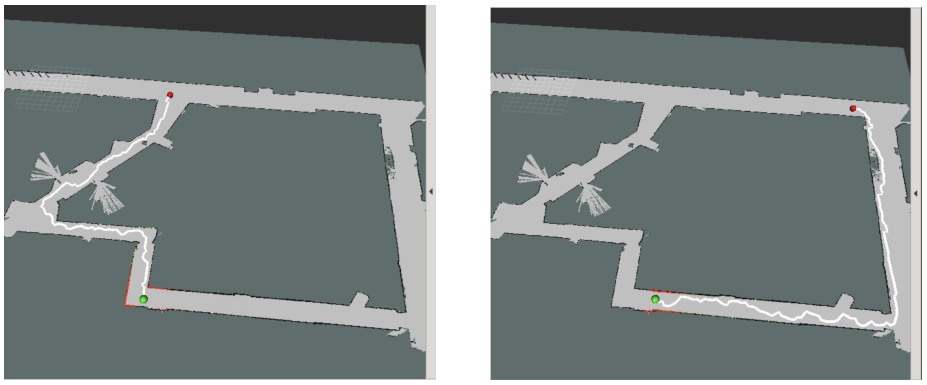

Figure 3: These images taken from the simulator deploying the RRT algorithm demonstrate that RRT produces viable paths from the initial pose to the goal pose albeit with often small jagged curves. The path on the left took 0.86 seconds to complete while the path on the left took 2.8 seconds, respectively.

two main parameters, max length and goal bias probability. Max length is an upper threshold on the distance a new node can be from the nearest existing node. Further, the goal bias probability variable skews the random sampling slightly in the direction of the goal pose. Optimizing values over these variables we were able to improve the optimality of the solution RRT was producing in simulation as shown below.

2.3.3 Strengths/Weaknesses

In comparison to the search based path planning implementation of the A* algorithm, sampling based approaches are faster but produce less optimal paths. In our case, less optimal is defined as greater total distance travelled while still producing a viable path which avoids walls and obstacles. Additionally, the sampling based algorithms work in continuous space meaning the solution space does not to be discretized. This property makes sampling based approaches better equipped to handle models with multiple degrees of freedom such as a system of robotic arms and levers. Overall, while sampling based approaches lose optimality they provide rapid and practical solutions. For this lab, we chose to implement RRT because of its rapid runtime, but it is worth noting the existence of a modified RRT algorithm RRT* which does converge to an optimal solution. This algorithm builds upon RRT by augmenting nodes to include a cost variable which tracks the cumulative distance from the initial pose to the node. Then, using this cost augmentation, when a new node is added to the tree, we are able to check if any nearby nodes can decrease their cost by rerouting through the new node. These strengths and weaknesses were well-exemplified in the simulator, where the RRT algorithm produced paths based on manually set initial and goal poses.

2.4 Path Following

By: Jesse George-Akpenyi

2.4.1 Non-ideal Conditions

Once a trajectory has been mapped out, the next course of action is for the robot to follow that trajectory. In an ideal world, we could give the robot path coordinates that it would follow exactly, navigating the pre-computed path by simply turning according to the curvature of the path as it moved forward. However, the uncertainty in sensor measurements and the car's odometry means that we are never fully certain of the position of the robot in the world. This means that we must take into account the current estimate of the robot position as it follows the path and implement a way to correct any deviations from the path.

2.4.2 A Robust Solution: Pure Pursuit

Pure pursuit is a robust trajectory following algorithm that accomplishes both of these requirements, accounting for the robot's position and deviations from the path trajectory. The algorithm works by first detecting the closest point on the trajectory to the robot. Then a goal point is selected further along the trajectory (closer to the end of the trajectory) that is a certain distance away from the robot's current position. The robot "looks ahead" to this goal point, with driving instructions being calculated to get the robot to that point from it's current position. This process of determining the goal point is done repeatedly as the robot moves, causing the robot to follow the planned trajectory and to correct any deviations the robot has from this trajectory. Even a relatively jagged and rough trajectory can be reliably followed with a controller based on this algorithm.

2.4.3 Simulation Results and Tuning

The pure pursuit controller was verified in simulation with trajectories built by choosing points within the RViz representation of the Stata basement. After verifying that the built trajectories were being followed, we tested the affects of changing the lookahead distance (which was initially set to 2 meters) to slightly different values. The velocity of the simulated car was 1 meter/second. The experiments for different lookahead distances involved having the robot follow a straight path that had a sharp change in the y direction of 2 meters. The results of these experiments are depicted in the figure below.

This information on how the pure pursuit controller handles corners at different lookahead distances for a set speed is very useful in terms of tuning for particular paths. We decided to leave the lookahead distance of 2 meters unchanged due to its relative stability in both low curvature and high curvature paths compared

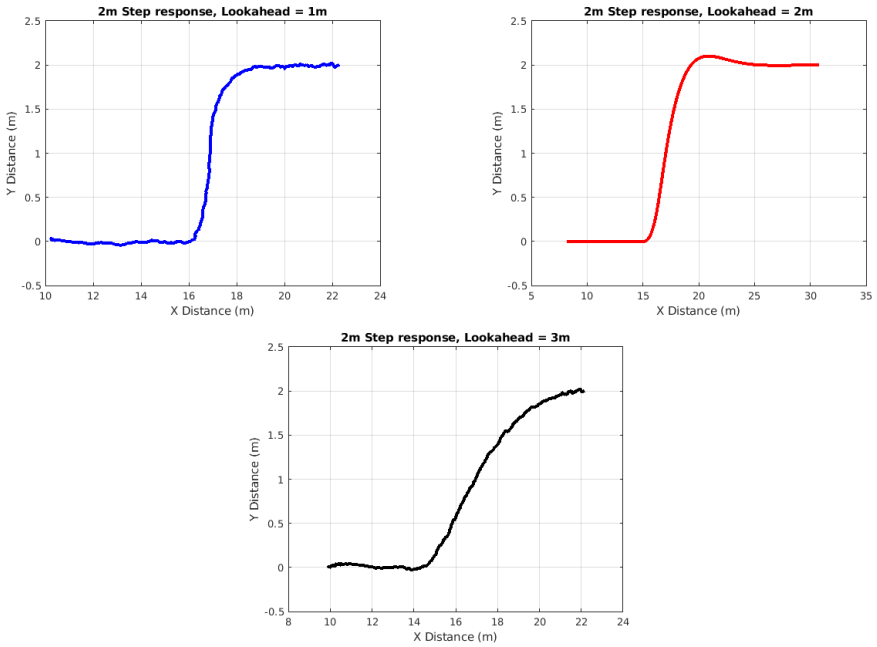

Figure 4: These images are of results of a step response to a 2-meter change in y for the pure pursuit controller set at different lookahead distances in simulation to understand the effect of changing lookahead distance. From the results we can see that smaller lookahead distance results allow for tighter corners to be more reliably followed.

to both shorter and longer lookahead distances. Despite the approximately five percent overshoot for this step size, the 2 meter lookahead distance has a comparable settling time to the one meter lookahead controller and would demand less extreme steering angles for the vehicle fo follow.

3 Experimental Evaluation

By: Jesse George-Akpenyi

The performance of the physical robot was roughly equivalent to simulation, with the exception that the estimated pose of the robot was significantly more noisy due to the slight differences in the geography of the Stata basement. However, the trajectories charted by the path planning module were entirely feasible within the Stata basement's boundaries and were successfully followed by the pure pursuit controller. Longer trajectories would be needed to verify how error resistant the controller is. However, the surprising amount of traffic that crosses through the main long hallway of the basement regularly prevented such tests

from happening. Yet the initial short trajectory tests, depicted in the figure below, indicate that the pure pursuit controller is capable of closely following a path and correcting large deviations of the robot's position from the intended trajectory.

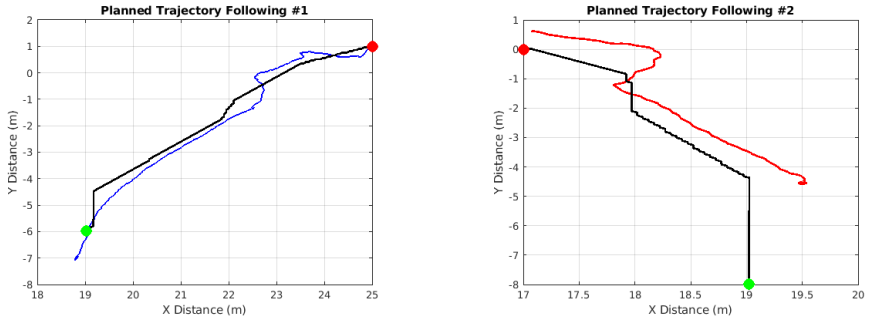

Figure 5: These two images are the estimated trajectories of the robot (blue and red) based on the particle filter localization method plotted with the actual planned trajectories (in black). In the scenario on the left, the initial pose of the vehicle overlapped the starting point of the planned trajectory, resulting in a very close following of the planned trajectory. In the scenario on the right, the initial pose of the vehicle was quite distant from the trajectory, however it quickly begins to converge with the planned path.

4 Conclusion

By: Kaleb Blake
In this lab we explored and implemented path planning and path following algorithms. With respect to path planning we were able to develop A* and Rapidly-Exploring Random Trees (RRT) algorithms. Both algorithms were able to find paths if they existed, but the RRT path was less optimal and less smooth, even though it could find paths faster. We could make the A* algorithm faster with the use of downsampling, though. We used a pure pursuit algorithm to follow the predetermined trajectory created from path planning by changing the steering angle of the car so that it would reach a specified look ahead point if the steering angle was kept constant. We combined these two algorithms to plan a path and follow it in simulation and implemented them on the racecar to see it autonomously drive around the Stata basement.

5 Lessons Learned

Kaleb: In this lab, I helped out with debugging pure pursuit and running simulations and testing out our code on the real car. I got a better understanding

on how pure pursuit works because in the wall following lab we didn't really implement pure pursuit. Instead we used a trial-and-error based PID approach. In fact, when I tried to implement pure pursuit equations at first to get the PID values, my wall follower did not work. I realize now that pure pursuit is most useful, when you have a predetermined path or trajectory to follow. Since, I was working primarily on debugging, I also got to practice reading code written by others and understanding it, as well as using print statements and analyzing outputs to decipher where the problem is. This lab I was more satisfied with my communication on what progress I have accomplished when working and understanding the progress that others have made, so I can more smoothly jump in to help out. I think as a team we worked very well in completing things in parallel too, as we even got one of the extra credit tasks done.

Lucian: For this lab, I worked on optimizing our search based planner (A*) by implementing dilation and downsampling on our search space, and debugging the A* algorithm originally written by Ivory. Additionally, I wrote the infrastructure of our path planner, such as the map setters, and conversion functions between pixel and world coordinates, and implemented our integrated system on the robot. Being in a position to work on many different facets of this project allowed me to see how everything fit together, which gave me a high level of respect for the organization and moving parts between the different features we've written over the last few weeks. Over the span of this course, I've learned to fully trust my teammates to be able to implement the tasks assigned to them, which has allowed me to focus entirely on what I need to do as well. I think this trust and parallelization has allowed our team to quickly implement features and come together to debug the difficult issues in our programs.

Seth: In this lab, my work focused on implementing the RRT algorithm. While this allowed me to explore the specifics and strengths/weaknesses of this particular algorithm, I was also able to connect how RRT relates to other well known algorithms. Specifically, relating in the case of RRT* the optimality statement by using the process of relaxation common in graph optimization problems. It was cool to be able to connect concepts from prior work to this course. From the implementation and testing perspective, I saw the necessity for modular design and modular testing which made adjusting from RRT to RRT* possible. Finally, from a team and communication perspective, we divided work up well doing designs collectively, implementations individually, and coming back together to test integrations.

Jesse: In an attempt to overcome the onslaught of work that I knew this lab would entail, I began to write out as much as the code for pure pursuit (the section of the lab that I chose to work on) as quickly as possible. However, I neglected to include many helpful debugging statements and to check the functions I was implementing before feeding them into other parts of the algorithm. As a result, I had to resolve many issues that I could have caught earlier on if I had been more careful in the steps I was taking and checking my work as I

went. Another valuable lesson I learned during this lab was that separation of function in a larger team like this is very important to maintaining efficiency. Thanks to the fact that we could tackle this lab from two essentially separate angles, we were able to implement something that would've taken all of us much more time if we tackled the entire project at once.

Ivory: In this lab, I focused on the search based implementation and wrote the A* algorithm. I was most interested in the path planning portion of this lab, so I also explored the sampling based algorithms via the research papers provided and got a good sense of the strengths and weaknesses of search based versus sampling based algorithms and the different scenarios they are each good for. Having taken many algorithm classes in the past, it was really interesting to see some of those algorithms applied in a real world scenario on our racecar. The importance of optimizing algorithms to achieve greater efficiency was also emphasized specifically by this lab, as the map could be complicated and a slow algorithm could prevent us from finding an accurate path within time restrictions. This lab really elucidated why there are so many teams and research groups working on constructing new algorithms and improving on existing ones, for there are so many tradeoffs to consider between accuracy and speed even in a relatively simple problem such as navigating the stata basement in which there were few obstacles along the way. Finally, from a team perspective, this lab was quite modularized, meaning splitting up the work and testing the various modules separately and in simulation was effective before we combined the pure pursuit and planning algorithm part to create our final product.