Lab 3 Report: Wall Follower and Safety Controller

Team 12

Angel Gomez
Ariel Fuchs
Gokul Kolady
Sharmi Shah

6.141

March 5, 2022

1 Introduction (Author: Angel, Editor: Sharmi)

In this lab, we were introduced to the physical racecar platform for the first time, with this also being the first project we all worked on as a team together. This introduction served to familiarize ourselves with the combination of hardware and software testing, and it gave us a few simple yet important goals to meet. The first of these goals was setting up our connection and method of communication with the racecar hardware, and forming an understanding of how to operate the racecar. The second of these goals laid in developing our capability to visualize the data off of the sensors mounted on the racecar, namely the LiDAR scanner, in order to use it for navigation purposes.

The third goal centered around implementing a version of the wall following code that each of us created in Lab 2 into the physical racecar, in which we gave a virtual racecar the ability to follow a wall at a desired distance. This was done utilizing the data obtained by the visualization of the LiDAR scanner, a set of equations to calculate the distance of the racecar to any potential obstacles, and a PID controller to evaluate this data and generate the necessary navigation messages to accomplish this task. Translating these scripts from simulation to hardware reliably required the testing of the scripts, all of which are described in more detail below.

The fourth goal revolved around the implementation of a safety controller for our racecar to reduce the risk of damage throughout the rest of the projects we'll end up delving into. This aspect is especially crucial, as these racecar platforms are very expensive (in the $>$ \$4k range) and cannot be easily and quickly replaced for the future projects our team will work on for this class. This safety controller was applied with an average range found within a slice of the LiDAR data to obtain the distance of the racecar to a certain obstacle, and a simple bang-bang controller that stops the car depending on the distance the car is from the obstacle. More detail on the approaches taken to implement this controller and results can be found in the following sections.

Finally, the last goal boiled down to understanding and creating our team dynamic. As a newly formed team, it was crucial for us to form our own workflow path in order to be able to meet the deliverables effectively and on time. Thanks to messenger chats and frequent update meetings, we have been able to find our balance as a team, which we can carry forward onto future labs.

2 Technical Approach

2.1 Wall Follower (Author: Gokul, Editor: Angel)

The following section describes our team's technical approach and decision making regarding the wall following module we were tasked with creating. This was

the first of the two major modules we aimed to implement in this lab.

The first technical challenge that we aimed to tackle in this lab was creating a wall-following module for our race car. The goal was to be able to have the race car autonomously drive at a set speed alongside either a leftward or rightward wall, while maintaining a set distance from the given wall as accurately as possible. This wall following module had to be robust when presented with corners, turns, and protrusions along the wall. Furthermore, it ideally had to work at varying driving speeds, as the module will be used in future race car projects to assist in accurate movement.

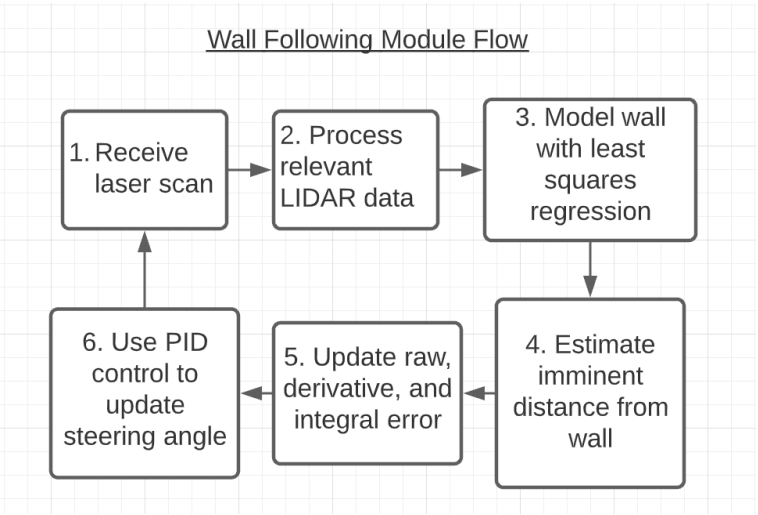

Figure 1: Shown here is the high level ROS implementation flow for the final wall following module our team used.

1. Receive laser scan: The wall following module is subscribed to the wall following package's laser scan topic. Once the module receives a laser scan message (LIDAR data) from this topic, it executes a callback function that processes the data and steers the robot accordingly.

2. Process relevant LIDAR data: Once a laser scan is received, the module slices out the relevant LIDAR data based on the wall it is assigned to follow. Each individual scan in the LIDAR data has a corresponding angle in radians and distance in meters. If the right wall is being followed, it keeps the scans from the angle range of $\frac{-4\pi}{8}$ to $\frac{\pi}{8}$, where 0 would be forward and negative angles are to the right. If the left wall is being followed, the angle range selected is instead $\frac{-\pi}{8}$ to $\frac{4\pi}{8}$. The scans that have a distance value greater than the mean of all sliced scan distances plus 1m are thrown away, as these are usually caused by holes/gaps in the wall that only serve to add noise to the wall estimation process. Next, the angle/distance pairs are each converted to rectangular (x, y)

coordinates using the following formulas:

$$x = distance * cos(angle) \tag{1}$$

$$y = distance * sin(angle) \tag{2}$$

3. Model wall with least squares regression: Now that the module has selected relevant scans based on the wall it is attempting to follow in rectangular coordinate form, it performs least squares regression to create a best-fit line that approximates the wall's location on a rectangular coordinate plane (with the robot at the origin, pointing in the positive x direction). The slope and intercept for this line are calculated as follows:

$$slope = \frac{\sum_{\forall i}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{\forall i}(x_i - \bar{x})^2} \tag{3}$$

$$intercept = \bar{y} - slope * \bar{x} \tag{4}$$

4. Estimate imminent distance from wall: The module then uses the robot's velocity to estimate where the robot will be on the coordinate plane in 0.2 seconds, assuming the robot will move straight along the positive x-axis. With this computed position and the previously calculated best-fit line for the wall, the module estimates the near-future distance from the wall with the following equation:

$$wall\_distance = \frac{abs(slope * x_{future} + (-1) * y_{future} + intercept)}{\sqrt{slope^2 + (-1)^2}} \tag{5}$$

5. Update raw, derivative, and integral error: Next, the module computes the raw, derivative, and integral error for the current time step to feed into the PID controller as follows:

$$dt = t - t_{previous} \tag{6}$$

$$error = wall\_distance - DESIRED\_DISTANCE \tag{7}$$

$$derivative = \frac{error - error_{previous}}{dt} \tag{8}$$

$$integral = integral_{previous} + error * dt \tag{9}$$

6. Use PID control to update steering angle: Finally, these values are used to update the robot's steering angle using the following PID gain constants and update equation:

$$K_p = 7.0 \tag{10}$$

$$K_i = 0.086 \tag{11}$$

$$K_d = 0.094 \tag{12}$$

$$steering\_angle = SIDE * (K_p * error + K_i * integral + K_d * derivative) \tag{13}$$

where $SIDE = -1$ when following the right wall and $SIDE = 1$ when following the left wall. The idea is that these different forms of error represent how accurately the robot is maintaining its goal distance from the wall, and based on this the PID controller can guide the robot to turn in a way that reduces these forms of error, resulting in accurate wall following. This control scheme is shown in Figure 2.

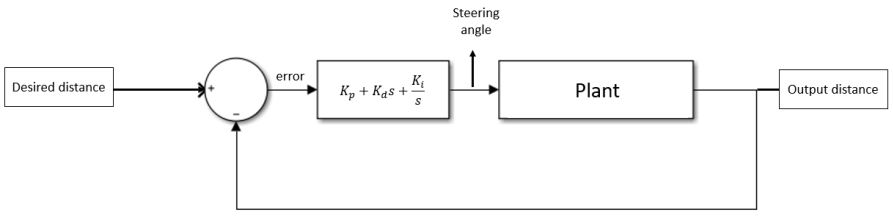

Figure 2: Shown here is the control flow for the PID steering update process.

Major Design Decisions

The first major design decision we made was switching from a PD controller to a PID controller. Originally, we were using a team member's code that had implemented PD control. While this worked decently well when executed on the real-world robot, during testing we found that the error (actual distance from wall vs. desired distance from wall) would often settle at a non-zero but small-magnitude amount. We realized that this was likely because we had no integral component to our controller's algorithm, meaning there was nothing in the wall following module penalizing a steady-state error. Thus, we decided to switch to another team member's code that utilized a PID controller. We noticed that after this change, the robot tended to reach zero-error more consistently during periods of wall following.

Another important design decision was setting the values of the proportional, integral, and derivative gain constants for the wall following PID controller.

This was done in simulation using the Ziegler-Nichols Tuning Method. First, the K_p was increased until the robot started to oscillate its turning near the wall. This resulted in a critical gain value of $K_C = 3.82$. Next, the period T_C of this oscillation was timed. This resulted in a value of $T_C = 0.3$. Finally, using these values, the actual gain values for the PID controller were set to $K_p = 0.6 * K_C$, $K_i = 2 * K_p/T_C$, and $K_D = K_p * T_C/8$. The robot was next tested in simulation and the turns were not aggressive enough, so the K_p was raised gradually until the turning was sharp but not over-aggressive. The final K_p value was 7.0.

2.2 Safety Controller (Author: Ariel, Editor: Gokul)

The next section describes our team's technical approach and decision making regarding the safety controller we implemented. We created a safety controller in order to make sure that our robot would not get damaged or damage other obstacles —including people — while running any scripts. We needed to create a script that would override any running code at a moment's notice if something obstructed the robot's path. Therefore, we designed our robot to completely stop if there was anything in front of the car within 0.9 meters.

Approach 1: Linear Regression

1. Calculating the distance between the vehicle and any objects using linear regression. Our first approach in creating a vehicle that would stop within our Hard Stop Distance, 0.9 meters, of an obstacle was composed of two parts. The first part was reading in LIDAR data from the SCAN_TOPIC. The SCAN_TOPIC data returns arrays of angle LIDAR readings, which we then sliced in order to measure the relevant range, [-pi/8:pi/8], for our safety controller. The range [33:66] encompasses the LIDAR data that is relevant to tracking obstacles in front of our vehicle. Using Equations 1 through 5 (except now in equation 5 we use (0, 0) instead of (x_{future}, y_{future})), we were able to calculate the Car to Wall Distance.

2. Determining when to stop the vehicle. The second part of our safety controller was understanding what to do given the distance of the car to any obstacle. If the car was 0.9 meters or further, the car would continue its current trajectory. However, if the distance from the car to an obstacle was less than 0.9 meters, the car would immediately stop. In the former case, our script does not interfere with any commands as the car has not detected any dangers. In the latter case, we set the drive speed to zero and publish that value to the SAFETY_TOPIC using an AckermannDriveStamped message.

3. Drawbacks and Improvements with Approach 1
Although our vehicle was able to successfully pass our initial tests, we realized that there would be edge cases that our car would not pass. Since we were using a linear regression algorithm, if there was a thin object in front of the car, such

as a pole, that shorter distance read would be generalized with the rest of the data, and the car would run into it! This edge case is likely so we needed a better approach to make our safety controller robust.

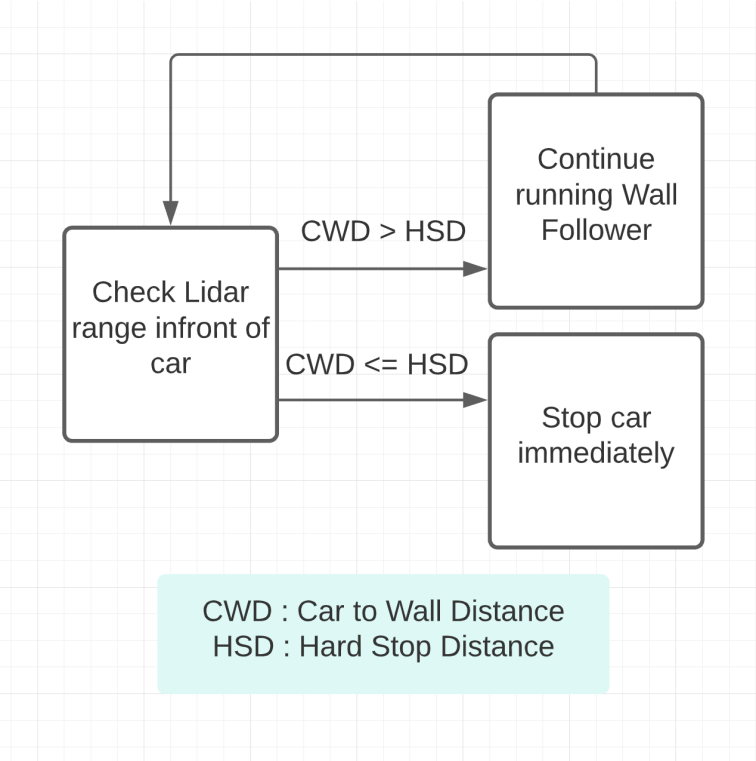

Figure 3: Shown here is the high level ROS implementation flow for the Safety Controller module our team used. Car to wall distance is always being recalculated while the Hard stop distance is set to 0.9 meters.

Approach 2: Averaging Data Points

1. Improvements to Approach 1. Our second technical approach was similar to Approach 1: Linear Regressions, but we now changed which values we were comparing against our threshold Hard Stop Distance. We made an adjustment to the first part of our safety controller, while keeping the second part of our safety controller the same. For the first part of our safety controller, we continue to utilize the data from SCAN_TOPIC within the range of [33:66]. Instead of creating a linear regression from the relevant data points, we determine

the data point with the shortest distance from the car to an object.

2. Drawbacks and Improvements with Approach 2
Approach 2 addresses the flaw with our initial approach: even if there are only a few data points in closer range, the car will still stop so that it will not collide with anything. Our car passed all of our initial tests and will be able to pass foreseeable edge cases. However, we are considerate of the drawbacks and understand that improvements will still be necessary for what values we use to get our Car to Wall Distance and what value we should make our Hard Stop Distance.

a) Improvements for Car to Wall Distance. Data can have a lot of noise which can give accidental readings. If a data point read is really close in one instance but disappears in the next, then it is unlikely that there was ever an obstacle in the way. Regardless, our vehicle will stop if the Car to Wall Distance calculated to equal or be less than our Hard Stop Distance. Trusting a single data point to determine what our autonomous vehicle should do will not always be sufficient. Therefore, an improvement we can make to our safety controller is monitoring multiple points that are calculated to be the shortest. If the amount of shortest points is above a determined threshold, we can trigger the car to stop. Another improvement we could make is tracking the shortest distance data point to make sure that it is consistently short. We would keep a variable to monitor if this point has consistently been getting closer, or if it was just in a single instance. The last improvement we could make is finding the data point with the shortest distance and check the adjacent LIDAR data point to check for a thin object.

b) Improvements for Hard Stop Distance. Although our Hard Stop Distance is set at 0.9 meters, we understand that car momentum will vary with its velocity even if the car is suddenly told to stop. Therefore, 0.9 meters should give us enough buffer space to not collide with an object even if it was driving very fast. In the following section we will address and analyze the data we gathered after implementing our Safety Controller to reanalyze this point. If there is a noticeable difference in final stopping distance based on its velocity before stopping, a design consideration we could implement is having the Hard Stop Distance vary with the speed of the car. These improvements can be implemented on their own or combined, and will be implemented by Lab 4.

Priorities

How does the safety controller keep the car safe while other commands are also being sent to the car? The safety controller is able to override other scripts because of its priority order. The highest order is Teleop, then Safety, then Navigation. Any commands given from Teleop — the joystick controller — will override commands from lower order topics: Safety and Navigation. Similarly, the Safety topic will be able to override any commands from Navigation. This means that if the wall follower script is running from the Navigation topic and

the car suddenly comes too close to an obstacle, the safety controller will kick in and stop the car. Within each priority level, there can be layered structures. For example, if we created a second safety controller script that stopped the car whenever it saw a red object, this code would overrun any code from the Navigation topic. Yet within the Safety topic, the script that would take precedent would depend on the ordering.

3 Experimental Evaluation (Author: Sharmi, Editor: Ariel)

3.1 RVIZ Visualization

After setting up the racecar's hardware, the LiDAR was visualized using Rviz. As seen in Figure 4, the team was able to successfully visualize the surrounding walls as seen by the colored lines and the IMU heading seen by the purple arrow. This functionality was helpful in understanding how the racecar was viewing its environment and assisted in future debugging.

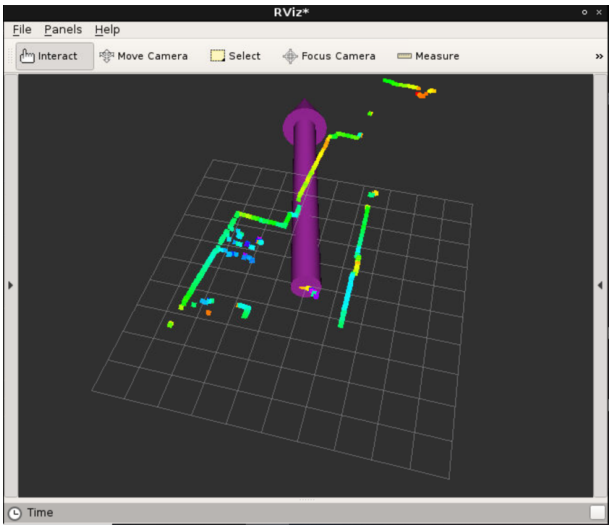

Figure 4: Rviz visualization of LiDAR scan data and IMU heading (purple arrow).

3.2 Wall Follower

The first goal for the Wall Follower was to demonstrate the simulated success of the wall following code on the physical robot in a real environment. The first test performed on the physical robot was with a PD controller and demonstrated a qualitative proof-of-concept that the robot could follow a wall.This video shows

this initial proof-of-concept. Although the robot slightly bumps into the wall while turning a corner, it otherwise successfully follows the wall at the desired distance. The next sections describe further tests that were performed and modifications that improved the performance of the wall follower.

3.2.1 Testing Plan

Several trials were performed to evaluate the performance of the wall follower as seen in Table 1. As a result of extensive simulated experimentation, and tuning of controller gains in the simulation, tuning of the controller on the physical robot was not required for the wall follower to perform successfully in these trials. A few key parameters to test on the robot to substantiate the functionality of the wall follower design were different robot speeds, wall sides (right vs left), and types of walls. If the robot was able to perform well in settings, it should theoretically be able to wall follow in new environments at various speeds.

Table 1: List of tests performed for Wall Follower

Test	Controller Type	Speed ($\frac{m}{s}$)	Wall Side	Wall Type
1	PD	0.5	left	straight wall
2	PD	0.5	left	curvy wall
3	PID	0.5	left	combined
4	PID	1.25	left	combined
5	PID	0.5	right	combined
6	PID	1.25	right	combined

3.2.2 Performance Metrics

The specifications for the wall follower are to (1) avoid collisions with the wall, (2) maintain the desired distance from the wall, and (3) subside excessive oscillatory behavior. The main quantitative performance metric used to evaluate the first and second specifications was average error which is defined as

$$error = \frac{1}{n} \sum |distance_{desired} - distance_{true}|$$

The robot was also qualitatively evaluated for the first and third specifications. If in any trial, the robot displayed excessive oscillations, it was recorded.

3.2.3 Testing

The first two trials with PD control are visible in Figure 5 which show more error for the curvy wall when compared to the straight wall. Furthermore, there is a steady state error seen in the first trial when the robot is following a straight line because there is no integral control. Although there were no bumps or collisions with the wall, the PD controller had strong oscillations in certain

cases, specifically if the robot was initially placed far from the desired distance setpoint.

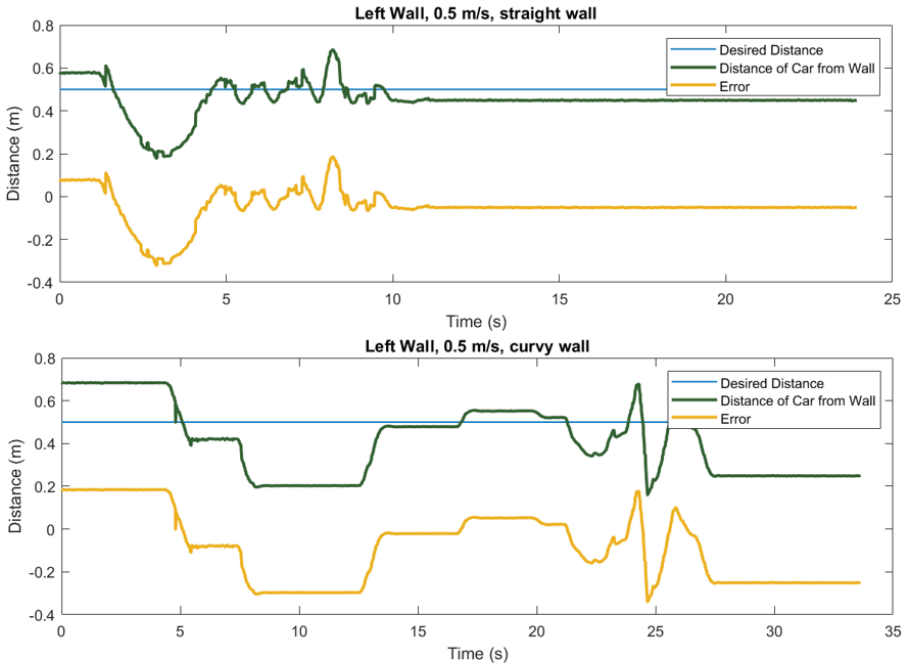

Figure 5: Trials 1 and 2 in order. High error values in parts of the plot deem PD control as satisfactory.

The next four trials were with PID control and are visible in Figure 6. Qualitatively observing the videos for trial 3, trial 4, trial 5, trial 6 showed that this controller resulted in subtler oscillations compared to the PD control. Furthermore, the controller was robust against more complex hallways as seen in the videos. Figure 4 shows that error increases as velocity increases because there is more overshoot and the greater momentum of the robot creates an a greater lag between the controller's instructions and the robot's adherence to those instructions. Unlike initial tests with PD control in which the robot collided with the wall, none of the PID tests resulted in a wall collision. Comparing Figures 5 and 6 shows that the PID controller is able to keep the robot centered about the setpoint, despite oscillations, whereas the PD control is not.

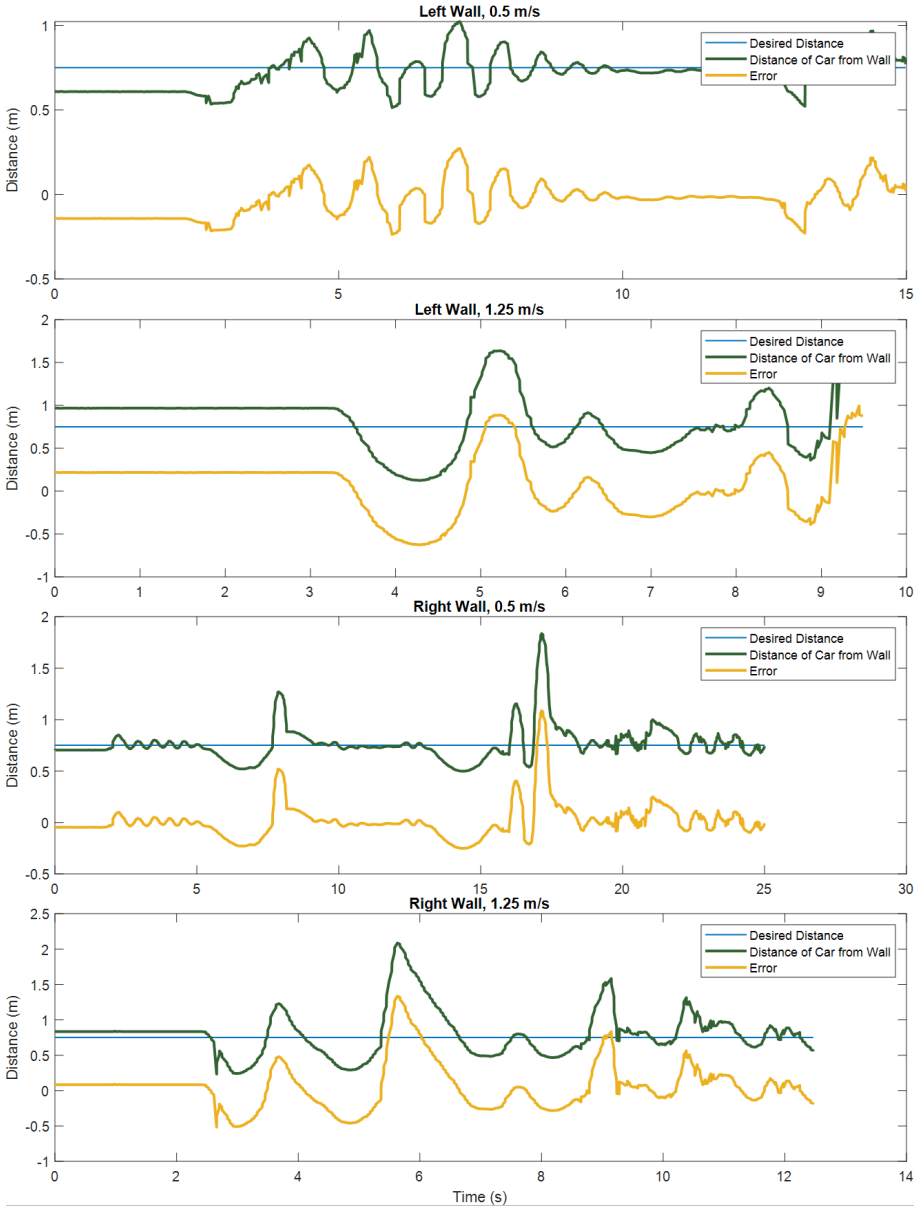

Figure 6: Trials 3-6 in order. The PID controller successfully demonstrates desired wall follower specifications.

Table 2: Results for all wall following trials.

Test	Controller Type	Speed ($\frac{m}{s}$)	Average Error (m)
1	PD	0.5	0.0679
2	PD	0.5	0.1547
3	PID	0.5	0.0871
4	PID	1.25	0.2845
5	PID	0.5	0.1019
6	PID	1.25	0.2347

The average error in Table 2 shows that the robot has more difficulty following the wall when it has a greater velocity. Although the PID controller trials have greater errors in the table, it was following more complex walls and was able to follow them successfully as opposed to the PD controller which in many trials (not visible here) stopped following the wall altogether or was unable to make a turn and instead almost crashed into the wall.

3.2.4 Performance Assessment

Overall, the PID controller had a great performance and met all three specifications described. It was able to avoid colliding with walls, handle turning walls, and had subtle oscillations.

3.3 Safety Controller

The goal of the Safety Controller was to avoid colliding with the wall if the wall follower failed or any other sudden obstacle came within the a close distance of the robot's direct path. Two types of safety controllers were tested as described in Section 2.2. The first safety controller fitted a line to a range of points in ahead of it and the second averaged the range of points ahead of it to dictate whether the robot should stop.

3.3.1 Testing Plan

Trials performed to evaluate the performance of the safety controller are seen in Table 3. Having the robot stop when it is traveling at different speeds or if an obstacle is suddenly placed in front of it were two key parameters that were tested. Varying these parameters to test the robot should allow the safety controller to work well outside of this testing environment.

Table 3: List of tests performed for Safety Controller

Test	Controller Type	Speed ($\frac{m}{s}$)	Setpoint (m)	Sudden Obstacle?
1	line fitting	0.25	0.9	no
2	line fitting	0.75	0.9	no
3	line fitting	0.75	-	yes
4	averaging	0.25	0.9	no
5	averaging	1.00	0.9	no
6	averaging	0.70	-	yes

3.3.2 Performance Metrics

The specifications for the safety controller are to (1) stop within a given distance from the wall at different speeds and (2) stop if any obstacle is placed in the path of the robot. If the robot is able to perform well in the previously stated tests, it should be able to prevent wall collisions in other environments. For non-obstacle trials, the quantitative performance metrics used to evaluate the specifications were the time it took the robot to stop after given the command to stop and the steady state error defined as

$$error = |distance_{setpoint} - distance_{final}|.$$

For obstacle trials, the robot was qualitatively evaluated to ensure it did not collide with the obstacle.

3.3.3 Testing

Initial testing of the first safety controller showed that the inertia of the robot along with some small lags in the ROS environment caused the robot to continue moving past the given setpoint. The first two trials with the line-fitting safety controller are shown in Figure 7. The plots show that even after the robot is given the command to stop, as indicated by a 0 speed value, the distance to the wall still changes. This substantiates the point made above about the inertia of the robot causes a delay in the robot stopping.

As seen in table 4, it takes longer for the robot to stop at higher velocities. This means the safety controller may be modified in the future to have higher setpoints if the robot has a greater velocity to prevent it from overshooting too much and crashing into a wall or other obstacle. It also brings up one limit of the robot; that is, if a sudden obstacle is placed close to a robot moving at a high velocity, there is no way to stop the robot from colliding as a result of the time lag.

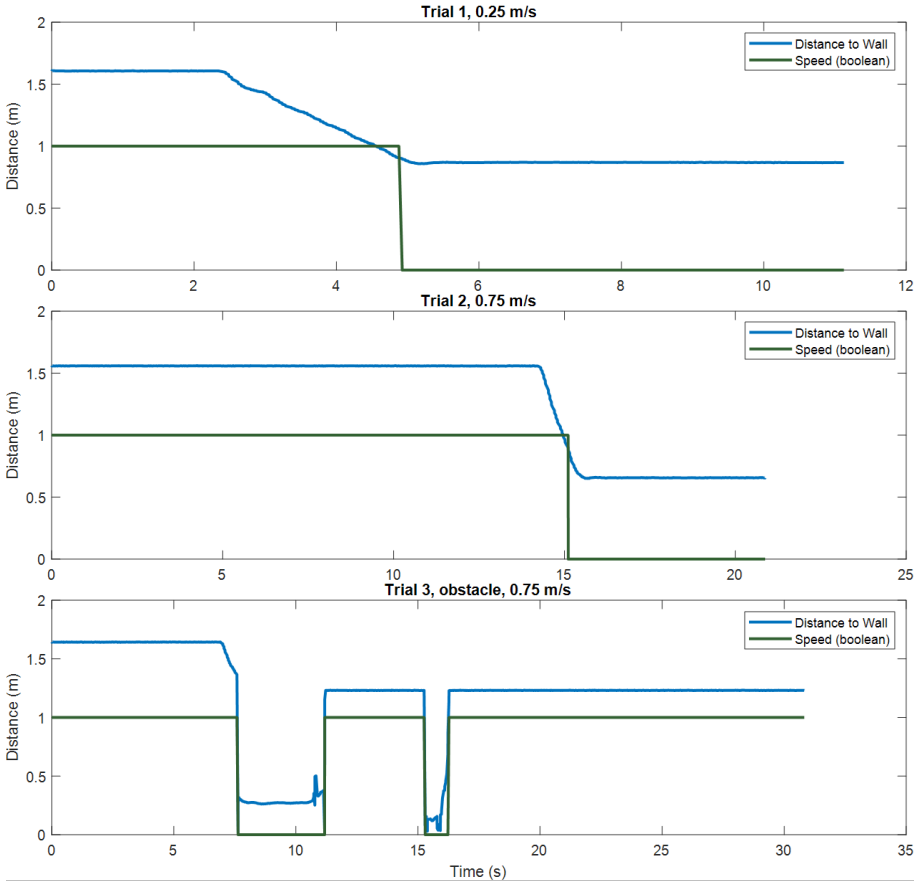

Figure 7: Trials 1-3 in order, demonstrating performance of the first version of safety controller. Note: the speed values are boolean, meaning the speed=1 when the robot is moving and 0 when the robot is given the command to stop.

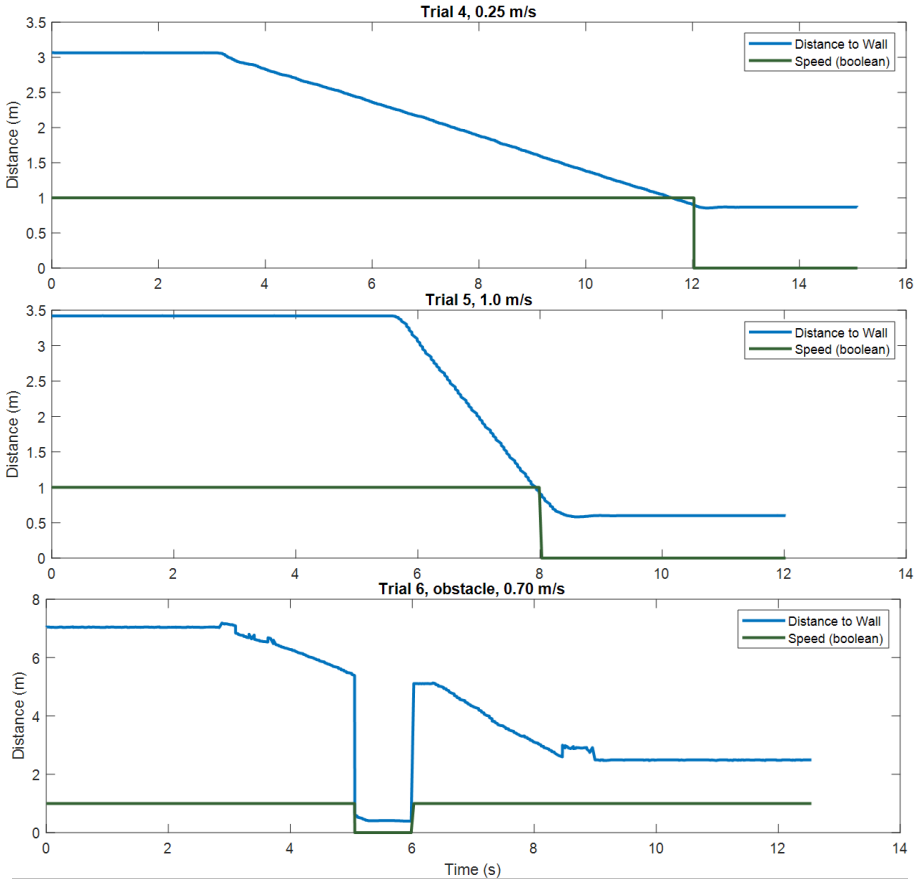

Figure 8: Trials 3-6 in order, demonstrating performance of the second and final version of safety controller. Note: the speed values are boolean, meaning the speed=1 when the robot is moving and 0 when the robot is given the command to stop.

Table 4: Results for non-obstacle safety controller trials.

Test	Setpoint (m)	Final Distance (m)	Error (m)	Time to Stop (s)
1	0.900	0.868	0.032	0.49
2	0.900	0.656	0.244	0.71
4	0.900	0.868	0.032	0.36
5	0.900	0.600	0.300	0.75

Qualitatively watching trial 3 and trial 6, one could see that the robot stopped in time to avoid colliding with the obstacle which was seen as a success for the robot. As seen in the video for trial 6, the robot was not only able to stop when the obstacle was placed in its way, but it was also able to carry on once the obstacle was removed.

3.3.4 Performance Assessment

The safety controller was able to successfully meet its specifications. Although the two types of controllers had similar results in error and time to stop values (table 4), the second controller is more effective in theory because the line fitter may ignore a small obstacle that is directly in front of it by ignoring those points as outliers, whereas the averager will pay more attention to these values.

4 Conclusion (Author: Angel, Editor: Sharmi)

Overall, we were able to either meet all of the goals we set ourselves up for this lab or highlight key points to improve as we move forward with future projects. Let us run through the report with the same order as we visited them during the introduction. First off, we were able to consistently connect to our racecar utilizing our computers, and seamlessly edit files within its environment and transfer them between our computers and the racecar. With this skill, our team will be able to more quickly make changes to the behavior of our racecar with a fewer amount of errors. Secondly, we were able to visualize all the data from the LiDAR scanner and utilize it effectively in our scripts. A good note for this in regards to future projects, however, can be to find more efficient ways with dealing with the noisiness of the data.

We were able to successfully implement the wall follower via improvements on all of the scripts we had previously written. The addition of an I component to the PID controller allowed the whole process to be more robust; having no crashes occur at both fast and slow speeds. For the gains however, more tuning could be done to reduce the error even further. Furthermore, we were able to implement the safety controller successfully, and tested it in a variety of orientations and speeds. The robot was successful in stopping before reaching the wall in all these tests. Something that was discovered through this testing, however, was the effect of speed on the racecar's stopping power; whenever the racecar had a faster speed, it stopped closer to the wall than the 0.9m boundary that we had set for it. This could point us in the direction of changing this hard coded boundary into one that's more velocity dependent, so as to increase our trust in the robustness of the safety controller for future labs to come.

Finally, given the time we spent together working and the ease at which we set up the communication channels within us, this lab served its purpose very well as an opener to our team's dynamic, wherein we were able to effectively find our rhythm in dividing up tasks amongst ourselves and working to complete all the goals set in the time allotted. The habits that were formed in the scripting, tuning, and debugging stages of this lab will serve as guideposts for us to refer to for future labs, and will end up helping us to work quicker and better in the long run. Without a doubt, this team was able to find its footing and flow thanks to this lab, and it will be able to more efficiently complete the tasks that

future labs may put on our plate.

5 Lessons Learned

Ariel: Both the Wall Follower and Safety Controller labs gave me a lot of new skills! I learned what LIDAR data is and how to use it for different purposes. I can calculate distances between the car and any objects within the range of the LIDAR scan. Furthermore, I now understand Proportional Derivative controllers and have successfully implemented it twice. Moving forward, I am going to make sure that I focus on what each Publisher and Subscriber does. When initially creating the Safety Controller, I added unnecessary subscribers and published to the wrong topics. I caught my mistakes by rereading the specs on GitHub, walking through my code with my teammate, and printing out messages to see what I was receiving and sending. These tools are good practices and will speed up easily fixable errors in the future.

Angel: I can certainly say that this lab gave me a new set of skills that will really help me moving forward. I was able to have experience with SSH-ing into other machines and applying changes from my own terminals. Furthermore, this lab helped further my understanding of how PID controllers work, and their effectiveness, which will help me better identify cases in which one of these controllers can improve a system I'm working on. Moving forward, this lab taught me the importance to be aware of all of the files you're talking to and using (bug came specifically from the topics in the .yaml files), as the bugs you're trying to fix might not necessarily be in the file you're currently looking at. Also, it helped me refine my sense of identifying syntax errors within my code in this ROS environment.

Gokul: This lab was very informative on a technical level. Firstly, I became a lot more comfortable and familiar with how to interface with the robot framework. For example, now I can quickly and efficiently copy code onto the robot, execute code on the robot, and run modules on the robot. Furthermore, I have learned that in robotics, testing and debugging code on a live robot is a lot more difficult than doing the same things in simulation. It requires more thoughtful scheduling of work. You have to be mindful of how hardware constraints (ie. robot battery, motor battery, router service range) effect when and where you can code. Furthermore, because there is only one robot available for testing at any given time, I realized that communication is extremely key to getting work done in parallel. If one sub-team is testing one module on the robot, it is ideal that another sub-team is either working on coding a different module or documentation rather than waiting to test another module. Finally, I learned that teamwork is crucial to getting anything done, especially in terms of robot experimentation. It is impossible for one person to run code, record the robot,

record rosbags, and be in charge of the Logitech controller if things go wrong.

Sharmi: This lab was integral to becoming comfortable and efficient at navigating the workflow for connecting to the robot and performing experiments with it. The lab also helped me learn how to use ROS without being told exactly what files or simulations had to be performed. I learned how to use ROS as a resource to help succeed at our goal, which involved creating our own nodes, rosbags, and associated files. One major aspect that I improved on during this lab was debugging and understanding what consequences result from which mistakes so that I know the steps I need to take to fix problems I see in the future. Another thing I learned during this lab was how important teamwork is and how important a diverse skill set it to accomplishing a task. Being able to split the tasks was really useful, especially when we all couldn't be in the same location to work on the lab. One thing we could improve on is starting the lab earlier so that we have enough time to prepare for the presentation of our work.