Lab 3 Report: Wall Follower

Team 13

Ra Mour (Ronald Alvarez)
Srinath Mahankali
Nandini Thakur
Jorge Tomaylla
Sarah Zhang

RSS

March 10, 2023

1 Introduction

Author: Ra Mour

What is the most important part of a car? The brakes.

The most basic autonomous navigation system should allow a robot to independently explore its environment without crashing or stopping unnecessarily. Once simple, safe navigation is guaranteed, richer instructions can be built on top to perform higher-level tasks. A basic approach to novel-environment exploration is to move along its boundaries while maintaining enough distance to avoid collision. In fact, this *wall-following* algorithm is the simplest way to escape a large collection of mazes.
In this lab, we transfer a wall following algorithm, originally developed for a simulated scenario, to a physical racecar-robot. The racecar is instructed to move along at a specified distance from the nearest wall. We use LIDAR data to measure the racecar's true distance from target wall. We define an error signal by comparing the true and desired distances, then generate steering instructions that correct the racecar's trajectory by combining the error signal with a PD controller. Finally, we perform a series of tests to ensure the robot consistently performs as expected in a variety of scenarios.

2 Technical Approach

2.1 Useful Concepts and Terminology

Author: Jorge Tomaylla

The following terms are recurrent concepts that are integral to understand the problems faced during this lab and the solution process.

PD Controller: The Proportional Derivative Integral controller is one of the most accurate and stable controllers for driving a system towards a target position using a closed feedback loop. This is achieved by keeping the output of the system as close as possible to the target position.
The output is calculated by the formula:
$\mathbf{K_p}$: The proportional gain controls how quickly to turn the steering angle when the heading is not at the target distance.
$\mathbf{K_i}$: The integral gain adds to the steering angle if an error persists for a long time.
$\mathbf{K_d}$: The derivative gain reduces the speed of the steering angle response to error change.
LIDAR scan: Distance data from the race car's LIDAR sensor to all objects around it on a range of $-3\pi/4$ to $3\pi/4$.
Linear Regression: A mathematical model to linearize noisy data.
IQR Filtering: A technique to eliminate outlier points in a data set.
Safety threshold: The distance at which the safety controller must be triggered to stop the race car from crashing or to veer away from an obstacle.
ROS Topic: An object of the Robotic Operating System environment that allows the user to subscribe (listen) to data in the topic or to publish data to the topic.

2.2 Wall follower

Author: Jorge Tomaylla

The first objective of this lab is to successfully implement a wall following program on the physical race car that is functional in most environments and corner cases. I will explain our wall following approach and how our algorithm tackles the most pressing problems.

How to give the race car a way to visualize a wall?
We check what resources and sensors we have available to understand our surroundings. The race car utilizes a LIDAR sensor that that publishes Laser Scan data to a /scan ROS topic in the form of an array of distances radially outward from $-3\pi/4$ to $3\pi/4$. Depending on the target wall to follow (left or right) we break down the ranges array on a left, right, and front array (specify the ranges chosen). We then create an array of angles that correspond to each distance in

the ranges array and break it into left, right, or front angles accordingly. We then run a linear regression on the ranges and angles arrays to obtain a line of best fit for the laser scan data points. This line represents how the race car interprets its environment.

How to modify the car's steering angle so that we are a target distance away from the wall?
We employ a popular and stable closed-loop feedback control system, a PD controller. We define our error to be the closest distance to the wall minus the desired target distance from the wall. This is negated when changing from left to right target wall because both cases are symmetrical but must steer the opposite direction away from the wall. We describe the process to choose the correct K_p, K_i, and K_d values in the experimentation section. Finally, the output of the PD control system is the new steering angle to send to the race car.

How do we prevent outlier points from influencing the regression line the car uses to observe the wall?
We run a filtering algorithm to eliminate outliers from our ranges and angles array before making the linear regression. We chose to take the mean and standard deviation of the data. After subtracting the mean from all data points, normalizing the data, we eliminate all points that are greater than the standard deviation. This is equivalent of deleting the influence of the bottom and top 15% of the data distribution.

2.3 ROS Implementation and Breakdown

Author: Sarah Zhang

We use ROS with Python 2.7 to implement our wall-following algorithm. Our wall follower gets data via a ROS subscriber listening to the racecar's Hokuyo LIDAR scanner data in the form of LaserScan messages, and controls the racecar's motion via a ROS publisher that publishes AckermannDriveStamped commands. We will first present the pseudocode for wall following logic, which is driven primarily by a PD controller, followed by a breakdown of the code and the design choices behind them.

Pseudocode for callback function, which manages all the logic and calculations for determining the AckermannDriveStamped command:

```
    procedure CALLBACK(scan data)
2:      // Take three slices of the LIDAR scan data as follows:
        front slice = the middle 5 scans
4:      if following left side then
            side slice = scans from π/5 to π/2
6:          back slice = scans from 3π/5 to (max scan angle - π/5)
        else if following right side then
8:          side slice = scans from −π/5 to −π/2
```

back slice = scans from $-3\pi/5$ to (min scan angle + $\pi/5$)

end if

Filter out scans beyond the distance of 4 m, 2.5 m, and 3.5 m for the side, front, and back slices, respectively.

if if len(side slices) $<$ 5 and len(back slice) == 0 **then**

// Car is not near walls

steering angle = 0

else if if len(side slice) $<$ 5 and len(back slice) $>$ 0 **then**

// Car is going around corner

steering angle = maximum steering angle in the direction of wall

else if average(front slice scans) $<$ 1.5 **then**

// Bank away from the wall if frontal collision oncoming

steering angle = .25 in the opposite direction of the wall

else

// Use linear regression to simulate the line of the detected wall.

Remove scans outside 1.5 standard deviations from side slice

Convert side slice to rectangular coordinates

Get slope and intercept of the best-fit line via linear regression

// Use PD control to determine steering angle:

measured distance = shortest distance between robot and best-fit line

error = desired distance - measured distance

Kp = .45

Kd = .14

$$\text{steering angle} = \text{Kp} \cdot \text{error} + \text{Kd} \cdot -\frac{\text{measured distance - previously recorded distance}}{\text{current time - last recorded time}}$$

end if

Publish steering angle, with the appropriate scaling of 1 or -1 to account for which wall the racecar is following, in Ackermann Drive message

end procedure

In lines 2-10, we slice the LIDAR scan data into three different ranges: the front slice, which looks at data directly in front of the racecar; the side slice, which looks at data to the side of the racecar (according to whether the racecar is following walls to the left or right of it), and the back slice, which also looks at data to the side of the racecar but is angled towards the rear. Figure 1 illustrates these ranges relative to the racecar. The front slice will be used to get an approximate distance to anything in front of the racecar and works for frontal collision detection.

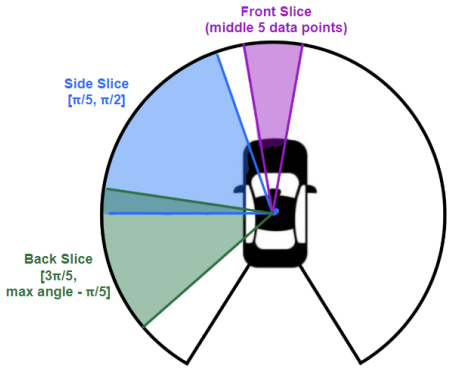

Figure 1: Illustration of the three LIDAR scan slices and their ranges which we use for following walls to the left of the racecar.

Next, we filter out any data points from the slices above a pre-determined threshold distance (line 12). The motivation for this is that any objects beyond those distances are not relevant for current decision-making since we never set our desired wall-following distance anywhere near 3 or 4 m. The front slice distance threshold is lower than for the other slices because it only needs near-range data to initiate turns to avoid collisions.

Lines 13-32 determine the course of action based on what type of environment the racecar is in. We consider the following four cases:

- Lines 13-15: If the car is not near any walls, tell the car to go straight.
- Lines 16-18: If the car has just passed an outside corner, have the car take on the maximum steering angle in the direction of the wall to maintain the desired distance from the wall.
- Lines 19-21: If there is an obstruction within 1.5 m directly in front of the car, initiate a turn. This case usually occurs at inside corners or when approaching a wall head-on.
- Lines 22-32: This is the general case: if the car is roughly parallel to and near a wall, we use PD control. We apply another filter to remove outliers (defined here as over 1.5 standard deviations), then convert our scan data into rectangular coordinates and fit a line of best fit to the side slice using least squares linear regression. This represents the detected wall that the car should reference. Finally, we apply PD control to calculate the steering angle: see Section 2.4 for a detailed explanation on our use of PD control and how we determined the K_p and K_d values for our car.

While we had the option to use PID control, i.e. have an additional integral term, we found that including just the proportional and derivative control terms

were sufficient to perform well on the wall-following tasks we tested our robot in (see Section 3 on experimental evaluation).

2.4 PD Control and Tuning

Author: Ra Mour

The PD control law is:

$$u(t) = K_p \cdot e(t) + K_d \frac{de(t)}{dt} \tag{1}$$

$$e(t) = \text{desired distance - measured distance} \tag{2}$$

The gain terms K_p and K_d must first be set to properly match the scale of the error term and control variable u(t), i.e. so that information about car distance is translated to reasonable steering angle values. These values must then be adjusted to find the right balance between the influence of the proportional and derivative terms. A higher K_p reduces the error faster but can lead to oscillations around the steady-state error. We correct for this by increasing K_d to reduce the effect of proportional control when the error is changing rapidly. Refer to Figures 6, 7, 8 to observe changes in the car's trajectory as K_p and K_d are tuned.

2.5 Safety Controller

Author: Nandini Thakur

A key technical problem was to create a safety controller which would prevent any crashes, while also not hindering the car's movement. The controller had to be robust and reliable with a variety of situations, but we didn't want it to be so conservative that our car was unable to drive fast and make sharp, narrow turns near walls.

One option was to create a highly complex controller, which predicted the path of the robot and used the prediction to influence whether the car needed to stop or change its path. An alternative was to make a much simpler controller, which relied on laser scan data to calculate simple distances to the wall and stopped the car just based on how far it was from the wall. We landed on using the simpler controller because it would guarantee less possibility for failure and edge cases. The wall follower code and any future code we write is already incredibly complex, which is why there's some room for the algorithm failing in certain environments, resulting in a crash. If the safety controller, meant to prevent these crashes, was also itself complex, the initial issue with complex controllers would present itself again. Therefore, we proceeded with the simpler design.

The pseudocode for the safety controller design is as follows:

```
def callback():
  if wall_dist < 0.45 and steering_angle < pi/25:
    stop()
```

This code has a fixed distance away from the wall that the car should stop, while also taking into account whether the car is already turning. If the car has a large turn angle while being 0.45m away from the wall, then it will be able to turn without hitting the wall. If the code has directed it to not turn, or not turn severely enough, then the car is very likely to hit the wall at this distance, and therefore we want it to stop. Through testing the safety controller while running wall following, we observe that this strategy doesn't interfere at all with the original wall following program, unless the car is actually unable to complete a turn or is angled in a way where it will hit the wall.

2.6 Key Challenges We Faced

Author: Srinath Mahankali

While developing the safety controller, there were some unexpected issues we ran into, which we were able to fix. Initially, when using the safety controller, the racecar did not move at all. It turned out that we were incorrectly calculating the distances from the racecar to objects that were in front of it. These distances ended up being negative, causing the racecar to constantly detect itself colliding with other objects and stopping movement. After calculating the distances correctly, the racecar was able to move while using the safety controller.

After fixing this issue, we observed that the safety controller did not prevent the car from colliding with obstacles in front of it. This happened because of how we initially decided the racecar should stop when it gets too close to an object. Initially, we had the following stop condition:

```
def callback():
  if wall_dist < 0.1:
    stop()
```

This stop condition might seem correct at first, but it turned out that this condition did not give enough distance between the racecar and any object in front of it. Using this code, the racecar did not have enough time to decelerate even if it correctly detected an object from the LIDAR scan data. This was fixed by simply increasing the minimum allowed distance before stopping the racecar, from 0.1 to 0.45. After this, we saw that the racecar stopped, as expected, before hitting an object. We noticed that the racecar still moved forward for a small amount of time even after its velocity was set to 0. This confirmed our hypothesis that the minimum allowed distance needed to be sufficiently large to

Figure 2: The setup used for the first experiment. We set the racecar at a 45° angle away from the wall, and the task is for it to follow the left wall. This experiment allows us to check whether the racecar can drive straight, get close to the desired distance, and correct its angle.

prevent collisions.

After this, our safety controller performed well, but we noticed that the calculated distances from the racecar to objects in front of it were around 0.5, which was very small. There were no objects which were that close to the racecar. However, the racecar was very close to the wall to its right side, and it turned out that the racecar was detecting this wall as an object in front of it. The racecar originally detected objects within the angle range $\left[-\frac{\pi}{8}, \frac{\pi}{8}\right]$, where an angle of 0 represents the line directly in front of the car. However, having such a wide angle range caused the racecar to detect the wall as an object in front of it. We tackled this issue by shrinking the angle range to $\left[-\frac{\pi}{12}, \frac{\pi}{12}\right]$. After this, the distance measurements from objects in front of the racecar to the racecar became more accurate.

3 Experimental Evaluation

We now detail the performance of our racecar on the task of wall following without crashes. To do this, we quantitatively test the racecar's ability to follow walls on both sides in a variety of environments. We also qualitatively evaluate the smoothness of the racecar's trajectory. Finally, we evaluate our safety controller's performance at preventing crashes.

3.1 Setup

Author: Nandini Thakur

An autonomous car can be always be made even more perfect and even more accurate, as long as the designer is unsatisfied. We decided to design a series of tests and measure certain metrics to evaluate whether the car's capabilities were accurate and robust enough. We created different environments to test the many scenarios an autonomous car could encounter when following a wall.

Test 1: Driving Straight
Start exactly the desired distance away from the wall, facing straight, and follow the wall for x meters. Repeat for both left and right side wall following.

Test 2: Getting Closer to Desired Distance
Start away from the path to be followed, facing straight. Repeat for both left and right side wall following.

Test 3: Correcting Angle
Start at the desired distance away from the wall and at an angle facing away from the wall. Repeat from both left and right side wall following, and with an angle facing towards the wall.

Test 4: Inside Corners
The path being followed includes a corner that must be followed along the concave part of the corner. Repeat for both left and right side wall following.

Test 5: Outside Corners
The path being followed includes a corner that must be followed along the convex part of the corner. Repeat for both left and right side wall following.

Test 6: Ignoring Outliers
Follow a path which has some random objects place between the path and the wall. Car should ignore these objects and remain at the correct distance away from the wall. Repeat for both left and right side wall following.

These tests will be executed when testing new values for the proportional and derivative constants, and when testing different velocity values. Performance during these tests will be evaluated by calculating the average distance away from the wall, observing success rate of the car in reaching the end goal point, plotting graphs of the distance from the wall and observing oscillation, and visually watching how smoothly the car drives.

3.2 Testing the Wall Follower

Author: Srinath Mahankali

We evaluate our wall follower in a variety of settings by analyzing how close its behavior is to the desired behavior. To analyze this, we consider the discrepancy between the distance from the wall measured by the racecar and the desired distance from the wall. Specifically, the error we measure is $e(t) = d_t - d'$, where d_t is the distance from the car to the wall at time t and d' is the desired distance to the wall. The optimal scenario is if $e(t)$ is as close to 0 as possible. Then, we evaluate the wall follower's performance with respect to all the goals listed above using two tests.

Figure 3: The setup used for the second experiment. In this environment, the task is for the racecar to make one right turn, followed by two left turns. This allows us to test whether the racecar can correctly turn along both inside and outside corners.

3.2.1 Moving Forwards with the Wall Follower

We first test whether our wall follower can perform the following task: starting at a sharp angle away from a straight wall, make the error $e(t) = d_t - d'$ as close to 0 as possible. The environment for this task is shown in Figure 2. Through this test, we measure the capability of the wall following controller to drive straight, get closer to the desired distance, and correct the angle to allow it to drive parallel to the wall.

We show the error over time of the racecar while following the wall in Figure 4 for three different trials. Note that the result is slightly different across trials because the initial conditions such as initial distance from the wall and initial angle away from the wall differ between trials. We found that the loss, which is

computed as

$$L = \frac{1}{T} \sum_{t=0}^{T-1} |d_t - d'|, \tag{3}$$

is 0.08 on average over the three trials. The desired distance was $d' = 0.7$ and the velocity was 0.5. This means the racecar was able to remain reasonably close to the desired distance, with $L = 0$ meaning that the racecar stayed exactly on the line of desired distance from the wall. The loss being greater than 0 is unavoidable, since the car starts at an angle and a different distance away from the wall.

3.2.2 Turning Corners with the Wall Follower

Next, we test whether our wall follower can turn across corners, both inside and outside corners. We use the environment shown in Figure 3, where the racecar must move around three corners by performing one right turn followed by two left turns. We present the error over time of the racecar over three different trials in Figure 5. Based on the loss which has formula shown in Equation (3), we find that the average loss over our three different trials is 0.40.

While this is much higher than the average loss for the previous task of driving at an angle, this is reasonable since the task is much more difficult. The loss is likely to be high because even if the racecar makes the turn correctly, it may not make the turn sharp enough and thus would need to adjust afterwards. However, our environment has almost no distance between consecutive turns, making it likely that the racecar would not have enough time to adjust to keep the error close to 0. Furthermore, we find that in all three trials, the racecar successfully completed all three turns, showing that our wall following controller is capable of reliably turning across both inside and outside corners.

3.3 Qualitative Evaluation of Wall Following Behavior

Author: Sarah Zhang

In this section, we will discuss how we used qualitative analysis of our robot's behavior to test, improve, and tune it. While the quantitative analysis methods described in the above sections are undoubtedly the most concrete form of evidence, more quantitative techniques like visual observation enabled us to make changes on the fly. Further, in some cases, visual checks provided faster and more informative feedback than the data we can read from the car.

Some extrinsic measures we considered include the smoothness of the vehicle's motion (e.g. lurching motions, sharp turns), wobbliness of the wheels, and whether the car made contact with anything it wasn't supposed to (e.g. nearby walls or small objects),

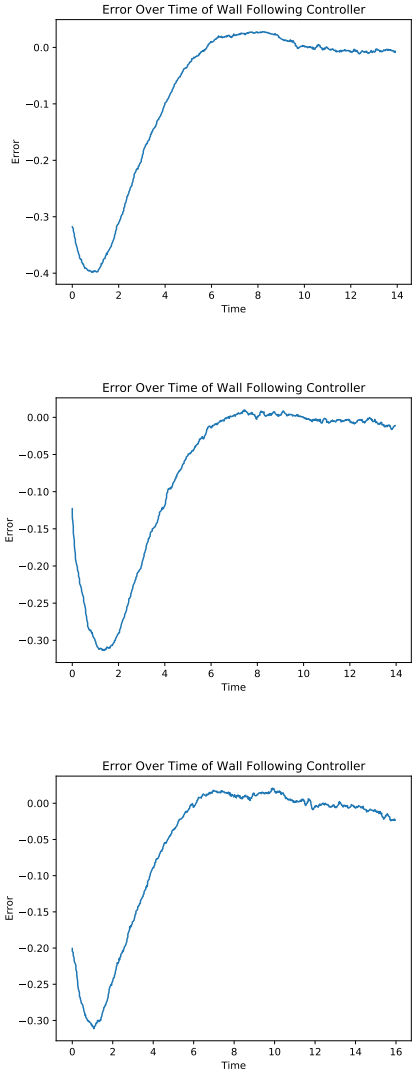

Figure 4: Error of the wall following controller over time in the task where the racecar starts at a sharp angle away from the wall. Note that the error sharply decreases then returns back towards 0, after the racecar starts turning to a direction parallel to the wall. We test the controller's performance on this task for three trials. Here, $K_p = 0.45$ and $K_d = 0.14$.

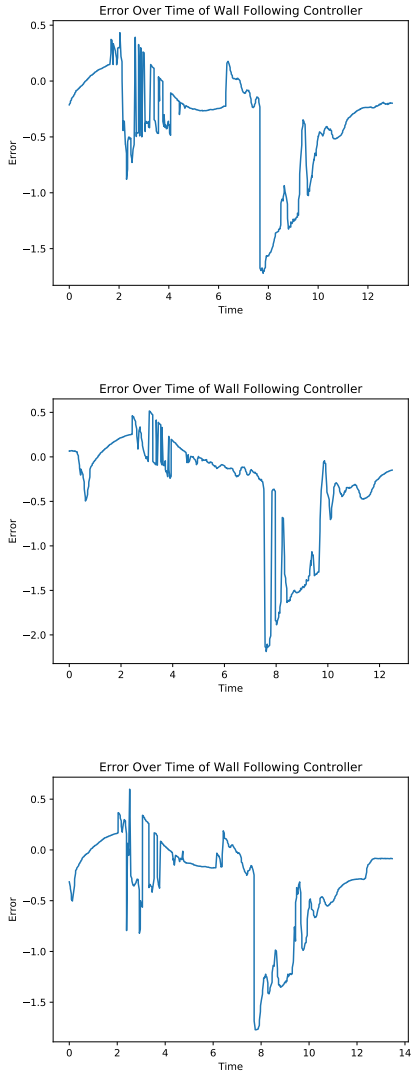

Figure 5: Error of the wall following controller over time where the racecar has to make one right turn followed by two left turns. The error is highly oscillatory because the racecar is making turns. As the racecar adjusts its steering angle using our PD controller, the error oscillates between positive and negative. We test the controller's performance on this task for three trials. Here, $K_p = 0.45$ and $K_d = 0.14$.

In particular, we evaluated Test 6: Ignoring Outliers completely via qualitative analysis. Our implementation of this test was to set the robot up such that it would encounter thin table legs, chair legs, and backpacks near its path. In this environment, we found that our wall-following algorithm was somewhat robust to distractions. When a table leg was in between the wall and the car, the car reacted to the table leg by veering slightly away but would correct itself soon after passing it. This is likely due to the fact that the width of the table leg means it does not greatly affect the detected wall as calculated via lease squares linear regression. Small distractions like backpacks did slightly affect the trajectory but the robot still avoided collision without safety controller intervention.

We also used observations on the wobbliness of the racecar's wheels while the wall-following algorithm ran to tune the D-term's gain in the PD controller. We saw that when K_d grew too large in proportion to K_p, the wheels were extremely wobbly and changed direction back and forth too often: this is reflected in the frequent spikes in error in Figure 5. This made sense as the purpose of the D-term is to counteract the oscillations caused by the P-term, and aggressively over-correctly the P-term would lead to too many minute corrections.

4 Conclusion

Author: Jorge Tomaylla

Throughout this lab we aimed to implement a robust wall following algorithm that navigates diverse environments, as well as constructing safety control measures to prevent the race car from being damaged. We tested our implementation against a variety of tests that challenged the car's ability to be safe but still follow its wall following mission regardless of initial orientation, target wall, and distance from the wall. The results confirm a robust implementation that renders our car functional in most environments. Although we consider the results to be successful, we acknowledge there is room for further exploration and improvement. For instance, to create more chaotic, noisy environments that will further test the safety and accuracy of our implementation. There are also mathematical models and better regressions taht could help us have a more complex visualization of a wall so that the car can better predict changes in steering angle according to the wall's surface. These changes would strive to reduce the PD error and create a smooth wall following path that can more accurately predict where to go even with noisy data.

5 Lessons Learned

5.1 Ra Mour

I feel more confident working on code collaboratively in real-time after this lab. It was clear that we could deal with multiple tasks efficiently if we coordinated.

This happens most naturally when someone on the team has a clear vision of the steps that need to be taken; then it's a matter of each team member taking responsibility for a task according to their own interests, skills, and availability. Communication remains the key element in this whole process, and I've realized that this skill must be adapted and honed for each new project and group that one work with.

5.2 Srinath Mahankali

This lab was my first experience working with a real robot, and prior to this, I only worked on robots in simulation. Thanks to this lab and the wall following simulation lab, I also learned about PD control, which was a great introduction. When we used PD control for our wall follower in an extremely messy environment with people walking around and other objects in different places, the racecar was frequently distracted by the other objects and could not reliably follow the wall. From this, I learned the importance of other sources of information for these tasks, such as vision. Furthermore, I practiced my communication and collaboration skills through this lab through dividing tasks evenly.

5.3 Nandini Thakur

The technical lessons I've gained from this lab reach far and wide. From exploring PD tuning to experimenting with a safety controller that is reliable but still trusting of the car's autonomy, every line of code written often took hours behind it to understand the mechanics of the robot. Outside of the technical realm, I learned a lot about coordinating with my team, communicating with other teams, and optimizing my time well. Through creating the report and slides, I started to understand the importance of documenting every step of the testing process, as well as how to communicate findings in a clear way to others.

5.4 Jorge Tomaylla

Throughout this lab I reinforced a lot of communication and technical skills. For instance, for the technical skills I developed, I learned how to properly ssh into the race car environment and the necessary steps to command it and upload code to the car, including using teleop and subscribing to the correct drive ROS topic. In the process where we selected the best PID controller among our group, I got to see alternative approaches that were simply mesmerizing to me, as sometimes they would be simple ideas that make a profound difference in the driving algorithm, and sometimes it would be an out of the box, complex solution that led to the same results. This brings me to skills I learned when collaborating with others. The group nature of the project required us to hold great communication to coordinate who will work on what parts of the project, and how to divide the work equally. We all helped each other in whatever way possible, even if it was not the assigned part, and that contributed to our success and performance.

5.5 Sarah Zhang

For me, this lab served as a sort of "onboarding" into the universe of working with a team and with physical robots. In a technical sense, this included getting familiar with ssh-ing into the car's environment and transferring files between my local machine and the car's environment, as well as gaining a better understanding of using rosbags after using them for our data analysis in this lab. It was also very fascinating to see how my teammates approached the wall following problem and to compare our algorithms and perspectives. Over the week, I've also gained a better sense of the type of scheduling compromises necessary to fit the needs of my team and the robot's availability. I also relearned to not assume things will work smoothly on the first go, especially when we are working with complicated systems that have to interface with each other, as some things took longer to set up than we expected leaving us with greater time crunch than desired.

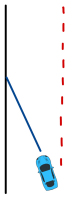

Figure 6: No Error Correction

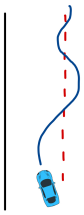

Figure 7: Proportional Controller

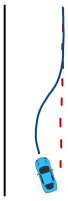

Figure 8: PD Controller