Lab 3 Report: Wall Follower

Team 3

Thomas Buckley
Fiona Wang
Sriram Sethuraman
Ben Lammers

6.4200 RSS

March 15, 2025

1 Introduction

Fiona Wang

In autonomous vehicles, safety controllers and wall following are both basic features that can play an important role when accomplishing more complex tasks in the future. Our safety controller will be used in all future labs to ensure safe testing and prevent damage to our robot. The wall follower is our first step towards more complex path following and navigation abilities. By building off and learning from our current wall follower, we will be able to accomplish tasks such as lane and trajectory following.

In Lab 3, our first goal was to implement a safety controller to ensure that our racecar will always stop when it drives too close to an obstacle. This controller considers both a minimum distance and time to collision when deciding whether or not to stop. Our second goal was to create a wall follower to accurately maintain a set distance from a wall. The finished wall follower should be able to maintain desired distances and adapt to changing wall contours regardless of the given speed. The steps we took to complete this task include splicing our 2D LiDAR scan, performing linear regression, and implementing a PD controller. During this lab, we transitioned from simulation to real world, adapting our technical solution and experimental evaluation to fit the physical robot.

2 Technical Approach

Ben Lammers, Sriram Sethuraman
The car's drive commands are derived from three sources: the remote control, our wall follower, and the safety controller. In Figure 1, we can see that the safety controller overrides all autonomous commands, but the remote controller can override the safety controller if it is being used. The remote controller was

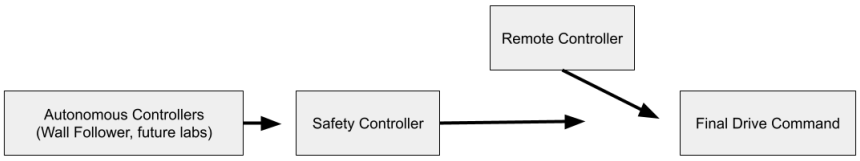

Figure 1: **System Diagram** demonstrating the priority between safety controller, wall follower, and the remote controller.

provided to our team, and sends drive commands from a joystick. We'll break down how the safety controller and wall follower work in the sections below.

2.1 Safety Controller

The goal of the safety controller is to prevent racecar collisions as we continue to develop and troubleshoot algorithms that drive the robot. In developing the safety controller, we wanted something that would prevent collisions at the last possible moment. In other words, we wanted something that was not too aggressive, but that would halted the robot before it could damage itself or its expensive equipment.

The safety controller works by constantly examining LIDAR data. This data is sliced to provide a 30° angle directly in front of the robot. This angle is enough to capture obstacles directly in front of the robot, while allowing the robot to get close to obstacles on its side (for purposes like wall following).

Once the LIDAR slice is obtained, the safety controller calculates the minimum distance within this region. This ensures that the robot only reacts to obstacles directly in front of it, preventing unnecessary stops due to objects on the side. The controller then calculates the time to collision, defined as:

$$t_{\text{collision}} = \frac{\text{minimum\_distance}}{\text{velocity} + 0.01} \tag{1}$$

where a small constant 0.01 is added to the denominator to avoid division by zero when the robot is stationary. Given this minimum distance to an obstacle,

the safety controller overrides all other programs controlling the robot in two cases:

1. The minimum distance is less than 0.3m
2. The collision time is less than 0.4s.

We found that these two conditions were able to balance both maintaining a minimum distance with accounting for greater stop time for higher velocities. The values for minimum distance and collision time were decided based on what delivered consistent performance, which will be discussed at length in the evaluation section.

When either of the two stopping conditions are met, the safety controller overrides any existing drive commands and issues a new command setting the speed to zero (to stop the robot). The controller publishes this zero speed command to the drive topic, overriding any other commands and ensuring a prompt response.

The controller is implemented as a ROS2 node that subscribes to both the LIDAR scan topic and the standard drive command topic to receiving distance and velocity data. The velocity of the robot is continuously updated from incoming AckermannDriveStamped messages, ensuring accurate collision-time calculations.

2.2 Wall Follower

The goal of the Wall Follower algorithm is to maintain a set distance from the wall while driving alongside it. This requires the robot to be able to determine where the wall is and what the distance from the wall is, as well as to control the steering angle to adjust for any discrepancy between the observed distance and the target distance.

First, we will address the problem of determining where the wall is. The robot has a two-dimensional LIDAR unit capable of accurately measuring the distance to a set of points within a $\frac{3\pi}{2}$ arc centered in front of the robot. The LIDAR unit takes 1080 evenly spaced distance measurements along this arc and returns them in a list. The list is ordered such that it begins to the right of the robot at LIDAR angle $-\frac{3\pi}{4}$ and ends to the left of the robot at LIDAR angle $\frac{3\pi}{4}$, with straight ahead corresponding to LIDAR angle 0.

To start, as shown in Figure 2, we define d_ℓ and d_{st} as the look-ahead and look-back distances, respectively. These are parameters that will help determine what part of the LIDAR data we need to read in order to see the wall.

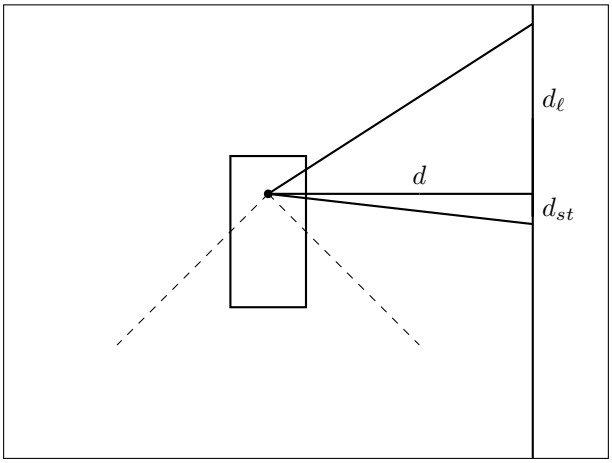

Figure 2: **LIDAR measurement visualization.** Forward is up in this figure, and the dashed lines show the limit of the LIDAR. d_ℓ and d_{st} shown are the look-ahead distance and look-back distance, as stated above.

Doing some geometry, the LIDAR start angle and LIDAR end angle can be calculated in terms of d (our target distance), d_ℓ, and d_{st} as follows:

$$\alpha_{st} = \frac{-\pi}{2} + \tan^{-1}\left(\frac{d_{st}}{d}\right) \tag{2}$$

$$\alpha_{end} = \frac{-\pi}{2} + \tan^{-1}\left(\frac{d_\ell}{d}\right) \tag{3}$$

And to be able to slice the incoming LIDAR list we can use these start and end angles to calculate the start and end indices of the target region of LIDAR data.

$$i_{st} = \frac{\alpha_{st} - \theta_{min}}{\Delta\theta} \tag{4}$$

$$i_{end} = \frac{\alpha_{end} - \theta_{min}}{\Delta\theta} \tag{5}$$

We how have a list of angles and distances and need to convert them to cartesian coordinates. This is fairly straightforward and can be done as so:

$$(x, y) = (d\cos(\theta), d\sin(\theta)) \tag{6}$$

(d is the distance the LIDAR reports and theta is the angle)

These coordinates are now run through a standard linear regression to fit a line to them. This line is our approximation of where exactly the wall is. Next, we have to calculate the distance from the robot to the wall.

Below is Figure 3, containing the line that the robot has estimated to be the wall in the coordinate frame of the robot. We can see that the sideways distance to the wall is approximately the y-intercept. However, since the robot is moving, we decided to take the projected distance to the wall at a future moment in time using the following equation:

$$\text{dist} = \left| \frac{\text{slope} \cdot (d_{\text{st}} \cdot 3 + d_{\text{l}})}{4} + \text{intercept} \right| \tag{7}$$

This allowed for more accurate wall following at higher speeds.

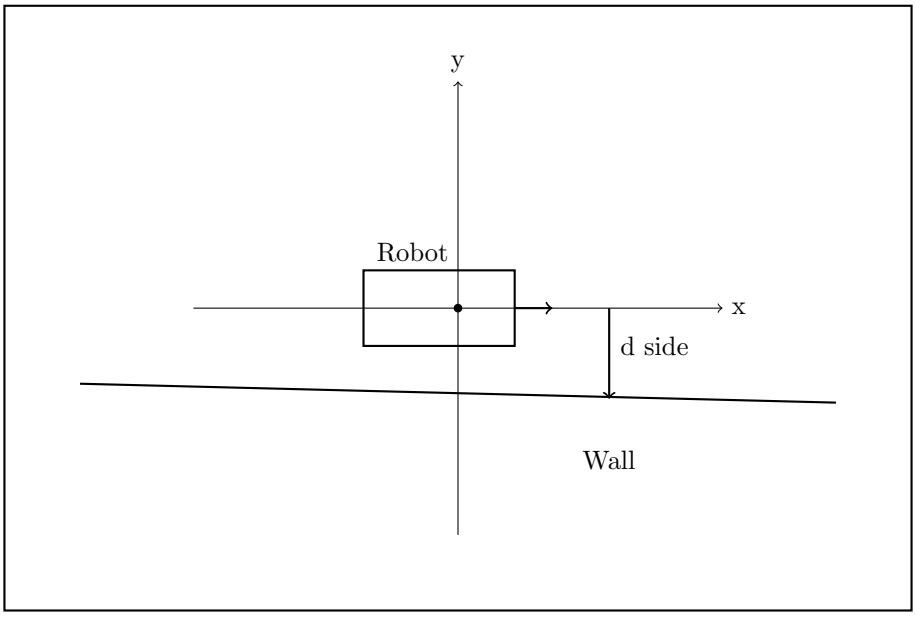

Figure 3: The wall line and side distance in the coordinate frame of the robot

Additionally, in order to take turns with the wall directly in front of the car, for example at a corner of a square room, we also calculated the forward distance to the nearest wall using the same calculation as the Safety Controller.
We can combine these two distances (wall distance and forward distance) into an error term. Ideally, this error term is zero and this occurs when the robot is exactly the correct distance sideways from the wall and there is no wall in front of it. We want this error term to increase in magnitude when we are the wrong distance away from the wall sideways or are approaching a front wall. To satisfy these requirements, we came up with this equation:

$$e(t) = K_p \cdot (d_{\text{des}} - d_{\text{side}}) + K_{p_{\text{front}}} \cdot \frac{1}{d_{\text{front}}} \tag{8}$$

where k_p and k_{pfront} are weighting terms that we tune.
Lastly, now that we have an error term, we used a PD feedback controller to minimize that error by altering the steering angle using the following control law:

$$\delta = K_p \cdot e(t) + K_d \cdot \frac{e(t) - e(t - \Delta t)}{\Delta t} \tag{9}$$

The ROS implementation of this was fairly straightforward, essentially typing these equations verbatim in Python inside our Wall Follower Module.

3 Experimental Evaluation

Fiona Wang, Thomas Buckley

To test our technical solution, we developed a replicable procedure to evaluate the accuracy of our safety controller and wall follower. When evaluating our safety controller, the performance metrics we focused on were the distance from the wall and the time from collision. For our wall follower, we chose to focus on the distance from the wall and the error compared to our desired distance.

For our analyses, we developed a pipeline that collects data directly from a Rosbag. The script subscribes to various topics—including LiDAR laser scans and drive commands—and saves the results to a CSV file for further examination. This pipeline enabled rapid robot testing and debugging. For instance, while developing our algorithm, we could visualize the robot's path and error in Rviz, which helped us pinpoint exactly where the algorithm was failing.

3.1 Safety Controller Evaluation

To evaluate our safety controller, we programmed the robot to drive straight forward into a wall at a given speed. In order to evaluate our controller at typical and extreme conditions, the speeds we chose to test were 1, 2 (typical speeds) and 3 m/s (more extreme speed). During each test, we logged the distance from the wall and time from collision starting from the moment the safety controller is activated, logging until the robot is fully stopped. After data collection, we used our data analysis script to graph the relationship between time the distance from the wall for each speed.

From these tests, our goal was to evaluate whether both the minimum distance and the desired stopping time are used effectively in our safety controller. From the 1 and 2 m/s graphs, the final stopping distances are 0.27 and 0.32 m, respectively, which matches our desired stopping distance (0.3 m) relatively

well, showing that the stopping distance case was activated for 1 and 2 m/s (Shown in Figure 4). From the 3 m/s trial, we received a final distance of 0.45 m from the wall. In this case, our results show that the robot stopped because the time-to-collision case was activated, resulting in a stop earlier than 0.3 m away. This feature enhances the safeness of our controller in high-speed situations, and from our wall follower evaluation, we can see that the feature is effective during real world testing.

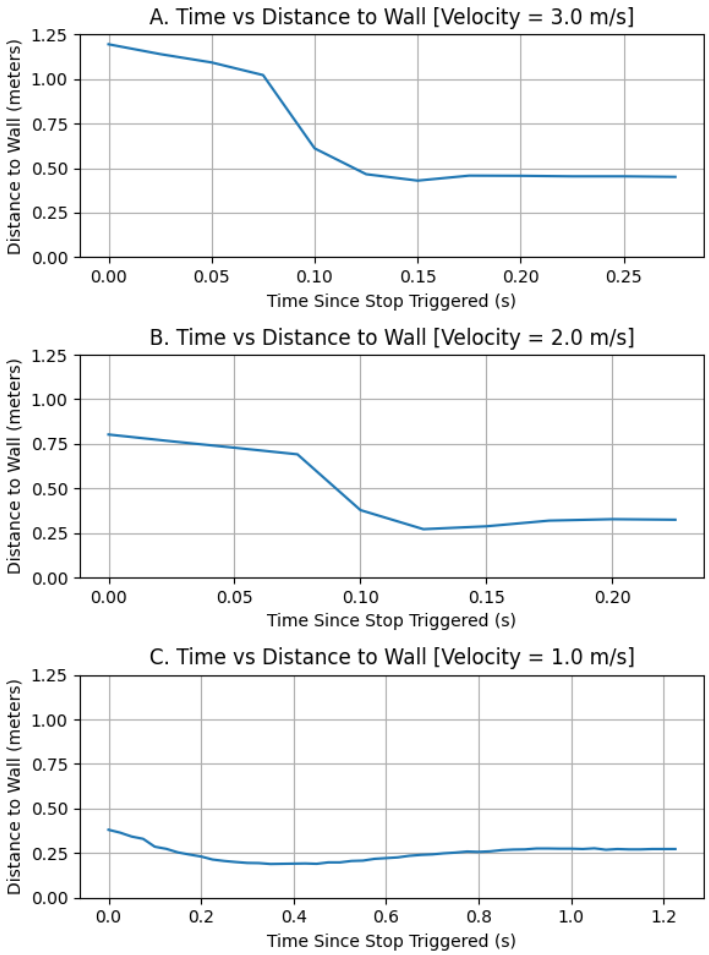

Figure 4: **Safety Controller Trials.** Safety controller trials for 3 (A), 2 (B), and 1 (C) m/s comparing the time since the safety controller was activated (seconds) and the distance to the wall (meters).

3.2 Wall Follower Evaluation

For developing and evaluating our wall follower, we chose two sections of the Stata basement. Each wall presented a unique challenge and gave us a replicable setup to refine our algorithm. We conducted multiple trials with Stata Wall 1, since it had a variety of challenging corners. We then performed a higher speed validation trial on Stata Wall 2 (equally challenging but with very different geometry than Stata Wall 1). Testing on both ensures that our algorithm is not overfitting to one wall or the other. The trials performed are summarized in Table I.

Table I: Wall Follower Test Cases at Stata Basement

Wall Test Case	Speed (m/s)	Desired Distance (m)	ROS Bag Recorded
Stata Wall 1	1.0	0.5	Yes
	1.0	1.0	Yes
	1.5	1.0	Yes
Stata Wall 2	2.0	0.5	Yes

To assess how closely the robot maintained the desired distance from the wall, we computed the average distance measured by the LiDAR sensor. Depending on whether the robot was following a wall on its left or right side, we selected the corresponding set of 20 LiDAR points (i.e., the 20 left-most points when following a wall on the left, and the 20 right-most points when following a wall on the right). This average provided a reliable measure of the robot's actual distance from the wall, which we then compared to the desired distance. This is different than the regression approach we use to map the wall in our algorithm since the regression line becomes a low-fidelity representation of the wall when approaching corners or obstacles.

The performance of our wall follower is illustrated in Figures 6 and 7. Overall, the system maintained proximity to the wall; however, we observed oscillatory behavior as each trial progressed. Additionally, the algorithm consistently generated wider turns than anticipated, resulting in increased deviations from the desired distance during cornering maneuvers. Notably, experiments conducted on Stata Wall 1 indicated that our algorithm was effective at higher velocities, but would that the higher velocity amplified these oscillations.

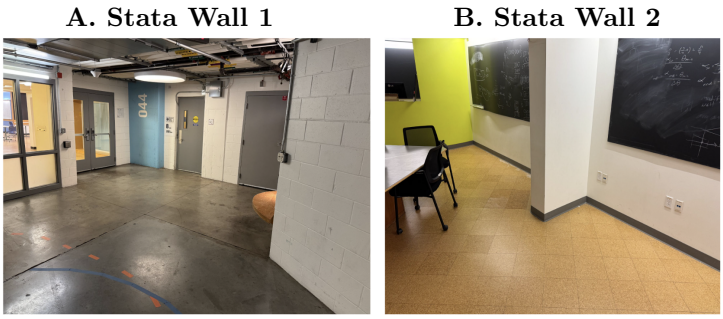

Figure 5: Two walls for testing and validation of our wall-following and safety controller algorithm. In A. Stata Wall 1 was used for testing different speed and distance parameters, and adjusting the PD parameters and algorithm. In B., Stata Wall 2 was used for validation of our algorithm at a higher velocity. Wall 2 contains a difficult wall protrusion and turn.

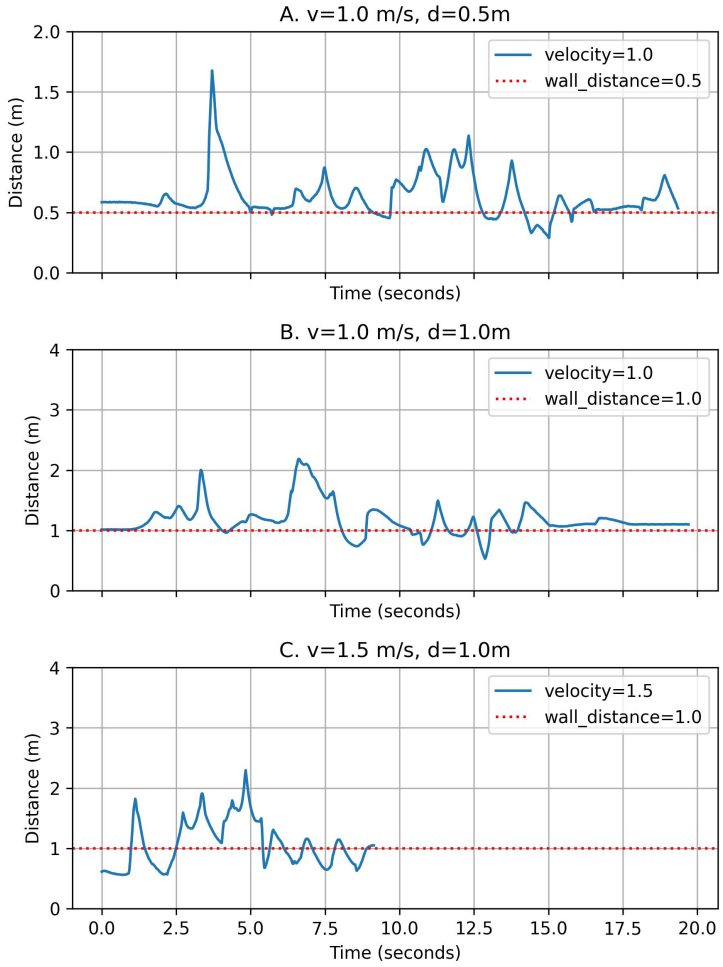

Figure 6: **Stata Wall 1 Trials.** Three different combinations and speed and desired distance. The blue line is actual distance to the wall, measured directly using the LiDAR scan. The dotted red line is the expected distance.

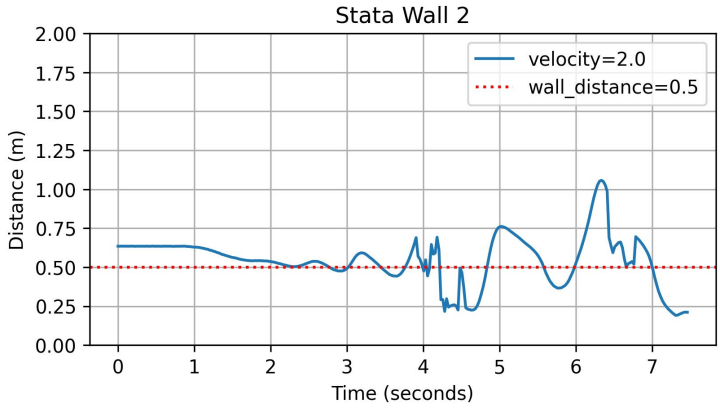

Figure 7: **Stata Wall 2 Trial.** Higher velocity validation trial on Stata Wall 2.

A detailed examination of the recorded RViz data revealed that the oscillations primarily stem from the error term designed to penalize close proximity to a forward wall. Although this term was intended to prompt earlier corrective action on turns, it inadvertently misclassified straight walls as an impending obstacle if the car was at a slight angle to the wall. This misclassification led to excessive corrective behavior, as the controller oscillated between turning actions.

4 Conclusion

Thomas Buckley

In this lab, we successfully developed a robust wall-follower algorithm that maintained accurate proximity to walls under a range of operational conditions—including varying speeds, distances, and environmental setups. Our experiments demonstrated that the system adapts well to real-world scenarios, and the integration of a safety controller further ensured that the vehicle halted reliably when encountering obstacles or impending collisions.

Through tuning our parameters, we observed that while the system effectively anticipates and executes tight turns, it sometimes exhibits oscillatory behavior. This trade-off highlights the challenge of balancing aggressive turning with smooth, stable motion. Future work will focus on improving this behavior in our wall tracking algorithm. This could be done by further refining our PID algorithm, or adjusting how we compute the error term. We plan to improve the algorithm by incorporating a higher-resolution mapping of the wall, since the linear regression provides a very poor representation of more complex environments.

Moreover, to improve the reliability and maintainability of our codebase, we plan to implement comprehensive unit tests for our Python ROS components. For example, these tests will verify the correct conversion of angles and the validity of data ranges, ensuring that downstream processes operate on accurate and consistent inputs. Overall, these refinements will contribute to a more robust and reliable wall-following system that is better equipped to handle dynamic and complex environments.

5 Lessons Learned

Fiona Wang

During this lab, I have learned several lessons both technical and collaborative. From a technical perspective, this lab taught me how to customize a PD controller (eg. adding an extra variable to make tight turns) for different situations. Building off a typical PD controller, we were able to discover what features can be added to enhance our racecar's performance. Additionally, this lab gave me experience using Docker and ROS on physical robot, highlighting challenges that come from migrating from simulation to the real world. There were many elements (such as the impact of driving on carpet vs.concrete) not present in the simulation that affected the effectiveness of our wall follower in real life, making it more difficult to tune our controller. However, we were able to adapt quickly by making use of RViz and learning which environments/hallways were best suited for testing our wall follower. Furthermore, I think I gained a lot of skills and insights working on a team with a wide range of backgrounds. We each had different majors and areas of expertise which came in handy when tuning and debugging our robot. It was really interesting to see how each member had a different thought process and approach to solving problems. I really enjoyed working with my team, and I hope to learn more from my team members throughout future labs!

Ben Lammers

I learned a lot about both how to work with a robot and how to work with a team. This was my first time working with ROS2 on a physical system, and stepping outside the simulation was challenging for me. For one, keeping batteries charged and understanding the new structure of the base code for the robot was more challenging than with the simulation. Despite this, I learned how to setup and operate our robot, modify code and push changes from our robot, and I learned some of the differences between simulations and real-world performance. For example, any of our algorithms would work great in simulations,

but would perform poorly in the real world, and this is likely because of different driving conditions and inherent damping of mechanical steering systems. From a collaborative perspective, I learned a lot about how to communicate and work with others. One of the biggest challenges our team faced was finding times to meet with each other, but by implementing a consensus-decision making framework, we were able to find times and meeting locations that worked for everybody. In addition, everyone had different priorities at the start of our team formation, but we've been able to combine those priorities to set ourselves up for success by being on the same page.

Thomas Buckley

This lab significantly enhanced my ability to work on real-world systems with collaborators. For technical lessons learned, I have previously developed software in a controlled environment on my personal computer, where I could rely on an integrated debugger and a full-featured IDE. Transitioning to embedded systems introduced a host of new challenges. Debugging became more complex, as I had to develop custom scripts to subscribe to variables and navigate the intricacies of ROS2 nodes and topics. Additionally, running a resource-intensive IDE directly on the embedded system was impractical. As a result, I had to quickly adapt by mastering the use of lightweight tools like the vim text editor for robot code development.
I also learned many lessons about working with a team on a challenging software task. I have used git before, but only by myself to track changes. Managing code changes with my collaborators required a more disciplined approach with git, as I needed to efficiently track modifications and synchronize updates between my development environment and the embedded system. I also needed to communicate with my team members about how to deal with merge conflicts (for example, which component of each algorithm would we want to keep).

Sriram Sethuraman

I learned a lot of things working on this lab, including how to work with physical hardware, ROS, and teamwork. Robot-wise, I learned how difficult it is to tune a system in the real world. In simulation, there were a lot of variables that weren't accounted for, such as the inertia of the physical robot, external forces such as friction, and the mechanical limitations of the robot. Accounting for these factors and tuning the system was a very time consuming process. I also learned about my teammates and how to work with them and I feel we've achieved a good team dynamic. Communication wise, I learned the importance of visuals in a presentation, as they can add a lot. I look forward to working on more labs with this team!