Lab 6 Report: Explorations in Path Planning

Team 24

Harry Heiberger (Editor)
Ayyub Abdulrezak
Amanial Abraham
Michael Chen
Nicole Khaimov

6.4200 (Robotics: Science and Systems)

April 27, 2023

1 Introduction

[Aman Abraham]

In previous labs, we have explored fundamental aspects of autonomous robotics, such as wall following (Lab 3), object detection (Lab 4), and Monte Carlo localization (Lab 5). Through these skills, our robot has gained a strong understanding of how to navigate its surrounding environment and where it is within it. While these ideas are very useful on their own, they have also provided the necessary foundation for the more complex problem of path planning and trajectory following for a robotic racecar.

In Lab 6, our objective is to design a robust path planning and trajectory following algorithm that builds upon the knowledge acquired in previous labs. The primary challenge lies in the integration of these various techniques into a cohesive system. Our approach involves independently developing optimal solutions for both path planning and trajectory following, followed by their integration.

Our path planning algorithm takes the inputs of a map, along with an initial and goal position, and returns a series of segments that form a collision-free path between the start and end positions. We have explored both sample-based and search-based methods, comparing their performance and computational efficiency. In our implementation, we have employed informed searches, to minimize the search space and enhance the efficiency of the path planning process.

In the trajectory following module, we employ a pure pursuit controller to follow the path generated by the path planning algorithm. Our implementation focuses on finding and tracking a lookahead point, which is a point defined by the intersection between a circle of a chosen radius with a lookahead distance and the planned path. This lookahead distance is a critical parameter that impacts the performance of our algorithm, and we have conducted extensive simulations to optimize its value.

2 Technical Approach

[Nicole Khaimov]
Our method for designing a robust path planning and trajectory following algorithm relied on independently developing optimal solutions to each of these problems prior to integrating them into one cohesive system. This section analyzes the trade-offs we determined in possible approaches for each of these modules and discusses the choices we ultimately made in our design.

2.1 Path Planning

2.1.1 Comparison of Algorithms

[Nicole Khaimov]
A path planning algorithm is a function which takes as input a map and an initial and goal position in the map frame. The algorithm may return a series of segments one may follow in order to reach the goal from the starting position while avoiding collision with stationary obstacles, or indicate that such a path was not found.

Some of the most common variants of motion planning algorithms are search-based and sample-based methods. Search-based approaches, which rely on the discretization of the space of possible robot configurations, come in two main forms. Uninformed searches are naive breadth-first or depth-first searches that start at the initial configuration and systematically try neighboring configurations until the goal is found. Naturally, this can get quite computationally intensive. Informed searches such as A* use heuristics to aim the search in directions more likely to lead to a solution and effectively decrease the search space. This heuristic is defined for each point as an estimate of the cost (distance) to travel to the goal. As long as the heuristic does not overestimate the cost, A* can achieve an optimal result.

A sample-based approach will take random samples of the map and connect them with trajectories, removing connections that intersect the obstacle set. A path from start to finish is then found from this randomly generated structure. Importantly, this type of approach does not require building a configuration space. As points are chosen and the paths connecting them drawn, we can use

a collision detection function to determine whether an obstacle intersects with any part of the path. However, this collision detection does require essentially searching the path after it's been constructed to ensure all obstacles are avoided.

Both techniques are capable of achieving an optimal result asymptotically. For a sample-based approach, this means that, if we take enough samples, it is likely that we will at some point encounter the best path to the goal. This is close to the idea of searching in an unordered manner in a continuous configuration space. For a search-based method, the result approaches optimality as the resolution of the map is increased toward continuity. These ideas indicate that either of these methods works well, as long as we are able to dedicate a long amount of time to creating a path. However, for our purpose of building a path planner for the racecar, efficiency was a significant consideration.

Search-based approaches may be described as single-query because the search will always terminate with either a valid path or a statement that such a path could not be found. Sample-based approaches may require multiple queries because it is possible for a valid path to exist, despite not being included in the first sample of points in the space.

Following these considerations, we decided to implement an informed search-based approach for our path planner. It seemed that this would lead to more deterministic outcomes, which would be more reliable in testing and contribute to a robust system for motion planning. This approach was also simple to reason about and implement using A* and the standard heuristic of Euclidean distance. Additionally, the use of a configuration space allowed us to abstract some of the considerations in dynamics restraints regarding obstacle avoidance. To ensure the car would not choose paths close to the wall, we used a process of map dilation in order to expand the boundaries of the obstacles on the map. This created a buffer within the search space, so that the algorithm would not consider positions too close to the wall as valid configurations.

2.1.2 Implementation

[Ayyub Abdulrezak]

For path planning, we went with A*, specifically a variant with a "visited" set that doesn't revisit any nodes. We found that this had the right balance between speed and optimality for our purposes.

As input to the path planner, we have a map consisting of an OccupancyGrid with cells and information on the "openness" of each one. A cell is either free, has an obstacle (this is represented with 100% filled), or is unexplored (these correspond to the insides of the walls in our case). We reduced these cases down to free or not free.

The map also contains information about resolution, and how to transform coordinates (i.e. a translation and orientation). This was used for conversion between pixel space and real-world space. A conversion from the pixel space to real-world space works by first multiplying the pixel coordinates by the map resolution and then applying a transform via a rotation matrix using the map orientation and translation information. These operations are reversed to convert from real-world to pixel space.

In full, our algorithm works as follows. A start position and goal position are given, and both are converted using the above steps into pixel coordinates. The algorithm begins at the start cell and explores outward to the cell's neighbors. The 'neighbors' of a given cell are determined by taking all of the free cells touching the given cell.

As is standard in A*, we add a heuristic, in our case Euclidean distance between our current exploration position and the goal, to the distance of the partial path so far. This is done in order to pick new nodes to explore first in a more efficient way. A heap, or priority queue, sorted on this value is used for the queue of nodes in order to support this. When a node is visited, it is also put into a hash set of visited nodes, and only nodes that have not been visited are added to the queue.

The partial paths followed are stored in the queue with the nodes themselves. So when the goal node is found, it is added to the end of the partial path, and this path now corresponds to a full path from start to end in pixel space. We then take this path, convert the points back to real-world coordinates (only having to do this for the ones in the final path speeds up runtime a lot, as transforming takes quite a bit of time), and return it as our final path to the goal.

2.2 Trajectory Following

[Michael Chen]
The goal behind trajectory following was to take the manually-defined path and implement a pure pursuit controller to follow it. On a high level, the main goal was to find and follow the lookahead point. This is a point defined by the intersection between the circle defined by a radius of the lookahead distance and the path found in path planning.

Trajectories from our path planning algorithm can be represented as a sequential series of points. We first separate the path into little segments represented by a start point, an end point, and the distance between those points. From the segments, we analyze the distance between the pose of the car and the line segments. We do this by extending our segment and finding the projection of where the point location of the car would fall on that line. If the projected point falls beyond the segment, we just clamp it to the starting or ending point.

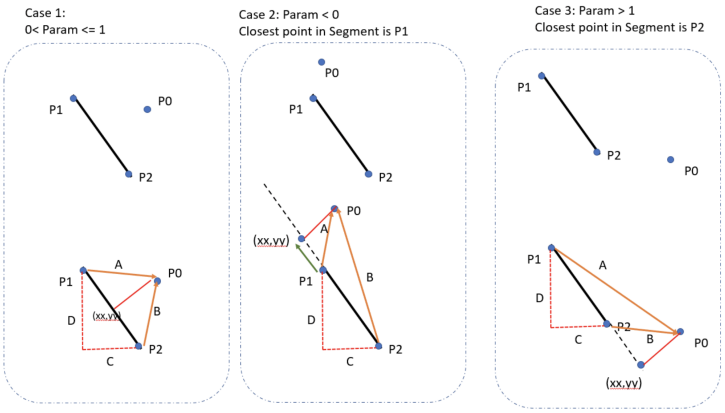

Fig. 1. The various cases we encounter when finding the closest distance the car is to the line segment. The leftmost case describes the scenario when the car is to the right of the line segment and shows that we extrapolate by making a perpendicular line to the line segment. It represents the general scenario when the car is within the bounds of the start and endpoint. The middle case, case 2, shows a case when the car is past the start and endpoint and is, as such, extrapolated to the endpoint. Case 3 shows an example of when the car is behind the start point and is, thus, extrapolated to the start point.

These distances are used to inform our stopping condition and give us metrics on how well our car follows the line. Our stopping condition occurs when we detect the last line segment. When this occurs, we stop the car. The minimum distance is also used to find the current portion of the path we are on.

Our algorithm iterates through each segment of the path starting from the segment closest to the current position of the vehicle. For every segment, we find a quadratic equation representing the intersection between the line segment and the lookahead distance. We then check to see if the quadratic equation we found has real solutions by using the discriminant. If the discriminant is less than zero, it's indicative that the line segment doesn't intersect with the circle. Otherwise, the intersection points are found and analyzed. If the intersection point closest to the end of the line segment is not inside the segment but the other intersection point is, the code selects the other intersection point as the lookahead point. If neither intersection point is inside the segment, the code moves on to the next segment.

If a lookahead point is found that we can follow, we apply a pure pursuit algorithm to follow it. This algorithm takes in the robot's orientation and position from odometry. It then transforms the robot's pose to match the world coordinate frame. Then, we find the relative distance between the robot's pose and the lookahead point to find out whether or not to continue moving. When moving

forward, the robot calculates the turning angle necessary to steer toward the goal. It does this by finding the circular arc that passes through the current position of the car and intersects the goal point.

$$R = \frac{L_1}{2\sin(\theta)} \tag{1}$$

Equation 1 gives the radius of the circular arc that passes through the current position of the vehicle and intersects the goal point.

The turning angle is found using the wheelbase of the vehicle and the distance between the front and back wheels.

$$turn\_angle = \arctan\left(\frac{L}{R}\right) \tag{2}$$

Equation 2 gives the turning angle needed to follow the circular arc.

3 Experimental Evaluation

[Harry Heiberger]

After implementing our motion planning algorithm, we moved on to analyzing its effectiveness and accuracy. This evaluation was performed through a variety of both quantitative and qualitative metrics that are discussed in the following sections.

3.1 Motion Planning Simulation Performance

3.1.1 Simulation Lookahead Distance Analysis

[Harry Heiberger]
Before running our algorithm on our physical racecar, we first performed an evaluation in simulation. This was done in order to avoid the noise and uncontrollable variables of working in the real world. Recall from our technical approach that our trajectory following implementation depends on a lookahead distance variable which determines how far along our trajectory our robot is looking to follow. As the performance of our algorithm depends heavily on the setting of this parameter, a big part of our simulation testing was tuning it in order to get the best results.

Tests were performed by running our trajectory following along the predetermined stata basement path shown in Fig. 2. The same pre-determined path was used in all trials in order to minimize outside factors impacting our data set and facilitate an easier comparison. The tests were performed at a speed of 1.5 for consistency. While higher speeds often require a higher optimal lookahead distance due to the faster rate our robot is encountering obstacles, having an understanding of how lookahead distance impacts following at any speed gives us a strong intuition on how to set it for other speeds. Quantitative data was taken through internal calculations that were logged throughout the duration of the test. Our quantitative data for the simulator tests at each lookahead distance value are shown in Table I. The average error refers to the average distance our simulated racecar strays from the given trajectory. Graphs for this error across the duration of the test for select lookahead distance values can be seen in Fig. 3.

Qualitative data was taken by observing how closely the behavior of the robot matched the given trajectories. Video recordings of the simulator at each lookahead distance value can be found in the link in Appendix section 5.1. In the videos, the car model represents the position of the simulated robot, the red arrow represents the estimated location of the robot's position using localization, and the white line represents the given trajectory that the robot is attempting to follow. The robot always begins at the beginning of the trajectory, signified by a green dot, and continues until it reaches the end of the trajectory, signified by a red dot.

Fig. 2. Pre-determined path used for our lookahead distance testing. This path contains both a left and right turn as well as in-path obstacles in order to more closely match possible real-world environments.

Looking at our data, high lookahead distances such as 2.5 and 3.0 led to the highest trajectory following errors in our testing. This is because with high lookahead distances, the robot notices curves and obstacles in our path far before we actually reach them. This causes the robot to begin turning far too early which, in turn, causes it stray from the given path. This effect can be visualized in the 3.0 lookahead distance graph in Fig. 3 where the large peaks represent places where our robot started turning too early.

As we lower our lookahead distance while keeping speed constant, this effect begins to decrease, leading to lower error. In our testing, we found that a lookahead distance of 1.5 performed the best. At this speed, the simulated car started its turns directly at the start of the trajectory curves, leading to a close matching and, thus, the best performance. This good performance can be visualized in the 1.5 lookahead distance graph in Fig. 3. We expect that at higher driving speeds, this optimal distance would be higher as the car would be reaching the curve more quickly and, as such, would have to start the turn sooner.

Continuing to lower the lookahead distance below 1.5 leads to an increase in trajectory following error. This is because with small lookahead distances, the

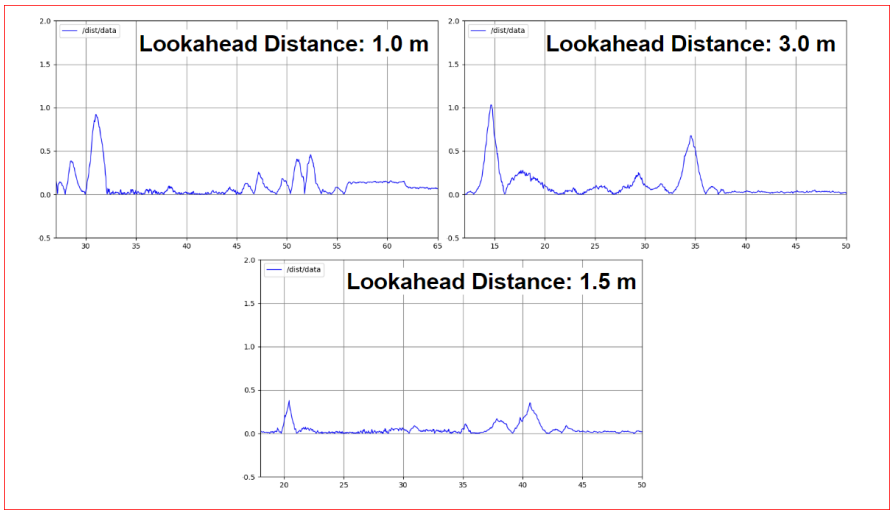

Fig. 3. Trajectory following error charts for various lookahead distance values. As shown in the graphs, the best performance was found with a parameter value of approximately 1.5. Too high of lookahead distance value led to large error peaks at turns due to it starting the turn too early. Too low of lookahead distance value led to oscillations in the following that caused the accumulation of many small error peaks.

point the robot is following is very close to the robot. This means the robot reaches this point very quickly and, as such, has to update the point it's following constantly. These frequent updates lead to a jerky and unstable trajectory following that will often oscillate to the left and right of the path. This effect can be visualized in the 1.0 lookahead distance graph in Fig. 3 where there is the accumulation of lots of smaller peaks due to these oscillations. At a lookahead distance of 0.5, our robot diverged from the path and crashed, meaning we didn't get a graph of this value.

Table I
SIMULATION LOOKAHEAD DISTANCE ANALYSIS

Lookahead Distance (m)	Average Error (m)
3.0	0.20
2.5	0.15
2.0	0.09
1.5	0.05
1.0	0.12
0.5	Crashed

3.1.2 Simulation Planned Trajectories Analysis

[Harry Heiberger]
Besides analyzing lookahead distance, we also qualitatively examined the optimality of our planned trajectories in simulation. Video recordings of these tests can be found in the link in Appendix section 5.2. At the basic level, given any two points in our stata basement map, our algorithm was able to quickly find an optimal path between them. As shown in the videos, when there are multiple possible paths available, our algorithm always chose the shortest direction available. Additionally, these generated paths tightly followed the inside corners and walls along the path, further minimizing the amount of open space the robot had to traverse. Given that minor fluctuations in the chosen path have minuscule impacts on the time to drive said path, our trajectory planning algorithm performed at a sufficient level of optimality.

3.2 Real World Performance

[Michael Chen]
To see how the robot performed amidst the noise of real-world data, we tested the path planning code on the physical robot. We used similar evaluation metrics to the simulation scenarios and drove the robot along the path shown in Fig. 2. We tested how the code performed under different velocities and recorded what lookahead distance we used, how fast the robot went, and the error over time.

Table 2: REAL WORLD DATA COLLECTION

Car Speed	Lookahead Distance (m)	Average Error(m)	Time (s)
0.5	1.2	0.14	77
1	1.2	0.16	38
1.5	1.8	0.17	25
2	2.3	0.22	19
2.5	3.0	0.24	18
3	2.8	0.32	18
3.5	2.8	0.30	18
4	2.8	0.17	17

The overall trends we noticed were that the time taken to complete the path decreased as expected for higher speeds, but plateaued after a speed setting of 2 m/s. This could be due to changes in the robot's actual speed not scaling linearly with the increase in the speed parameter, and the car enforcing an artificial speed cap. Our data also shows that the lookahead distance had to be increased as the car's speed went up. The initial increase can be explained by the cars need to see further ahead when it's going faster as it's reaching points more quickly than before. The plateau of the lookahead distance at 2.8 can be

explained by the speed cap of the car previously mentioned, meaning the speed was no longer increasing despite changing the parameter. The average error had a general upwards trend which makes sense as we'd expect it to be more difficult for our algorithm to follow the path as the speed of the car increased. The error had a great deal to do with how much the car would oscillate around the target path which relates to the lookahead distance. This is because the closer you set the lookahead distance, the more you are "micromanaging" the robot's trajectory which results in shorter quicker turns. You can see this oscillation in the figure below.

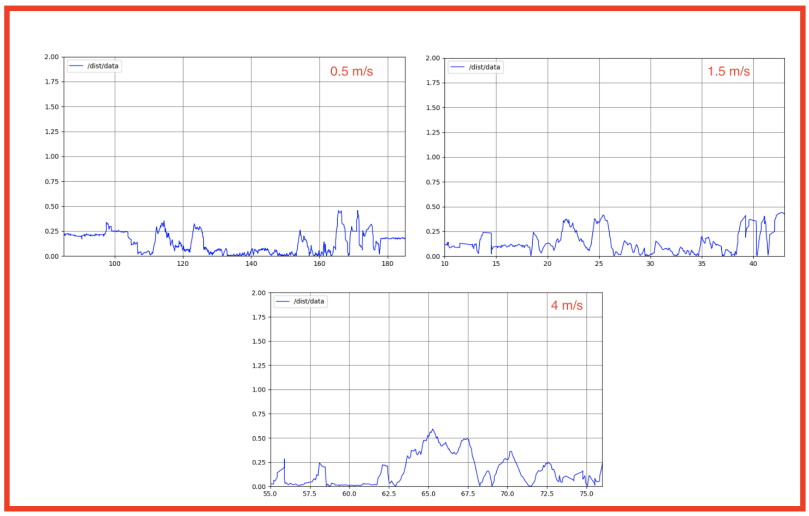

Fig. 4. These are examples of the distance error graphs we got in our testing. Top Left graph shows the 0.5 speed trial, Top right shows the 1.5 speed trial and the bottom shows the 4.0 trial.

3.2.1 Extra Credit: Maximum Speed Test on Course Track

Qualitatively, our vehicle performed very well at every velocity we tested at. Thus, we tried the challenge by completing the track laid out by the course staff. You can see the path our path planning code generated to complete this track in Fig. 5 below. Our best time was approximately 23 seconds. A video of this run-through was sent to our TA to be published on the leaderboard.

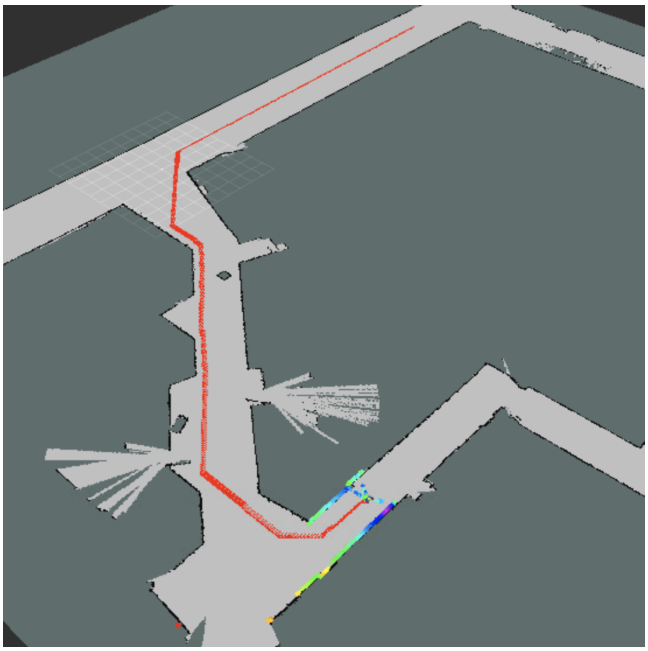

Fig. 5. The path generated by our path planning algorithm for the race competition. As shown, the path is fairly optimal as it moves tightly along corners/edges and only turns when necessary.

4 Conclusion

[Aman Abraham]
In conclusion, Lab 6 has been a pivotal step in our journey to understand and develop autonomous robotic systems, as it enabled us to incorporate the skills and methodologies acquired in previous labs, especially Lab 5 (Monte Carlo localization). By building on these foundations, we successfully designed an effective path-planning and trajectory-following algorithm for a robotic racecar.

Our technical approach involved optimizing the path planning and trajectory following modules individually before integrating them into a single, cohesive system. The path planning module harnessed the power of an informed search algorithm to minimize the search space and deliver computationally efficient solutions. The trajectory following module utilized a pure pursuit controller to track a lookahead point, which was determined through extensive simulations and fine-tuning of the lookahead distance parameter.

Experimental evaluations of the integrated system showcased its effectiveness and accuracy in various scenarios. We assessed the system's performance through both quantitative and qualitative metrics, including trajectory following error,

optimality of planned paths, and overall adaptability to environmental changes. Our results clearly demonstrate the successful implementation of our approach, resulting in a reliable and efficient autonomous robotic system.

In closing, Lab 6 has provided us with a solid foundation for future advancements in autonomous robotics. By continuously refining our algorithms and incorporating additional capabilities, we can develop increasingly sophisticated robotic systems capable of navigating and adapting to a diverse range of environments and challenges. As we move forward, we hope to improve and incorporate all our labs to produce a final product that functions robustly in our final project.

5 Appendix

5.1 Simulation Lookahead Distance Videos

Link: https://drive.google.com/drive/folders/1lE06iWaoPWp33uOGMn3BWxiAg8QmX-VZ?usp=sharing

5.2 Simulation Planning Videos

Link: https://drive.google.com/drive/folders/1OWLSCOnpLT8aSRdGFqbkVADWBAOuEkSI?usp=sharing

6 Lessons Learned

[Harry Heiberger]
For me, lab 6 was a very interesting assignment that grew my ability to implement and understand complex algorithms. In this lab, I had the opportunity to help implement both a mathematically intensive trajectory-following algorithm as well as a time-efficient graph search algorithm. Both of these components were very different problems that practiced different yet useful technical skills. Additionally, this lab also gave me the opportunity to improve my debugging ability. Integrating both of the components of this lab in the real world led to a lot of issues for our team, meaning I had to efficiently find ways to track down possible sources of error. This is an incredibly valuable skill that I will use throughout my career as a software developer.

On the communications side, this lab was very useful for improving my ability to plan and scope technical problems. Because this lab had two separate parts, our team had to agree on specifications that both of our modules would follow. Before starting our code, we all met to discuss our plans for the lab in detail. This planning saved us from potential issues during integration later. As a future software developer, I'll get the opportunity to work on a variety of large-scale projects. Thus, being able to plan out and agree on a specification

with my team is something that I enjoyed getting practice in.

Finally, on the collaboration side, working with a team of five continues to be an extremely valuable experience. From coordinating tasks to communicating our progress, our team has greatly improved our efficiency together over the semester. We all know each other's strengths and try to assign tasks such that we all get the opportunity to grow from the lab while getting any support needed. Being able to effectively utilize a team's strengths and weaknesses is extremely so it has been good to get practice with it.

[Ayyub Abdulrezak]

I learned a lot in this lab. I learned about how to structure good project code. This was very important when writing the path-planning code, as it was much easier to debug when it was well-structured. In addition, I also learned more about how to well structure the splitting up of work and how modules interact. This made it more interoperable between the path planner and trajectory follower. I think these lessons will be very helpful in the final project challenge.
[Michael Chen]

Lab 6 was extremely interesting! I really enjoyed the challenge of having to build off our localization code and move to path-planning. I learned a lot more about the different types of algorithms used to do path planning like A* and RRT*. I had the pleasure of learning and working on a sampling-based approach. It was very frustrating to work with the hardware this time around when I was collecting data. We ran into a plethora of issues just trying to get the car to run even when we knew the algorithm worked relatively well. It took us the most time trying to integrate everything together and it taught me many lessons. One lesson is just that integration is hard so it's important to integrate early and often will little pieces of the code before we perfect it. Another is to try out the code with simple versions of the algorithm on the real car instead of only in simulation before finishing the whole thing. On the aspect of collaboration, it's been fun as always to work with this incredibly talented team. It's been cool to see how much we've grown in terms of learning to work with each other. It's always difficult because everyone has different schedules which makes distributing work a lot more challenging. Moving forward, I really wish I spent more time on algorithms than being the guy that tests the robot. Regardless, I still learned a lot!

[Nicole Khaimov]
This lab was really interesting because it
[Aman Abraham]
Lab 6 was a fascinating experience that integrated lessons from all previous labs to deepen my understanding of autonomous systems and the variety of research and algorithms applicable in the field. This lab was crucial for improving

communication within our team, as the multiple components and incorporation of previous labs demanded clear and concise exchanges among team members regarding their tasks.

This week, communication played a vital role in enhancing my ability to work effectively within a team environment. As we became more familiar with each other's strengths and weaknesses, we were able to assign tasks that fostered personal growth and provided the necessary support for successful completion. In addition was able to identify what others worked well on.

In conclusion, this lab was an invaluable experience for me that really summed up what I was taught in this class and I am extremely grateful to understand these concepts and apply what I learned in my future.