Lab # 3 Report: Wall-Following and Safety Controller

Team 3

Wing Lam (Nicole) Chan
Shanti Mickens
Magnus-Tryggvi Kosoko-Thoroddsen
Sheng Huang
Jessica Zhang

RSS

March 10, 2023

1 Introduction - Shanti

In Lab 3, we had two objectives: implementing wall-following and building an adaptable safety controller. In Lab 2, we all successfully created a wall-follower algorithm that achieved between 96-98% accuracy in simulation, but when it came to bringing that into the real world, we realized that simulation does not equal the real world. We were going to need to tune our PID parameters and combine ideas from multiple team members' implementations to create a more robust controller, so we synthesized our implementations for wall-following in the simulator to create a stable and consistent wall-follower that works in the real world on our racecar in a variety of situations, including different starting angles relative to the wall, velocities, distances. Wall-following was the first step for our racecar to be able to autonomously navigate the world around it, and it will be important for future labs and challenges where we race the car around areas where being able to follow and avoid walls will be essential.

The second goal of Lab 3 was to implement a safety controller layer on our racecar controller to maintain a safe environment and protect our racecar from crashing into obstacles. For the safety controller, we were given the freedom to decide what situations it should step in and what that reaction should look like. With that freedom, we had to balance the trade-off between being cautious and allowing our robot to be fast and agile, which will be important when racing our car in constrained environments that require higher velocities, tight turns, and quick maneuvers. We were not given a set of test cases to tell us if our controllers are robust and working well. Rather, we had to consider how we would quantify our controller's performance, and if it met our standards for what we believed should be considered passing. For this, we defined a set of performance metrics: $90 - 95\%$ accuracy to the desired distance over time, very little to no oscillation in the wheels as it follows the wall, and being able to handle areas of sparse data points from the LIDAR scan.

2 Technical Approach - Sheng, Jessica, and Magnus

Before creating the driving instructions (*AckermannDriveStamped*) for our racecar to follow, the team had to understand the world from the robot's point of view. The primary vision system in the previous simulation and Lab 3 is Laser Imaging, Detecting, and Ranging (**LIDAR**), a vision system based on sending lasers from several angles at once and using the laser travel times to determine distance to surrounding objects and obstacles. This data collection method enables the implemented scripts to steer the racecar away from danger and towards the racecar's desired paths extremely fast. The LIDAR scan is relayed to the racecar's onboard computer as a *LaserScan* object, represented as coordinate system of polar coordinates, (r, ϑ), across the angle range $[-2\pi/3, 2\pi/3]$ and a range domain of several meters. Figure 1 shows the vision cone of the racecar with distance markers from the racecar's LIDAR data.

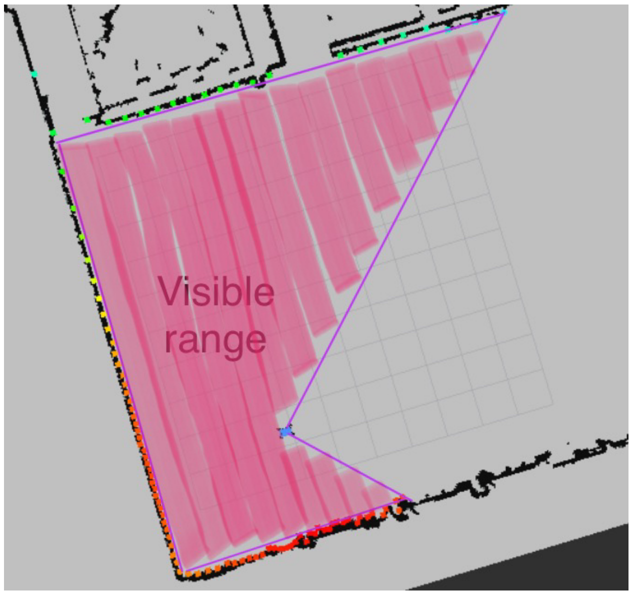

Fig. 1. The vision cone of the racecar's LIDAR scan where distance markers scale from red to blue, shortest to longest. The shaded red region represents what area is visible to the racecar.

There are some key known differences between the simulator and the real world racecar. First, the number of LIDAR beams from the Hokuyo module is approximately 10 times as many as in the simulator, which means the angle increment decreased by a factor of 10, so slicing the LIDAR information according to fixed indices would need to be adjusted. Fortunately, our system found such indices using the angles that correspond to the desired vision cones, so little tweaking was necessary. Second, the field of view is different than in the simulator, so keeping out inconsequential data was more important than before.

Now that the racecar's vision process is understood, the wall-following and safety controller algorithms can be broken down into three discrete steps:

1. Find the wall we want to follow in the *LaserScan* data and represent it accordingly,
2. evaluate the shortest, or normal, distance to said wall representation,
3. and determine driving instructions to send to the racecar based on desired distance, actual distance, and current speed.

First, this section will give an overview of our system, our wall-following controller, and then our safety controller.

2.1 System Diagram

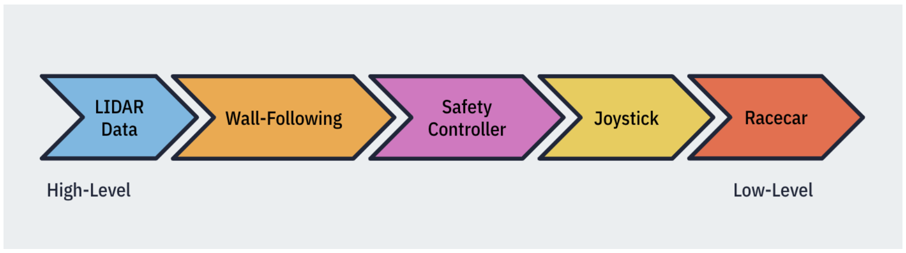

Fig. 2. Racecar's system diagram showing where wall-following and safety controllers are situated in the system.

To drive the racecar, we publish *AckermannDriveStamped* commands to define what speed and steering angle to drive with. The process for deciding what commands to send starts by first taking in the LIDAR scan data and running linear regression to determine where the wall is. From there, we use a PID controller to calculate the necessary steering commands to guide the racecar to the desired distance from the wall. Before passing these steering commands to the racecar, the commands are passed through our safety controller, which also looks at the LIDAR scan data and the steering commands from the wall follower publisher to check if the drive commands can be done safely without running into obstacles. At any point, if the joystick is activated, it will override any commands sent to our racecar, acting as a dead man's switch. If the joystick is not active, then the safety controller sends the potentially modified commands to the racecar to drive.

2.2 Wall-Following Controller

When the wall-following software is first initialized, it reads from the default or test parameters which wall to follow, indicated by a $+1$ for the left side and -1 for the right. This is because the inertial frame of the racecar is established as the x-axis passing through the center line of the racecar, the y-axis through the sides, and the z-axis through the top, as seen in Figure 3.

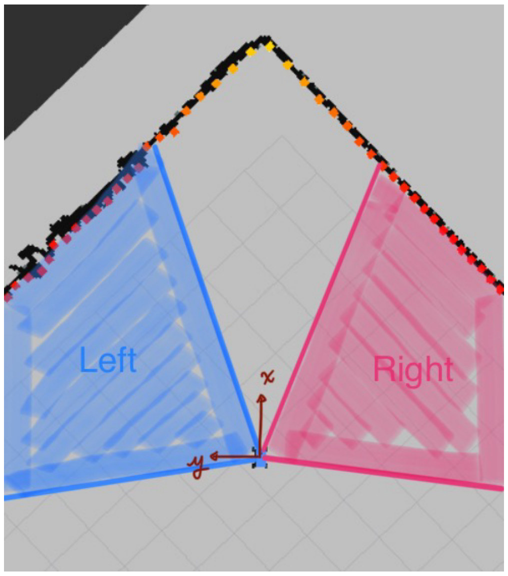

Fig. 3. The vision cone of the racecar's LIDAR data split into "left" (blue) and "right" (red) consideration regions for wall-following, with the inertial frame of the race car attached to itself

Once the wall has been selected, the software slices the *LaserScan* data into "left" and "right" regions, $[-1.97, -.4]$ and $[.4, 1.97]$ radians, respectively. Then inside the desired region, the shortest distance is found, and all the points "forward", or towards the front of the racecar, are portions of the wall that are considered. This is to avoid reading points behind us that are inconsequential because the racecar will not be driving backwards with no vision from its blind spot. Additionally, the racecar should also not regularly consider points that are directly in front of the racecar if they are several meters farther away than the desired distance. They are not immediately relevant and can skew the "image" of the wall the racecar sees, causing unintended turning behaviours, so they are excluded in most cases. However, there are edge cases where the racecar could turn into a wall even if it was properly following the path due to steering and speed limitations, so a state machine is implemented to control for this:

```
if $dist_(front) \leq \alpha * v * dist_{desired}$ then
    turn away from wall to safety
else
    continue regular wall-following
end if
```

The scalar factor a is an arbitrary value and v is the racecar's current velocity, enabling empirical testing until the racecar no longer crashes at corners and preemptively turns away from danger. One thing to note: when finding the walls to follow, parsing parabolic readings in polar coordinate systems is not easy to perform or quickly understand, as walls do not exhibit linear behavior in polar coordinate systems, so the group opted for a polar to Cartesian conversion, with $(x, y) = (rcos(\vartheta), rsin(\vartheta))$, simplifying visualization and error analysis for the rest of Lab 3. After the wall has been found and converted into Cartesian coordinates, a linear regression is performed with the intent of filtering out extraneous bumps in the actual wall the racecar should not consider, i.e. a wall outlet, Ethernet cord, etc. This linear regression (performed with *np.polyfit*) is then effectively considered the "wall", or the representation of the real-life wall the racecar should follow along, as shown in Figure 4.

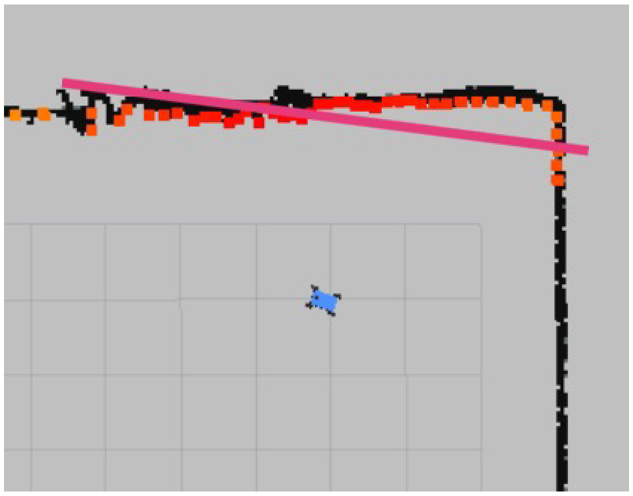

Fig. 4. A possible linear regression of the right wall, shown in pink, based on the racecar's LIDAR scan data, shown as in red and orange.

After the "wall" has been found, the normal distance between the racecar's LIDAR sensor and the "wall" is calculated using the coefficients from linear regression, which is found using Equation 1.

$$m, b = \text{np.polyfit(x, y)} \tag{1}$$

$$\Delta y = \frac{|b|}{\sqrt{1 + m^2}} \tag{2}$$

Then, Δy, from Equation 2, is used for steering the robot towards the desired path; therefore, our error signal $dist_{desired} - \Delta y$ is directly proportional to the steering angle Ψ. Unfortunately, this 1:1 ratio between error and Ψ does not perform well, as it would correct the error at a slower rate, causing undesirable oscillations in the path, so an error constant K_p was applied to the error signal to improve the rate at which the racecar approaches and corrects towards *desired*. Derived from the standard PID control equation $u(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de}{dt}$, our final determined steering angle $\Psi = K_p * \Delta y$. Our value for K_p was initially determined from the previous wall-follower simulation, but was empirically refined using the racecar's set velocity v and a quadratic regression of the K_p values that performed well at various set velocities to generate Equations 3 and 4, which define the final wall-following steering commands.

$$K_p = 0.1229 * v^2 - 0.8706 * v + 1.426 \tag{3}$$

$$\therefore \Psi = (0.1229 * v^2 - 0.8706 * v + 1.426) * \Delta y \tag{4}$$

Finally, the *AckermannDriveStamped* command uses Ψ and v as the final instructions for the racecar when wall-following is run. However, one more processing step is required to ensure the racecar does not break itself or surrounding objects during the main process: a safety controller.

2.3 Safety Controller

When implementing the safety controller, we were faced with the challenge of deciding what situations our safety controller should step in and how it should react. With this, we had to balance the trade off between being cautious and having the freedom for our racecar to be fast and agile to enable it to perform well in future challenges when we race our racecar. To acheive a balance, we decided to define Regions of Safety of the racecar's field of view, as shown in Figure 5 (all angle ranges are in radians). Region 1 is from -0.25 *rad* to 0, Region 3 is from 0 to 0.25 *rad*, Region LEFT is from 0.4 *rad* to 1.3 *rad*, and Region RIGHT is from -1.3 *rad* to -0.4 *rad*. These discrete ranges allow our racecar to swerve or turn to avoid obstacles when possible, and only stop if a collision with the obstacle is unavoidable.

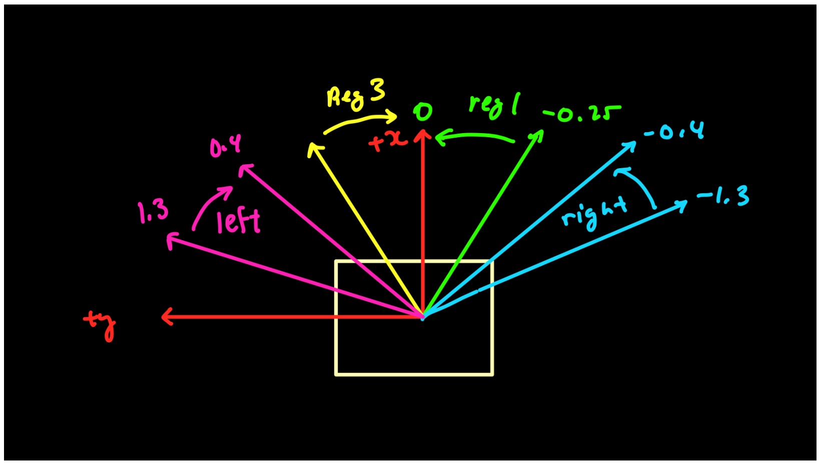

Fig. 5. A bird's eye view of our racecar's "Regions of Safety".

At a high level, the safety controller takes the following steps:

1. Receive *LaserScan* data for all regions,
2. calculate the closest allowable distance between the racecar's front bumper and the wall before an inevitable crash occurs using the following formula:

$$maxDist_{front} = 1 * dist_{desired} + v * \overline{\delta t} * 1.5 \tag{5}$$

which depends on the racecar's velocity v, $dist_{desired}$ and the average LIDAR's scan time step $\overline{\delta t}$,

3. then begin processing the LIDAR's *LaserScan* data using the following conditions.

The safety controller's processing is split into 4 different cases:

Case 1: if both Region 1 and Region 3 are blocked, then check to see if there is any space in LEFT or RIGHT for the racecar to turn into those regions.

Case 2: if Region 1 good and Region 3 is blocked, then turn the racecar in the direction of Region 1.

Case 3: if Region 1 is blocked and Region 3 good, then turn the racecar in the direction of Region 3.

Case 4: if all regions are blocked and the racecar cannot turn anywhere to get out of the current situation, then stop the racecar.

After this algorithm was implemented, the only thing we had to change was the topics the safety controller script published to, then the racecar was ready for manual tuning, and eventually autonomous operation.

3 Experimental Evaluation - Nicole, Magnus, and Shanti

3.1 Testing Procedure

In order to test our wall-follower code, we decided to repeat similar tests from the Lab 2 wall_follower_sim assignment in various locations, including Maseeh, Stata Basement, and Classroom 32-082. We defined six test cases: inner corner, outer corner, starting angled towards the wall, starting angled away from the wall, starting close to the wall, and starting far away from the wall. For each of these, we then tested it at varying velocities, desired distances from the wall, and racecar's orientation to wall (left or right side of the racecar). We chose this set of test cases to ensure our robot can meet our standards for performance at various speeds, starting positions, and distances relative to the wall.

For our set of test cases, we defined three factors for our performance metrics. First, it needs to have $90-95\%$ accuracy to the desired distance over time. Second, there should be very little to no oscillation in the wheels as it follows the wall. Lastly, it should be able to handle areas of sparse data points from the LIDAR scan. The following plot graphs the error of the wall-follower algorithm over time at different velocity intervals. Left and right wall-following is denoted by the legends.

To collect numerical data on the wall-following performance, we included code to record the error between the pre-set desired distance from the wall and the actual distance that the racecar is from the wall to an external file. With those numbers, we can evaluate the accuracy of the actual distance over the time of the algorithm by examining how often the racecar is driving at exactly the desired distance from the wall. We decided to test this metric because it can directly measure how well our controller can stay at the desired distance from the wall. The data presented later in this paper is from running our code in Maseeh down the halls and around various corners.

It was slightly more complex to test the performance and reliability of the safety controller since any manual input from the joystick will override the topics that the safety controller is publishing. To combat this, we had to use a different testing procedure which involved testing out different scenarios, such as getting too close to the wall with a poor wall following algorithm and randomly placing an object in front of it to emulate someone walking in front of our racecar.

3.2 Results

In the first tests, we ran the wall-follower code for the racecar to follow the right and left wall at different speeds (0.5m/s, 1.0m/s, and 1.5m/s). Figures 5 and 6 are the plots showing the racecar's percentage error (which we defined as the difference between the racecar's distance away from the wall being followed and the pre-set desired distance) as the racecar moves along the right and left wall, respectively. In both figures, we see that the blue line (the one signifying a velocity of 0.5m/s) has the least oscillation while the yellow line (the one signifying a velocity of 1.5m/s) has the most oscillation. At slower speeds such as 0.5m/s, the racecar was able to follow the wall at the desired distance with ±5%. At 1.0m/s, the error is within ±10%. At 1.5m/s, this error increases to $\pm 5 - 15\%$.

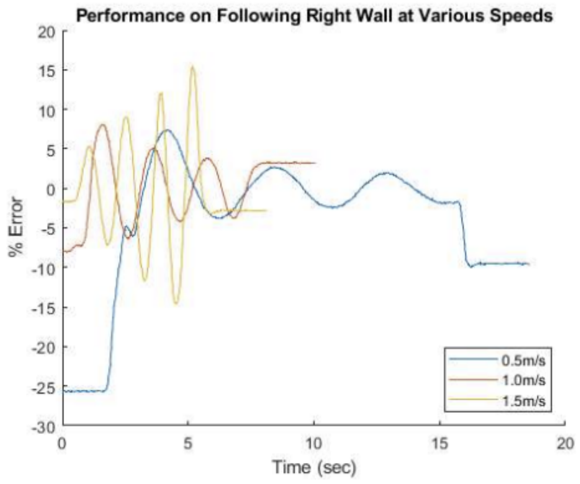

Fig. 6. The car is able to follow the right wall with minor amounts of error between 5-15%, increasing as speed increases from 0.5m/s to 1.5m/s.

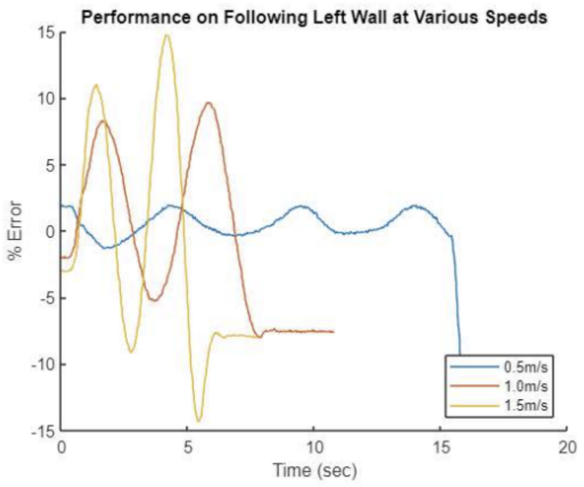

Fig. 7. The car has a similar error rate of between 5-15% when following the left wall at different speeds.

For the next tests, we checked the car's ability to follow the wall as it turns a corner. We separated the tests into smaller ones separated by the different turns possible: 90 degree turns and "outer corner turns" (these are turns that would travel along the outside of the aforementioned 90 degree turns).

Firstly, Figure 8 shows the performance of the car at following the 90 degree turn along the right wall. At 0.5m/s, the car began with some error initially but settled to approximately 0% error by the end of 4 seconds, which was when the turn occurred. The large peak between 5 and 7 seconds is the error caused by the actual turn. After which, the error settles from approximately 5% back to 0% again. A similar shape is seen when the car moves at 1.0m/s: settling from initial error until the turn that occurs at 2 seconds, a peak between 3 and 4 seconds being the error caused by the turn, then settling back to 0% error towards the end of the test window. From both these behaviors, we know that the car can sense the wall in front of it and turn appropriately to minimize the error from the desired distance after the turn. The difference in the heights of those error peaks can be attributed to again the faster velocity inducing more error, since it allows less time for the car to adjust for the movement away from a straight path and our scaling of PID gains with velocity terms.

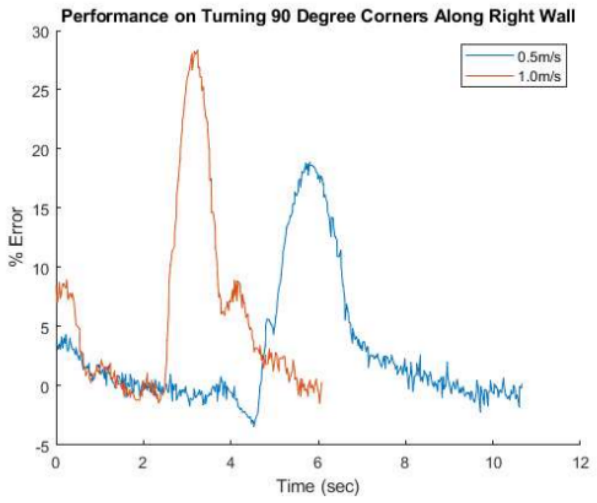

Fig. 8. At both speeds, the car can perform the left turn and return to 0% error fairly quickly.

Secondly, we performed outside corner turns with the car facing the left and right wall, shown in Figures 9 and 10, respectively. In both graphs, the car followed the wall at 0.5m/s, 1.0m/s, and 1.5m/s. At 0.5m/s in both orientations, the car adjusted for the error in the initial placement of the car until the turn occurred at 5 seconds after the tests started. The turn resulted in a spike in percentage error, but the car settles to within 10% error in 1 second after the turn completed. At 1.0m/s and 1.5m/s, the car behaved similarly with the only difference being the time at which the spike in percentage error happened, which was a direct result of the speed the car was traveling at.

One notable feature is the fact that there continues to be more oscillation in the percentage error as the car's velocity increased, which is a continuing result from our other wall-follower tests. Also, the peaks seem to have the same height in both graphs and at all speeds, meaning that the velocity and orientation of the car with respect to the wall did not have an impact on the error it accumulated as it traversed the corner.

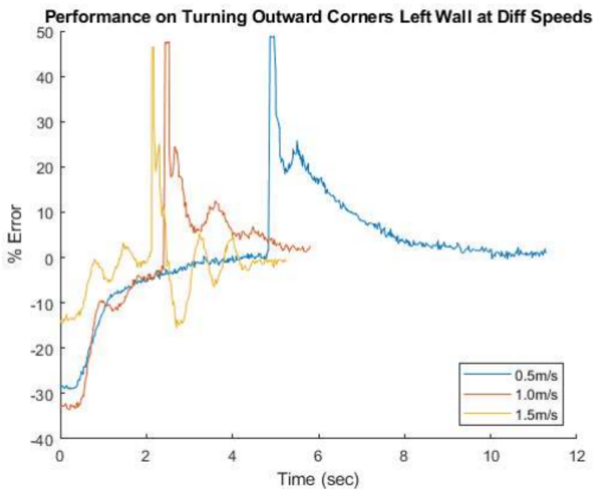

Fig. 9. The car always settled back to 0% error after the turns occurred, regardless of its velocity.

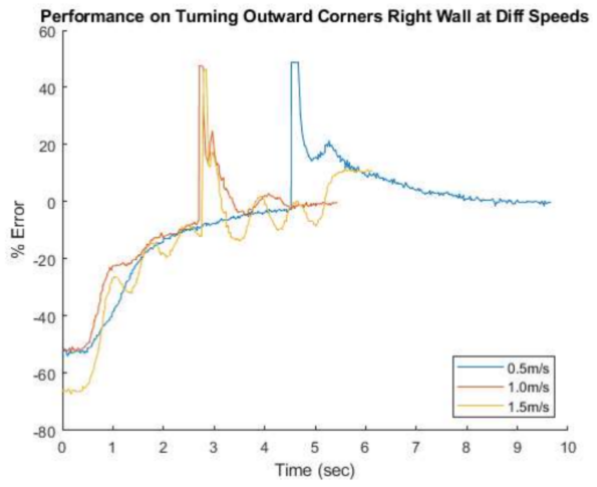

Fig. 10. Although there is some oscillations as the car travels at 1.5m/s, the car approached a steady state of approximately 5% error at the end of the testing period.

Next, we changed the starting distance from the wall. In Figures 11 and 12, the car was placed at 0.8 meters and 0.3 meters away from the wall respectively, in order to simulate both a closer and farther starting point. In Figure 11, we see that at all three velocities (0.5m/s, 1.0m/s, and 1.5m/s), the car begins with negative error (signifying that the car is too far from the wall) then moves closer to decrease the percentage error until there is only ±10% error after the test ran for 4 seconds. In Figure 12, the car displays the same oscillating motion as when it was following the left and right walls without any modifications to its starting position in Figures 6 and 7. It began with a positive error (signifying that the car is too close to the wall) then settles at ±5% error after a few seconds. We see that the car is able to stabilize quicker at higher velocities, though this results in a higher amplitude in percentage error.

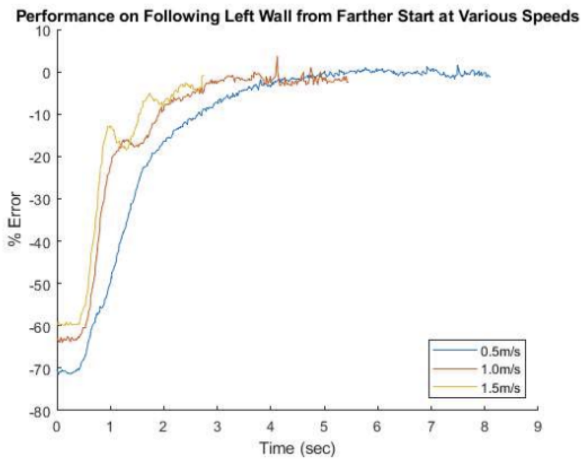

Fig. 11. Despite starting at a farther distance from the wall, the car readjusts back to the desired distance and maintains it for the duration of the tests regardless of its velocity.

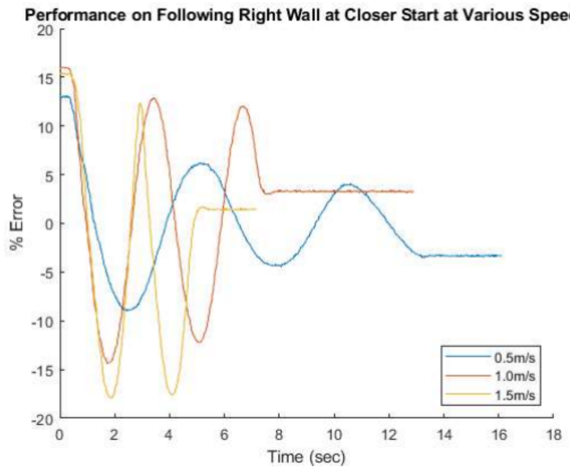

Fig. 12. The car oscillates for two periods before settling to minimal error, with higher speeds settling quicker but having a higher error.

After varying starting distance, the next test standardized the starting position and faced the front of the car away from the wall, reminiscent of the fourth test from Lab 2, where the car followed the left wall while starting at an outwards angle. Figures 13 and 14 depict the errors slowly approaching 0 as the car corrects its angles, settling to near 0% error around 4 seconds for both walls, implying the behaviour is independent of desired side, which is favourable. However, the errors near the beginning of the right wall following for 0.5m/s are very oscillatory before returning to predicted proportional behaviour. This could be due to calculation errors or noise, as visually, the car did not display that level of oscillation. Fortunately, higher speeds perform more consistently.

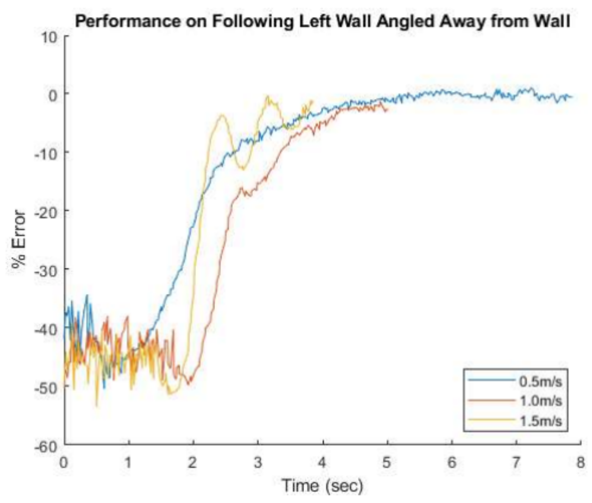

Fig. 13. The car began with some noise then settled to 0% error a few seconds into the tests, regardless of velocity.

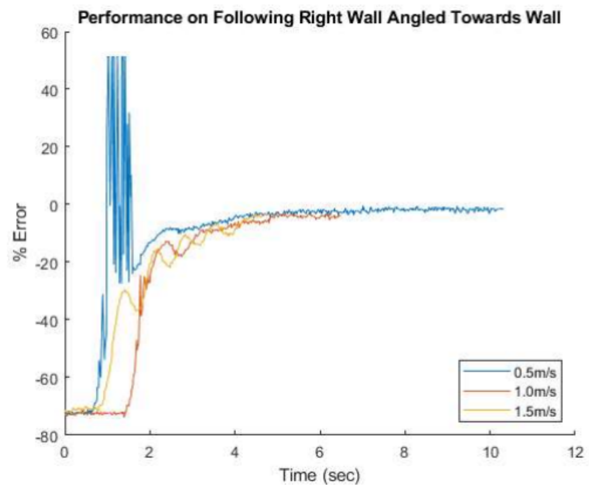

Fig. 14. Despite some noise in measurements in the 0.5m/s test case in the beginning, all three velocity test cases settled to having minor error of around 5%.

Finally, we tested the safety controller's effect on the error signal. Since the safety controller's only objective is to avoid obstacles and crashing, we simulated "dangerous" driving by adjusting our proportional gain such that the car oscillated with significant error and often "skim" the wall. In Figure 15, no safety controller was running in the background. As the car attempted to follow the wall, it got too close to the wall and required human intervention to pause the wall-follower code from continuing until our team could ensure that the car can perform the next oscillatory turn without crashing. These pauses are denoted by the plateaus in the graph at 9 and 13 seconds. In Figure 16, we ran the safety controller alongside the wall-follower code, but now no operator intervention was needed to ensure the safe testing procedures. The car slowed down as the safety controller took over whenever the car approached the wall. The data indicates that after avoiding the obstacle, the car moves back to following the wall. Although the timescale of this re-correction is not instantaneous, this is not a variable we can necessarily control as obstacles can appear in many shapes and sizes, therefore a return to the original path is considered successful behaviour. A video of this behaviour is available in the briefing slides for further proof.

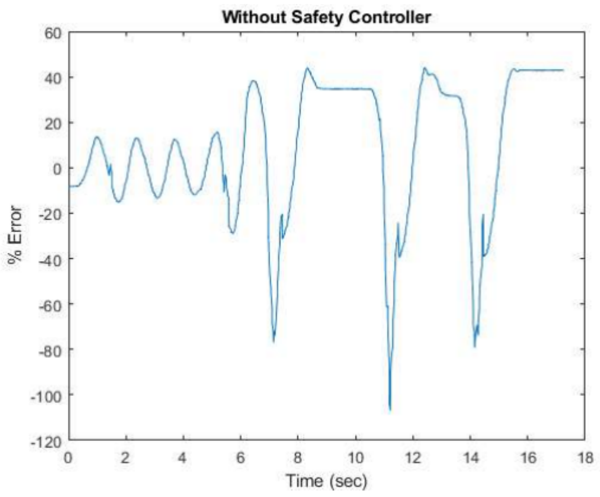

Fig. 15. Without the safety controller, human operators had to intervene by letting go of the dead man's switch during the tests whenever the car is on the brink of crashing.

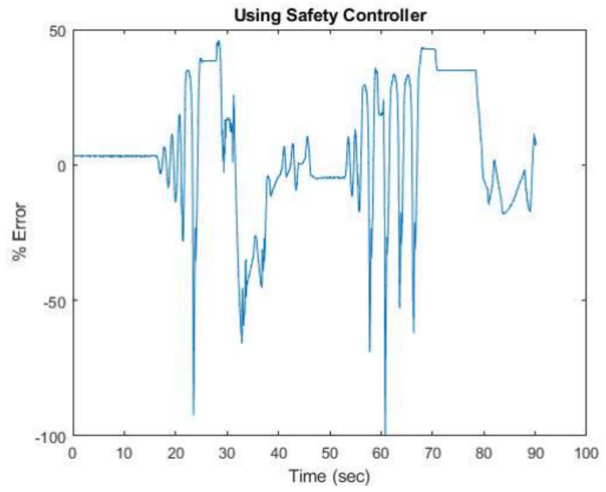

Fig. 16. With the safety controller implemented, the car slows down whenever it approaches an obstacle, turns away from said obstacle and attempts to continue the wall-following algorithm if the path to a wall is clear.

4 Conclusion - Shanti

Over the course of Lab 3, we have successfully implemented an algorithm to allow our racecar to follow walls and a safety controller to protect our racecar from crashing into obstacles We tested the wall follower in a wide-variety of scenarios, including different starting angles relative to the wall, velocities, distances to the wall, and orientations, to ensure it is robust. We also designed our safety controller to follow the principle that "the racecar is never too scared to move forward" as outlined in the Lab 3 instructions. When faced with obstacles in its vicinity, the racecar will slow down and turn away slowly and attempt to continue the wall-following algorithm.

We would still like to do more work to clean up our current codebase to make it easier to continue to build off of as we move into future labs. In addition, there still tends to be oscillation in the wheels when the racecar is ran at high speeds, such as 3.5-4.0 meters per second. To resolve this issue, we will need to do further tuning of the proportional and derivative gains.

In the next design phase, we will be using the camera and implementing object detection algorithms to allow the racecar to detect an orange cone in its field of view. We will also implement a parking controller that can navigate the racecar to go park by the cone. Lastly, we will extend our parking controller into a line following controller.

5 Lessons Learned

Below we each included our own self-reflection on the technical, communication, and collaboration lessons we learned in the course of this lab.

5.1 Nicole

This was the first time that I was put into a project-based class and communicate with a team about collaborating throughout the coding, consolidating our simulation results, debugging hardware problems, testing the performance and communicating our results in a written or spoken format. It was not difficult to consolidate our code early on, as we took turns evaluating the pros and cons of each member's code before deciding to use one code structure as the main wall-follower. I found that it was easier to convey mathematical concepts and discussing coding approaches using drawings before tackling the actual coding.

5.2 Shanti

In this lab, I learned that communicating with the help of visuals and drawing diagrams was very helpful when describing how my wall follower worked to my teammates. I also found it helpful to create a document to track troubleshooting notes, so that when we ran into the same problem as someone else on our team has, we have a document recording problems we have faced and what solutions have helped us solve them. Lastly, going from working with a racecar in simulation versus working with it in real life has made me see how hardware in real life doesn't align with how it behaves. I learned that since hardware can be finicky, it's important to create robust code that can handle noisy data.

5.3 Magnus

Using diagrams to convey and understand the ideas my teammates and I created for approaching the wall-following problem was the best method for synthesizing all the information we had learned. It also supported the less abstract discussions regarding how we handled the ranges and angle arrays to find walls and debugging when the racecar had issues with flipping axes and whatnot.

I also like how our team split the workload between report and coding, as too many cooks in the kitchen can often create more problems than it solves. But when it came time for us to hand off work to another teammate when other responsibilities called, we were all able to effectively relay any changes and concerns we had with the code, like the important ranges of the LIDAR, or the control constants for the PID controller. I could feel confident when I looked at our team's code, and know that they also understood how mine worked. The only downside this lab was that the hardware didn't work with all of our computers, so having multiple copies on everyone's computers gave us multiple options for testing.

5.4 Sheng

There was definitely a lot of experience learned in debugging the hardware compared to software.
There were many times when the racecar broke down in the middle of testing and we had to think of ways to debug either using rostopics to see if anything was being sent to the robot or if the hardware itself broke. Also, I gained experiences with working with the laser ranges from the LIDAR. When our wall follower was reacting too hard due to a "dip" in the wall, we changed the laser angle range to make the robot look further

ahead. When there were too much noise on the wall, we had to change the LIDAR distance again to make sure that we are focusing on just the wall that we are following and not everything that is surrounding the wall.

5.5 Jessica

This was the first project I've worked on that has such a heavy emphasis on both hardware and software. I learned a lot about how sometimes results in simulation don't translate well in the real world, and how to debug hardware problems, since most of our most time consuming errors were actually hardware related instead of software. I liked how our team divided up the work and communicated with each other. It felt like everyone was working on something at all times, every part of the project had someone looking at it, and everyone was vocal about their progress on their part.