Team 15 + Irene Terpstra, Miles Silva, Matthew Leonard, Lili Sun
6.141
March 4th, 2022

LAB 3 REPORT: Wall-following Racecar

1 Introduction (Matthew)

In this lab, we adapted our wall-following code from Lab 2 to work on a real robot. In that lab, each of us built a ROS node that listened for LIDAR data, used it to guess the location of a wall on one side of the robot, and gave steering commands to the robot. It used a Proportional-Derivative (PD) controller to issue a command based on the robot's distance from and angle relative to the wall. The specification for this lab was to produce a racecar that could not only follow a real wall autonomously, but that could also be controlled by a human, and that would stop automatically for obstacles in as many testing scenarios as possible.

Although our wall-follower could pass each of the tests in the simulation for Lab 2 with a high score, we still needed to modify the code to function in a real-world environment, to handle the deviations from the simulation caused by sources of error in the robot components. We also had to modify our code to ensure it was robust to more different wall shapes, not just the ones in the tests. We organized our wall-follower, our safety controller, and a provided tele-operation controller into a multiplexed input system wherein different controllers can each be given a priority level and can override each other. This allowed us to switch between automatic and manual control smoothly, and to implement safety overrides when necessary. We systematically tested our robot by developing numerous test scenarios for both wall-following and collision avoidance. We implemented continuous error tracking, and used this to analyze and graph our robot's performance in each scenario we came up with.

By running our wall-follower code on a real robot and modifying it as necessary, we made significant progress towards implementing a fully autonomous system. The ability to follow a wall automatically will be a fundamental skill for our robot competition, akin to the ability to run in a straight line for a road race. The multiplexed control system will make it easy to abstract away different levels of control. This means that we can easily implement our

wall-following code, which is complex and low-level, into a higher-level strategic control system. By splitting each segment of the controller into a separate module, we can swap modules in and out. For example, if we switch to a more sophisticated wall-following algorithm, such as pure pursuit, we can implement it with the same interfacing specifications as our original controller and swap it in with little or no modification to our other components. This is consistent with the ROS design philosophy, which encourages modularity.

In addition, we have learned how to filter sensor data and implement and tune a PID controller, which will be useful skills as we progress to more complicated sensors, like the camera.

2 Technical problem description

2.1 Wall Following (Miles)

The purpose of this portion of the lab was to design and implement an algorithm that would allow a robot to follow a wall on a given side, at a given distance, and at a given speed. We had access to a Velodyne LIDAR sensor and were able to control the robot's motors by publishing Ackermann drive commands. We had no access to any other input or output system. The robot needed to follow a variety of walls, straight or curvy, and turn sharp corners.

We had already implemented code to solve this problem in a simulation, however, porting this code to a real-life system proved to be a challenge. Most of our problems stemmed from the Velodyne LIDAR, which proved to be much less reliable than the LIDAR in the simulation. It provided data at inconsistent rates, which caused the data to flicker. Also, it has a minimum scanning distance, meaning that any objects or points on a wall within about 0.4 meters from the sensor cannot be read by the LIDAR. Even when objects are farther than this minimum scanning distance, the sensor sometimes fails to recognize them consistently, returning garbage data (usually returning the point as "infinite" distance away) at unpredictable time points. Finally, the LIDAR reports all data with a 60-degree offset. These were all problems we had to take into account while implementing our algorithm.

2.1.1 The algorithm

While designing our wall-following algorithm, we needed to solve two independent problems: detecting the wall and controlling our robot to always be at the correct distance from the detected wall. These algorithms will be explored in detail in the following sections.

2.1.1.1 Wall Detection

Our robot first rotates the data we receive from the LIDAR by 60 degrees to account for the constant offset error in the data. Then, we segment the data into one of two partitions, depending on the side of the car we want to follow the wall on. The partition consists of all the data on an angle sweep of 120 degrees starting from perpendicular to the side of the car. This can be seen in Figure 2.1, which depicts the angle sweep we look at if we want to follow the wall on the left side of the car. We chose to ignore the data points behind the car, as well as on the opposite side of the car, because they are irrelevant to following the wall while the car is moving forwards. However, we chose to include some extra data points in front of the car to make sure it can handle turning corners as well as possible.

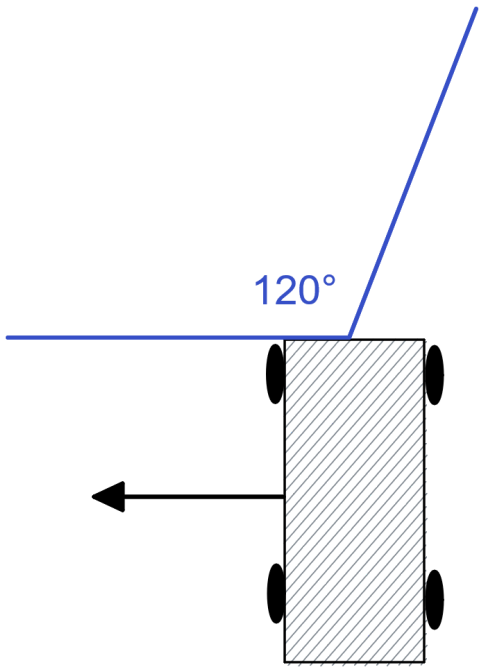

Figure 2.1: We look at a 120° angle sweep on the desired side of the car (in this case, the left side).

Before we can use these data points to detect the wall, we need to first filter out bad data from the LIDAR as well as eliminate any outliers in the data, which can be caused by small holes or confounds in the environment or more typically by noise from the LIDAR itself. To do this, we removed any garbage (“inf”) data from the laser scan, then removed any points that fell too far outside the median of the data. If this resulted in all points being removed, we can conclude that the LIDAR has given us entirely useless information, so we can safely drop the data and wait until receiving new data before performing any new calculations. We found that this method of removing outliers performed best in removing noise from the data, while not being too aggressive and removing necessary points such as readings from the wall in front of the car.

The final wall detection was done by performing a linear regression on the outlier-removed data after transforming the data into the robot’s cartesian coordinate frame. This provided a simple representation of the wall that was easy to perform distance and other calculations on, and provided an easy way for the robot to turn corners, as the line would slowly turn from the wall on the side of the robot to the wall in front of the robot as it approached a sharp corner. Figure 2.2 demonstrates this behavior.

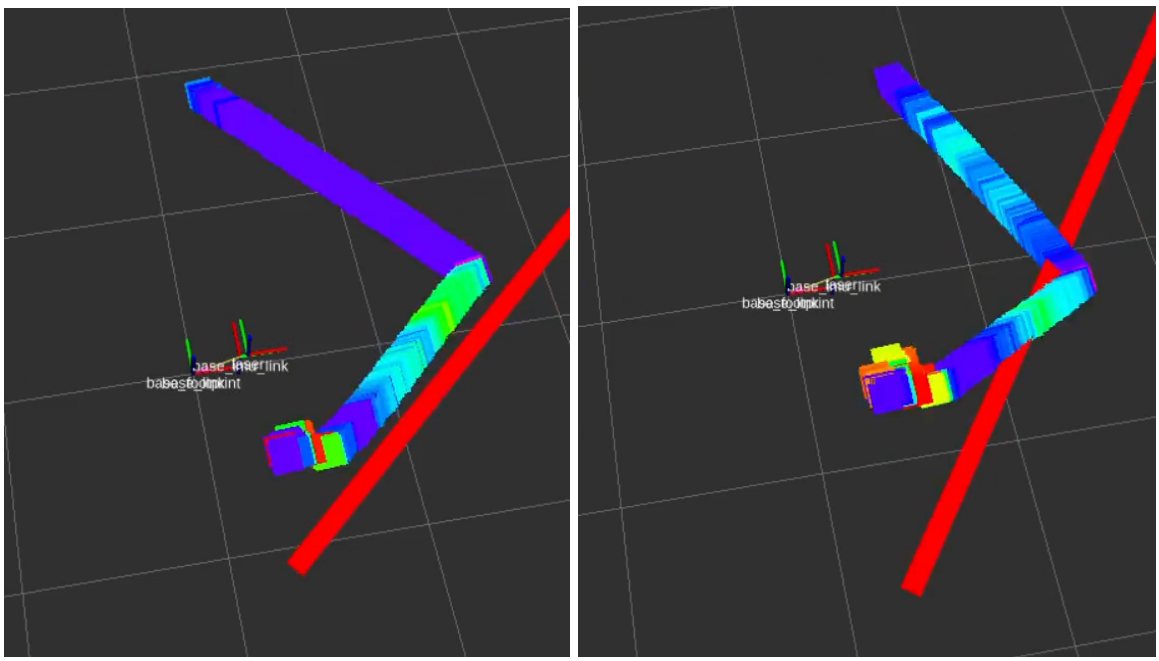

Figure 2.2: As the robot approaches a corner, the line representing the wall automatically adjusts to follow the corner.

2.1.1.2 Robot Controller

To control the robot to follow the wall, we implemented a PD controller. The error for this controller was calculated using equation 2.1, which is essentially how far the robot is from the desired distance from the wall. We augmented the derivative term of the controller with the slope of the wall in the frame of the robot; when the robot is parallel to the wall, this term is zero, and it increases as the robot turns away from the wall. We tuned the controller manually until the robot performed at the desired accuracy. The final steering control formula is given by equation 2.2.

```
error = np.abs(b)/np.sqrt(m**2 + 1) - self.DESIRED_DISTANCE
```

Equation 2.1: The error is the distance to the wall (which is represented as a linear function with slope 'm' and y-intercept 'b') minus the desired distance to the wall.

```
steering_angle = p_1 * dist * sign(b) + p_3 * d_error + p_2 * m
```

Equation 2.2: The final PD controller equation, where our tunings are: p_1=2.9, p_3=0.1, and p_2=2.

We found this PD controller performed with suitable accuracy, and the simplicity of it has the distinct advantages of having a relatively quick implementation time, more flexibility when porting to future projects, and makes adaptation and bug fixing easier than more complicated approaches. For these reasons, we chose not to pursue the implementation of more complicated and unnecessary algorithms for this lab.

2.1.2 Implementation in ROS

The entirety of our wall-following code is written in Python 2. Our wall detection code subscribes to data that is published from the LIDAR and processes the data using the number processing package numpy. Using this package, we were easily able to code the implementation of the algorithm described in section 2.1.1.

One important aspect of our implementation to note is that our code creates and publishes a driving command every time it receives relevant data so that the robot is able to quickly respond to any new data it receives. This driving command is published to the 'navigation'

channel, the lowest-priority movement channel so that the safety controller can override the commands if necessary to prevent a crash.

2.2 Safety Controller (Irene)

The second technical requirement of the lab is the implementation of a safety controller for the racecar. This feature is critical to the function of the racecar because it protects both the racecar hardware and the environment the robot is in. In order to function properly, the safety controller must be robust. The safety feature must be able to prevent the racecar from crashing in a variety of scenarios, such as crashing into a wall or running into people walking in front of it. However, the safety controller can not hinder the functionality of the racecar. The controller must balance being able to go at a high speed close to the wall while not being reckless.

2.2.1 Safety Override

We implemented the safety controller for the racecar by using the command mux's different levels of command priority to override drive messages when the safety mode is triggered. Drive messages are published to "input/navigation" while safety messages are published to "input/safety". When deciding which command to follow the racecar prioritizes "input/safety" messages. However, in order to make sure that we can stop the robot in case our autonomy goes wrong, the highest priority commands are the joysticks "input/teleop" messages so that we can stop the racecar manually if needed.

The safety controller subscribes to two inputs, the lidar sensor messages, and the wall follower's drive messages. The safety controller can be in two states, drive or stop. This can be seen in Figure 2.3. We determine the state of the safety controller based on the lidar sensor readings. If the car is in the drive state it checks if an obstacle is detected. If it sees an obstacle it sets the state to stopped. If the racecar is in drive mode it checks if there are no obstacles and sets the drive state to drive when the path is clear. Then when the safety controller gets a drive message, it checks the state of the racecar and publishes an "input/safety" message setting the drive and angle velocity to 0. We chose to implement this state-based control because of the way messages are published. If we try to publish the "input/safety" command to stop the robot when we receive a lidar sensor reading that detects an obstacle, we do not override the drive messages that are being published. This is because the two types of

messages are published at different rates. When we did this the robot would twitch because it was being told to drive then stop over and over. This approach also allows us to be more robust against faulty sensor readings. Since the stopped safety state is only left when we are sure that the path is clear, we can filter out bad readings.

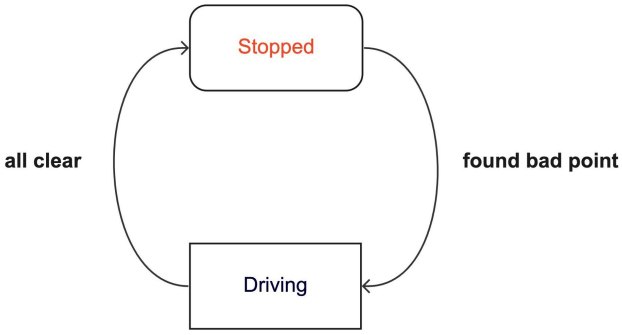

Figure 2.3: State diagram of safety controller.

2.2.2 Detecting Obstacles

We check for obstacles by taking a 40-degree scan of the lidar data directly in front of the robot. We chose to only check for obstacles in front of the robot so that it only reacts to obstacles that it might drive into. After slicing the appropriate region of the sensor data array, we filtered out all of the bad sensor readings. This was done in the same way as for the wall follower. The biggest issue we ran into when first implementing our safety controller was that as the racecar became more shaky at high speeds and close distances it would sometimes detect the wall as an obstacle. While we considered tightening our obstacle measurement range, or shifting it based on what side of the wall we were following. We decided that these options would reduce the racecars' ability to drive safely. Thus our solution was to filter out lidar readings that belong to the wall. The wall-following program already predicts the location of the wall so we created a custom line message to publish our wall prediction from the wall follower. In the safety controller we save our wall predictions and filter out all readings within our safety range that are within a certain distance to the predicted wall (see eq. 2.3).

```
dist_to_wall = abs(self.curr_m * x[i] + (-1) * y[i] + self.curr_b) /
                    math.sqrt(self.curr_m**2 + 1)
```

Equation 2.3: The distance from points x,y to the wall (which is represented as a linear function with slope ‘self.curr_m’ and y-intercept ‘self.curr_b’)

This allows us to follow the wall much more closely and at higher speeds without it triggering the safety controller see figure 2.3 for visualization.

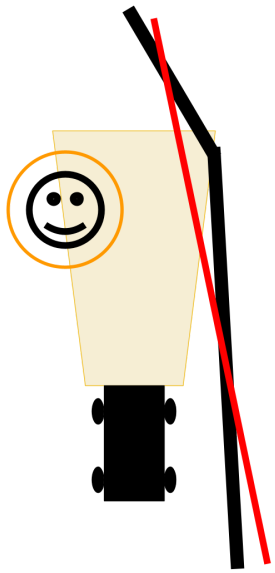

Figure 2.4: Visualization of lidar scan and wall filtering.

3 Experimental Evaluation (Lili)

3.1 Test Objectives

After implementing our wall follower and the safety controller, we needed to ensure that they were functional, robust, and reliable. Testing was also instrumental in giving us feedback and ensuring we knew what to iterate and improve on technically. Due to the criticality of the safety controller, it was especially essential to test multiple different scenarios to ensure that the robot would stop consistently and without issues. Additionally, since prior to the lab we had only tested the wall follower in simulation, we needed to make sure it worked for many different configurations in a real, physical environment. This included testing different wall shapes, turns, and locations. This also meant testing our PD controller at different speeds. We

also needed to verify that our modified processing of the LIDAR data was robust to noise, flickering, and range limitations. Other than testing different scenarios for robustness, we needed to test repeatability in all of our scenarios, in order to prove consistency and reliability. There were many more degrees of freedom and random processes once we transitioned to hardware, thus requiring extensive testing.

3.2 Test Procedures

We built up our tests incrementally, first running the robot in simple scenarios, and then to progressively more complicated ones. This way, we could implement fixes one at a time, rather than changing many things at once, allowing us to better analyze our system's issues and resolve them. After we tested basic functionality, including teleoperated control and the "dead man switch," we moved on to testing our specific implementations.

3.2.1 Wall Follower Tests

Our main tests for the wall follower were as follows: short drive, long drive, sharp turn, and shallow turn. These tests were repeated for both the left and right side, at different speeds, and different following distances. We started our tests at 0.5 m/s and eventually went up to 1 m/s.

To evaluate our system's performance, we made sure to publish tangible metrics so that we could process and graph them after each test. In each iteration we were able to determine if our LIDAR processing needed work, if we needed to further tune the PD gains, or something else.

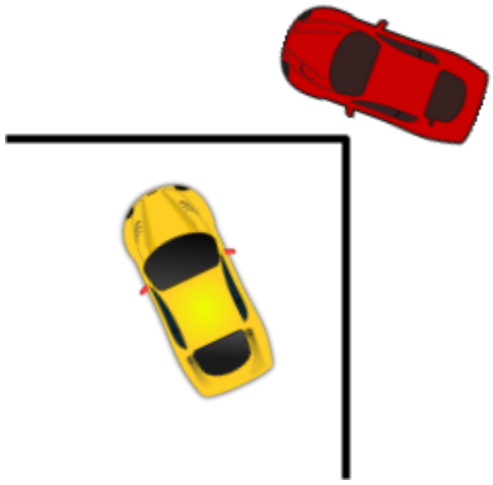

Figure 3.1: We tested turning on both the inside (yellow car) and outside (red car).

3.2.2 Safety Controller Tests

We started off our safety controller testing with the robot on its safety block. This way, we could test it without risking damage to both humans and the robot. We held up many different objects in front of it to try to account for as many different possibilities that could happen. This ranged from large to small objects including large wooden barriers, a water bottle, and a waving hand. Testing this range of objects initially on the block was important to know if our safety controller would indeed stop for seemingly insignificant blips in the LIDAR readings. From these tests we also noticed the robot's stuttering behavior and were able to fix that and a few more details before putting it on the ground.

Figure 3.2: A wooden panel being held in front of the robot on a block.

Due to the nature of its function, we tested the safety controller with less defined and more random scenarios. This included having people suddenly walk or jump in front of the robot, stand in its way along the path, and placing objects in front of it. It was important that we tested human movement (such as dancing) in front of the robot to see whether or not the stopping would persist despite an object going in and out of frame constantly. We also made sure to observe that the robot would not stop if it was too close to the wall but in the middle of a maneuver to move away from it. To test this, we made sure to run many turn tests with the wall follower and see if the robot hesitated at all when approaching a corner aggressively.

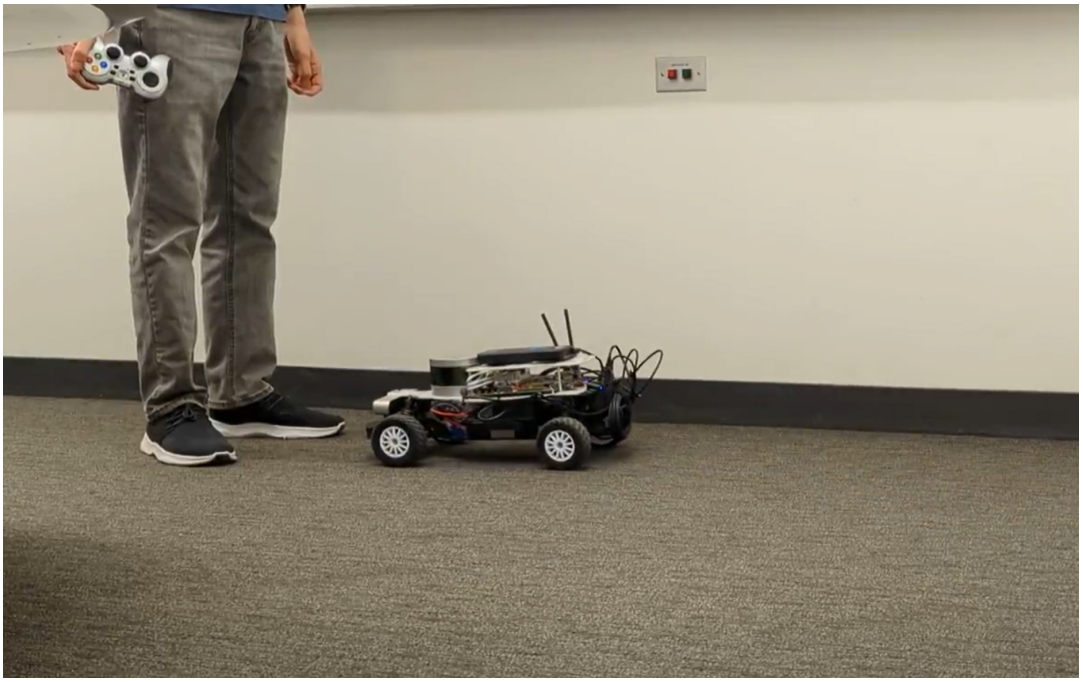

Figure 3.3: The robot stopped in front of a human in its way.

3.3 Test Results

3.3.1 Wall Follower

We mainly used distance error to analyze the effectiveness of our wall follower. Our average error in one of our recorded test series (figures below) was around 0.2 meters. We concluded that this value shows that our wall follower is effective. We can also see from the data (fig 3.4) that the paths with sharp turns tended to show higher and more varying error. The data also show the oscillation in our error term, suggesting we try increasing the D gain in the PD controller. We can also see that for the short straight passes there is some steady state error, so it may also be good to add an integral term to our controller.

It is interesting to observe that the average error in both the short right slow and fast runs are close (fig 3.5). We can also see that the average error for the sharp turns on either side are close and almost twice that of the straight tests.

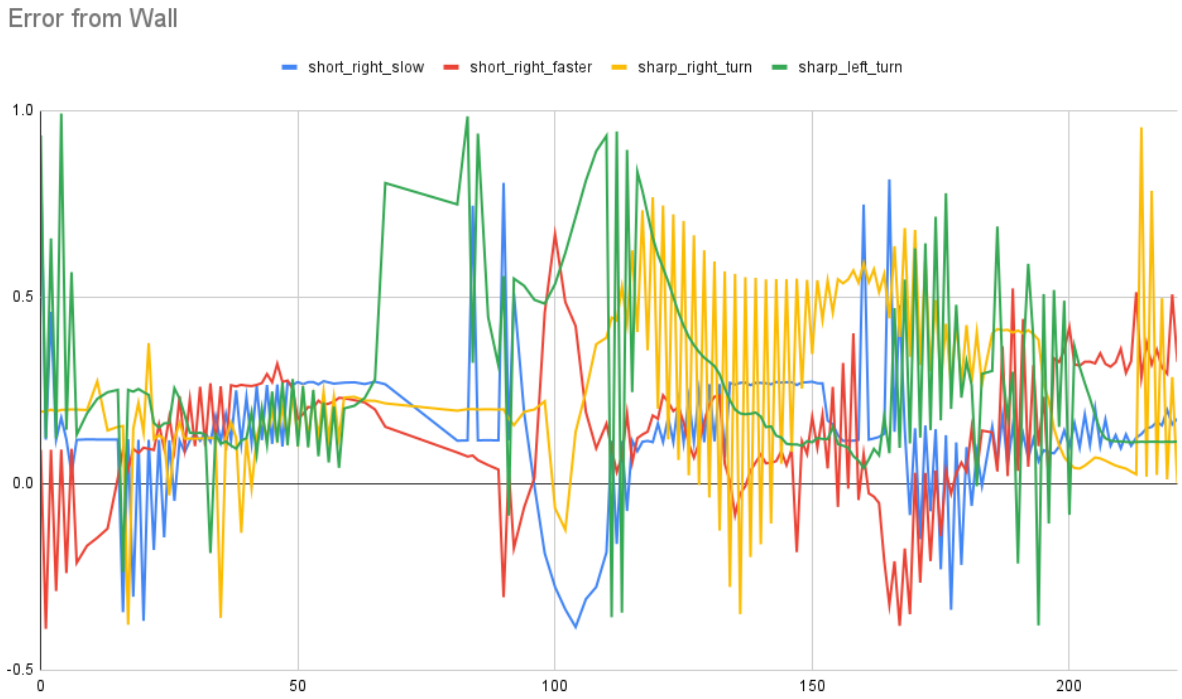

Figure 3.4: Error data from one of our test sequences.

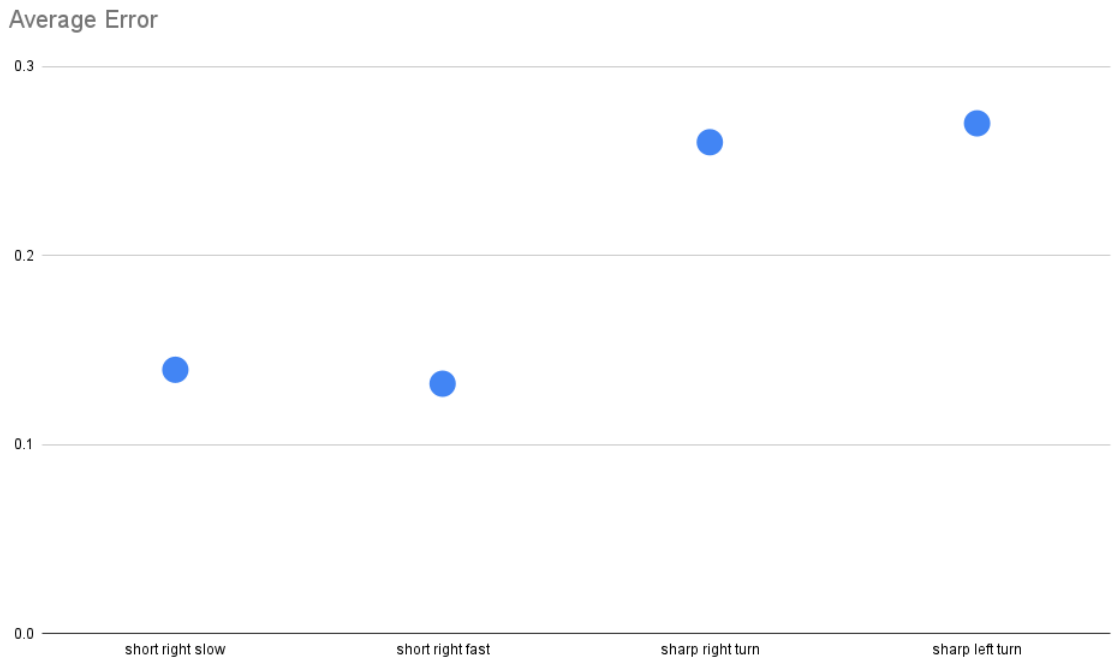

Figure 3.5: Average errors from one of our test sequences.

3.3.2 Safety Controller

Our testing for the safety controller was more qualitative, due in part to the nature of it, as well as the experiments we designed for it.

Scenario	Stopped?
Human Standing	✔
Human Dancing	✔
Human Walking In	✔
Wall (head on)	✔
Object	✔, sometimes went around it

Figure 3.6: Table showing different scenarios tested and corresponding behaviors.

In the initial tests of our safety controller we found that it stopped too often, and figured out that it was stopping for the walls even if the driver was trying to move away from it. Qualitative observations like this and inspecting the RViz real-time allowed us to make improvements. In our final testing, our safety controller proved to be robust and repeatable for different scenarios, including different human motion as well as objects.

3.3.3 Test Limitations and Observations

It is important we recognize the limitations of our testing. For example, despite our processing improvements, there is still a non-trivial amount of noise in the LIDAR data. Due to this, some of our extrapolated line estimates, and consequently, error estimates were incorrect. We saw spikes in our error data as high as 3 meters when we observed with our eyes that that behavior did not happen. This makes some of our measured data inaccurate. Moving forward we will make sure to take this into account when gathering metrics.

Overall, our data and qualitative observations show that both our safety controller and wall follower are effective and reliable.

4 Conclusion (Matthew)

From our experimental results, we conclude that we have created a robust wall-following robot that can maintain a minimal error in a wide variety of tests that cover many of the scenarios that would occur in a real race, and can stop for a range of obstacles that could block or damage the robot during testing. This means that as higher level code sets a goal for our robot, we will be able to follow it fairly well. This also means that we should be able to apply our LIDAR data filtering code to other nodes that need distance data and overcome the error enough to get reasonable performance. Depending on the exact application, it is possible we would need additional filtering to do this, but our code so far will be a good start. Our safety controller is another important fundamental building block for future labs; it will protect our expensive robot as we continue to test new code that could behave unpredictably.

We also have a modular control framework, which has already proved very useful to smoothly implement expected behavior in the robot. In future labs, we can substitute our wall follower with a more sophisticated autonomous controller that fits the same output specifications that our other nodes expect. We can even split the controller into smaller modules, as the skeleton framework is designed to be extensible. For example, we could have one node to handle image processing, and then another to take the results of the image processing and compute a drive output. As we add more and more untested autonomous behavior, we can continue to keep the new nodes lower in priority than the safety controller, which means that our safety controller can still work no matter what command the lower priority nodes send out (within reason).

We could improve our robot in the future by further tuning our PD controller, or possibly adding an integral term. We found that our robot oscillated somewhat as it moved, although we did not develop a system to measure these oscillations quantitatively. Further tuning is often necessary when PID controllers develop oscillations. We tested our robot in a wide variety of trials, and found the average error for each trial, but we only included one trial per test scenario. In the future, we could develop a more sophisticated system of analysis that finds oscillation magnitudes, and includes more than one trial per scenario, so that we could find an average error.

4.1 Lessons Learned

Miles:

In this lab, I learned the difficulties of transferring a piece of code that works well in a simulation to the real world. I learned how to fix problems as they arose, and learned various techniques to increase the accuracy of our wall-following and safety controller code. For example, I learned how to use a state system to make sure our robot remained stopped, even when it received garbage data from the LIDAR. On the CI side, I practiced my presentation skills, and through the feedback received, learned how to give better presentations in the future.

Irene:

I learned two main lessons from this lab. How to work through the bugs that arise from implementing code on a racecar that needs to perform in the real world and how to properly present the result of our engineering research. When transferring the code onto the racecar, the problems that were hardest to solve were the ones that we did not expect. For example, understanding the way that lidar vs drive messages were published out of sync and that the lidar data was noisy and inconsistent and finding strategies to deal with these issues were the most challenging. In addition, I learned that you can prove your engineering achievements through proper experimentation and that you need to present your results in a way that not only displays your technical approach but also shows how well it works. I also practiced my teamwork and presentation skills.

Matthew:

I learned a couple of lessons during this lab. One major lesson was seeing how different the LIDAR was from the simulation. I expected that our code would not follow the wall as closely as it did in simulation due to error in the LIDAR, but I did not expect us to need to change our code too much in order to even get the car to function. It looks like we took some aspects of our simulation for granted, particularly the idea that good data would arrive at every timeslot. I think it would have been difficult to predict this issue in our code in simulation, so I think the best strategy is to expect for and plan time to deal with major differences between the world and simulation. I also learned that we should continue to spend time thinking of unique tests that cover all cases. Our robot initially stopped when put in front of walls, but later would not

stop for people. If we continue to think of unique test cases, we will be able to catch many errors easily.

Lili:

I enjoyed doing this lab a lot! I learned a lot about how to effectively develop code, technical improvements, and more as a team. It was a bit tricky figuring out how to best work on technical things together at first. Only one person could be running the robot at one time basically, and we had to figure out things like, did we want to modify code directly on the robot vs someone else do it offline and transfer it afterwards. Also, figuring out how to make sure we got everyone's input on the technical work while one person was coding a lot is something I think we ended up working through fairly well. I also learned a lot about being organized and systematic in testing. I think our work in this lab was a good starting point, but I have a lot of ideas on how to improve on this for future ones.