Lab Report: Wall Following

Team 1

May Huang
Mohammadou Gningue
Haley Sanchez
Nikhil Kakarla
Neil Chowdhury

Class

July 3, 2024

1 Introduction

Our team's objective for this lab was to take a wall following algorithm that was written for a robot in a simulator and implement it in practice. We wanted to guide a physical robot along a wall as well as introduce a safety control feature to prevent crashes. This serves as an introductory exercise for programs for the robot and debugging during the physical testing process, which will only become more involved and complex.

Our robot's wall-following capabilities prioritize robustness and accuracy in order to keep it safe. We wanted the robot to be resilient to error in readings and self-correct if too close or far from the wall. We ensured repeatability of successful wall following runs for four test cases, which included: the robot starting too close to the wall, too far from the wall, completing a right turn, and completing a left turn. The safety controller is an additional fail-safe, also used in case of emergencies, i.e. someone walking directly in front of the robot.

2 Technical Approach

2.1 Wall Follower Simulator and Algorithm

The wall follower algorithm, originally coded in simulation, uses proportional and derivative (PD) control. The robot processes distance readings from a LIDAR scanner, calculating cartesian coordinates of the wall locations and applying a line of best fit to find the wall. We then find the actual distance between

the robot and the wall, and the difference between this and a set desired distance is used as error in the control algorithm. The following formula is used by the PD controller to set the steering angle:

$$\text{Steering Angle} = K_P \cdot e(t) + K_D \cdot \frac{de(t)}{dt}$$

with $e(t)$ being error with respect to time.

In the simulation, K_P and K_D were set by setting K_D to 0 and incrementing K_P until the controller behaved ideally, turning at an appropriate distance and speed to avoid hitting walls. Then, K_D was also tuned by increasing the value incrementally. Using these values, we were able to develop a robust and reliable wall following algorithm to be introduced onto the robot.

2.2 Safety Controller Algorithm

Ordinarily, we hope that the commands we send to the robot are robust, and will not cause collisions between the robot and other objects or people around it. However, mistakes in programming are inevitable, and unexpected situations arise (e.g., a person suddenly running in in front of the robot). Thus, it is essential to implement a safety controller that can override default robot actions.

We first came up with the scenarios when our safety controller should activate: when there is an obstacle directly in front of the robot's line of motion with which the robot would collide in a short amount of time, and when there is an obstacle very close to either side of the robot. We tested stopping a program to just go forward on the simulator, and then we tested trackers to measure the distance to the nearest obstacles in front of, next to, and behind the robot. When we had a functional safety controller on the simulation, we tested and fine-tuned our controller on the robot.

Our algorithm divides the LIDAR scan into the left, front, and right side of the car.

- The left measures LIDAR scans from $-140° < \psi < -15°$.
- The front measures LIDAR scans from $-15° < \theta < 15°$.
- The right measures LIDAR scans from $15° < \eta < 140°$.

For each side of the car, we compute the closest distance to the car as the mean of the closest 2% of returned LIDAR scan data. This allows us to confidently determine when an object is in the way. We then stop the car if it is projected to hit an obstacle within a certain time (1 second for the front and 0.3 seconds for the sides), which allows us to dynamically adjust the speed of the vehicle.

Compartmentalizing by the side of the vehicle allows us to put different thresholds on what is considered a dangerously close distance to the car. Our basic algorithm is

$$v_{pub} = 0 \quad \text{if} \quad d < k_{side} \cdot v_{curr}$$

where v_{pub} is the velocity to be published to the car, v_{curr} is the current velocity of the car, d is the closest distance to the car, and k_{side} is the stopping time ($k_{side} = 0.3$ for the sides and $k_{side} = 1$ for the front).

2.3 Implementing Algorithm with Physical Robot

After we had the algorithms in place and working in the simulator, we needed to then transfer the scripts to our working robot. It took us a bit of time to get acclimated with the pipeline of our programs from our computers to the robot and vice versa. From time to time, we additionally had hardware problems come up that we needed to resolve in order to continue our work. Once we were familiar with the process and the hardware, we began implementing our algorithms with the robot. Our first approach to this was to use whose script had the highest accuracy on the test cases from the wall following lab in the simulator. However, to our surprise, the wall following code that worked the best in the simulator did not necessarily work the best with the real robot. We tried out multiple team members' code until one of them seemed to be the most promising and had the fewest initial bugs. We still had to fine tune this program in order to make it work better on the robot in real life. We also needed to spend more time focusing on understanding how our units used for distance in the simulator translated into the real world. We preformed fine-tuning of K_p and K_d and multiple rounds of testing and experimentation in the real world with the robot in order to arrive at our finalized algorithm for wall following, which will be further discussed in the Experimental Evaluation section of the paper.

3 Experimental Evaluation

3.1 Wall Follower

3.1.1 Defining Success

Before beginning our tuning and testing phases of the wall follower code, we first had to define what we meant by success for our robot. After deliberating as a team, we identified three core areas that we were going to use to measure to determine the success of our wall follower.

The first was the overall error of the robot. We decided to measure this error both extrinsically, using a measured line of tape on the floor, as well as intrinsically, by comparing the robot's internal estimated distance from the wall

to the desired distance for the task. Our goal was to minimize the average error across the runs and therefore minimize distance from the desired location as well as oscillations.

Additionally, we wanted our robot to be able to deal with changes in the environment quickly. In this case, the changes in the environment was the change in shape of the wall, either turning into or away from the wall. In the case of environmental changes, we wanted to minimize the time it took for our robot to adapt to the new wall direction as well as reduce the amount of oscillation to ensure a smooth trajectory.

Finally, we wanted our robot to be robust. In this scenario, we defined robustness as the ability to operate in a variety of locations and walls. We ensured this by running tests in different environments with different lighting conditions to make sure our code was robust and general.

$$loss = \frac{1}{N}\sum_{i=0}^{N} |distance[i] - desired\_distance|$$

Figure 1: Loss Function

3.1.2 Kp and Kd Tuning

We decided to tune K_p and K_d in order to find the best values for our PD algorithm for our robot. We began by setting K_d to 0 and experimenting with K_p.

We calculated the loss of each run with using K_p values of 2, 4 and 6. We had the robot perform the wall following code on the same wall and going the same distance. The results can be seen in figure 1 below. Here we can see that the run with a K_p value of 4 has the lowest loss of 0.1353 m and the smoothest oscillations. We set K_p to 4 for all further experimentation.

We calculated the loss of each run with using K_d values of 2, 4 and 6. We had the robot perform the wall following code on the same wall and going the same distance. The results can be seen in figure 1 below. Here we can see that the run with a K_p value of 2 has the lowest absolute loss of 0.1404 m and the smoothest oscillations. We set K_d to 2 for all further experimentation. This incremental testing allowed us to isolate parts of the system and objectively determine the best values for our multiplicative constants.

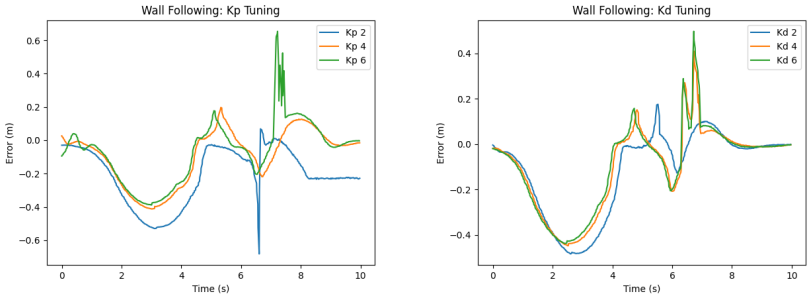
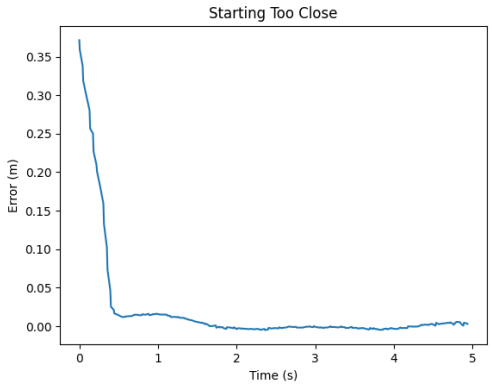


Figure 2: Time vs. Loss Graphs for Kp and Kd Tuning

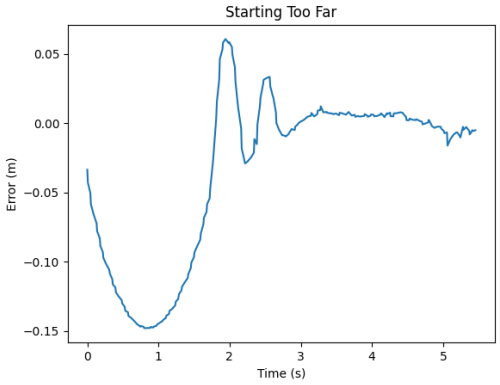
3.1.3 Driving Experiments

1. The first test we completed was when we placed the robot closer to the wall than the desired distance. The goal was for the robot to be able to adjust and get itself to be the described distance from the wall while driving forward as we ran the program. We achieved an average of .023 meters of loss on this experiment shown below. Also, we see a quick descent in error, within approximately 0.5 seconds, and very little oscillation as the trial progresses. These are both characteristics that are desirable in our car.

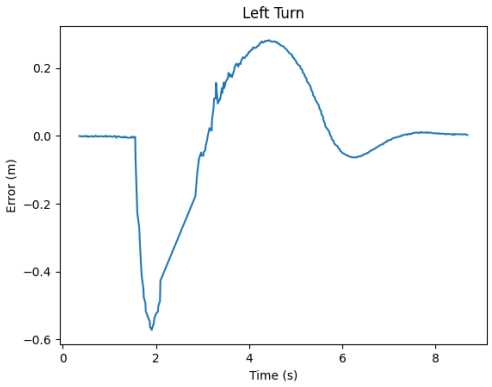


2. The second test we completed was when we placed the robot farther from the wall than the desired distance and face it away from the wall. The goal was for the robot to be able to adjust and get itself to be the desired distance from the wall while driving forward as we ran the program. We achieved an average of .047 meters of loss on this experiment shown below and achieved a settling time of about 3.2 seconds. With this test, we also saw some oscillations in the robot. This was expected as the starting angle

of the robot was not in line with the direction that we wanted. However, the oscillations were minimal and were properly damped by our derivative controller.

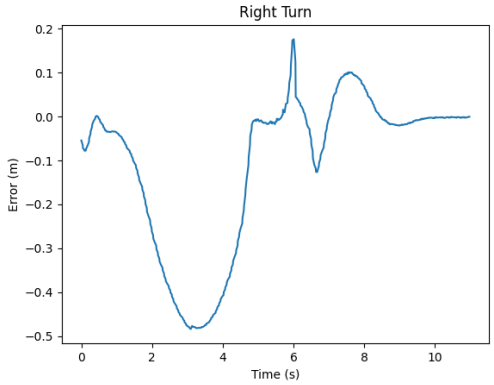


3. Our third test was for the robot to follow a wall that turned away from the robot. In this case, we were following the left wall and turning towards the left. The graph of the error over time is shown below. At time step 2, you will see the immediate drop in accuracy as the wall is farther from the robot, but you will also see the robot quickly adjust and return to an error of zero. Additionally, this test resulted in very little oscillation, as the robot quickly overshot and then settled down to an error of zero. Our robot was able to turn and readjust to the newly angled wall within approximately 4.3 seconds. This again was a good result, as our robot was able to adjust to external conditions and not oscillate too much. Overall, we achieved an average error of .108 meters.



4. The last test that we ran was for the robot to follow a wall that turned into the robot. In this case, we followed the left wall into a corner that

necessitated the robot turning to the right at time step 3. The test also featured a change in the shape of the wall at time step 6, where the wall jutted out towards the robot. The graph of the error over time is shown below. This example shows the performance of our robot as it was able to quickly adjust to the right turn and return to zero error. Moreover, when the wall jutted out towards the robot, the robot was able to quickly oscillate and return to zero error without hitting the wall. Therefore, this experiment was largely a success. We achieved an average error of .104 meters.



3.2 Safety Controller

3.2.1 Defining Success

After deliberating as a team, we identified two core areas that we were going to use to measure to determine the success of our safety controller.

The first was if the distance algorithm itself was working. We kept the robot still and placed a variety of objects of various widths at different sides of the car and at various distances from the vehicle. The narrow objects we tested were a 3-inch wide x 12-inch tall box and a human leg. The wide and large objects we tested were a wall, and the wide brick red-box. We objectively measured the distances with a tape measure and compared the measurements with our algorithm.

Secondly, we wanted the car to react to static and sudden stimuli. We define static stimuli as objects that are laid along the intended path of the robot before the car starts wall-following and sudden stimuli as objects suddenly placed at a close distance in front of or to the sides of the car while the car is following its path. We incorporated this into the tests that we ran the algorithm against.

3.2.2 Distance Threshold Tuning

Our goal was to find the smallest percentage of LIDAR data needed to accurately detect the closest object to the car. If we only rely on one or two data points, we can be susceptible to the intrinsic noise and error of the LIDAR scanner, which could cause false positives or negatives in our close-object detection algorithm. However, if we use too many points, a narrow object dangerously close to the car might be filtered out because the object corresponds to very few LIDAR data points.

Our LIDAR scanner stores 1081 distance data points over a range of 240°. Thus every LIDAR scan represents 0.222°. At 1 meter away from an object, an object that is 1-inch wide would correspond to 5 LIDAR scan data points, or .4% of our LIDAR scan spread. We did not believe this would be sufficient given how few data points there were. We settled on ensuring we could detect an object 1 meter away from the car that was 3-inches wide. This would correspond to 20 data points or 1.6% of our LIDAR scan data. Our algorithm calculated the object with the closest distance as the mean of the closest 1.6% of LIDAR scan data rounded to the nearest integer. This does not necessarily filter out narrower obstacles; it gives us further confidence that an obstacle is there if it's shorter than 3-inches wide. As the robot approaches a narrow object, it fills up the viewing range, so we are more confident that we can avoid it.

We tested close-object detection on multiple objects of various widths at multiple distances. We measured the distances objectively with a tape measure and compared the measurements with our algorithm. We found an average error of 0.4% between our calculated distances and the actual distance, which we found acceptable for our safety controls.

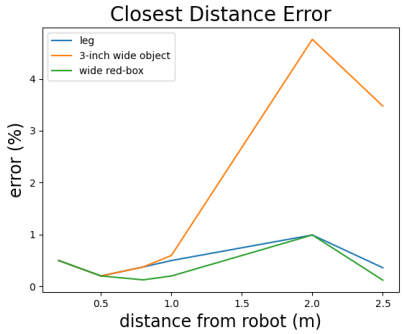


Figure 3: Measured distance Percent error by object

3.2.3 Driving Experiments

We tested both static and sudden stimuli in multiple environments.

1. For static stimuli, we tested putting the red box in front of the intended path of the robot and waited for the car to reach it and observed its reaction. We did the same test with the red box to the left and right side of the vehicle in close proximity to its intended path. We tested these both when the car was following a straight line and when going around a turn.

2. For sudden stimuli, we walked in front the car, put a red wide box, and a narrow 3-in wide tube right in front of the car while it was following and tested to see if it would stop. We then removed the object and observed it continue on its wall following path.

In both cases, we found that the car was able to avoid the obstacles as seen in Figure 4 and Figure 5.

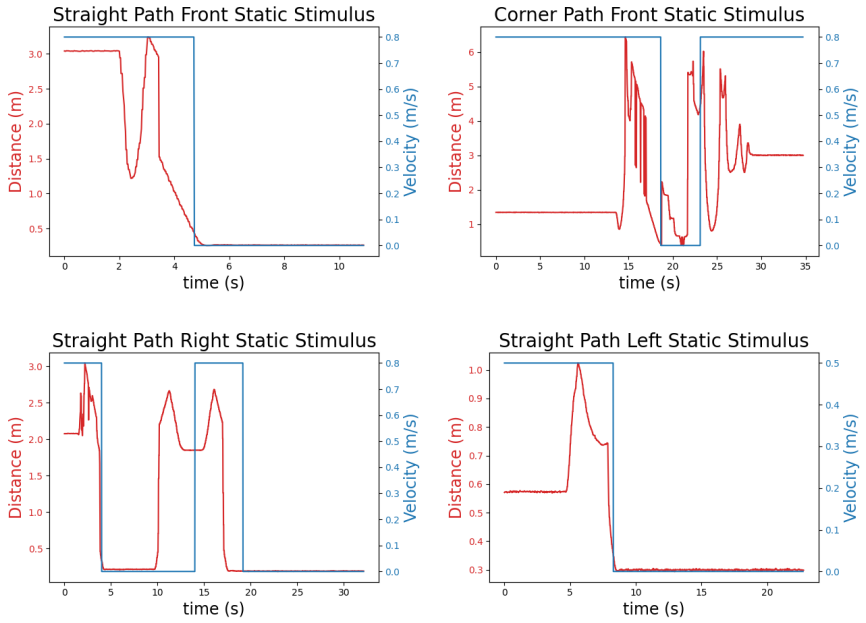


Figure 4: Measured distance vs Velocity for Static Stimuli

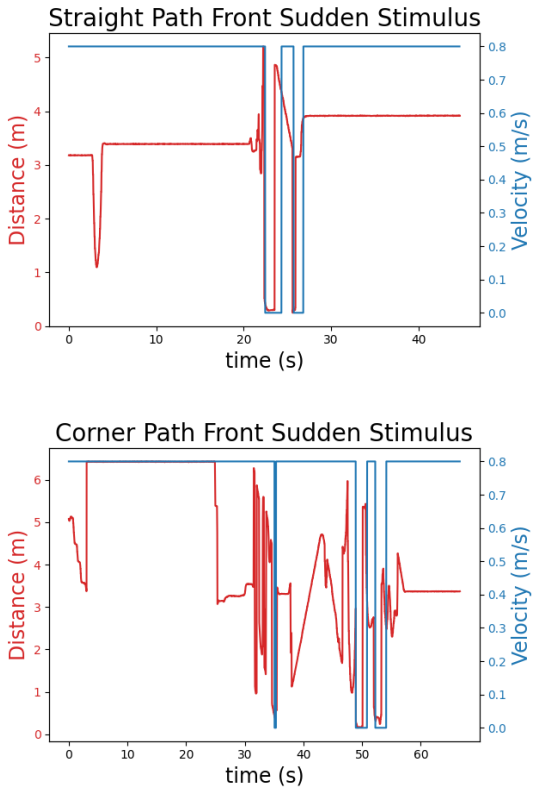


Figure 5: Measured distance vs Velocity for Sudden Stimuli

4 Conclusion

In this design phase, we achieved three major goals. The first was gaining familiarity with the robot interface and more fully understanding how to deploy and test code in the robot operating system. Additionally, we were able to develop and implement a wall following algorithm that allowed our robot to use LIDAR to follow a variety of shaped walls at any given distance. We achieved an average accuracy of .071 meters and were able to follow a variety of differently shaped walls with good accuracy, minimal oscillations, and good settling time. Finally, we were able to implement a safety controller to prevent our robot from crashing. By utilizing the different levels of control for the robot, we were able to create a overriding fail safe that can be deployed throughout the development cycle on future labs. This safety controller will ensure the safety of our robot regardless of the code running on top.

In the future, we'd like to enhance our safety controller by identifying the

shapes, sizes, and locations of all nearby objects, instead of having a simple breaker when we get too close to an object. This could allow us to implement obstacle avoidance in our pathing directly. Additionally, once we have the object detection in place, we could use this to implement path adjustment so that we are still able reach our desired goals even if a roadblock is the way.

Within our experiments for tuning K_p and K_d, we found that the loss was the lowest when we left K_d as 0. However, we know that in general principle, PD control is more powerful than just P control. So we chose a K_d value that gave the lowest error and continued from there. Since we only used one set velocity speed during our experimentation, we hypothesize that at faster speeds the K_d value with have a higher impact on robot wall following performance. if given more time, we would want to run more experiments with the robot performing our wall following algorithm at various higher speeds in order to test our hypothesis further.

Overall, we achieved the goals of the lab while learning many valuable lessons about robotic algorithm development. There is no more work to be completed before the start of the next design phase, but we remain eager to implement camera functionality that will augment the LIDAR scans.

5 Lessons Learned

5.1 Haley

The debugging process was much easier and faster with a team of 5 than it was when we all individually had to complete the labs. I really appreciated having a group to bounce ideas off of and to utilize each others strengths to complete the tasks as efficiently as possible. Additionally, when we initially began taking the wall following code and translating it into the real life robot, it was very important to understand how units and certain calculations worked within the wall follower simulation code. Being able to contextualize your code and data makes applying it to similar, but different, situations and tasks much easier.

5.2 May

This week showed me that it really pays to be strategic with time management and planning. Having the robot as a limited resource was a challenge to work around, and much of that could have been mitigated if as a group we clearly blocked out our availability and delegated work with the robot ahead of time. It was much more seamless to debug in a group of people than it was to do so on the simulator, so I really valued working with the others during experimentation and asking eachother for help even with our more individual tasks.

5.3 Mo

I learned that one's first approach to solving a problem can be limited by a lack of knowledge of the environment. Our first solution for safety control seemed so obvious and almost too easy, until we tested it out and saw the fundamental flaw in how we calculated a close distance. It was eye-opening and refreshing to have to take a different approach. Working with others and combining ideas is a really fruitful experience and led to quicker issue-solving. I also learned a lot about adapting to situations. I got injured during the height of working on the project and the team and I were able to adapt, and delegate roles differently. Our team was very accommodating and helpful while still allowing me to contribute equally to the project.

5.4 Neil

I learned the importance of compartmentalizing and modularizing code. One contribution I made to this lab was a distance controller that we use for wall following, as well as the initial version of the safety controller (although we later changed distance measurement to a simpler algorithm for that). Since it was likely that this code would need to be reused and understood by different groups of people, I tested it in the simulator rigorously and made it a well-defined function that could stand on its own without having to dig into its internals – simply give it the parameters it expects, and it will return the appropriate output. For those who need more granularity into how it works, I also wrote comments each significant line of code. I noticed that for any team, it's important for a member to use someone else's contribution with as little friction as possible. Proper coding practices helps significantly.

5.5 Nikhil

One of the most important things that I learned throughout this process was the importance of delegation. At the beginning of the project, all of our team members tried to work on everything all at once. This created massive inefficiencies as many of our team members were not being as productive as possible with their time. An important shift occurred when our team started to more cleverly delegate tasks and ensure that each person had a more isolated goal to accomplished. This allowed us to be much more efficient with our robot time and ensure that everyone was able to work on some aspect of the project. We are continuing this trend in the next lab and have already outlined our tasks and timelines for next week.

6 Credits Page

6.1 Nikhil

- Defining Success
- Driving Experiments 3 and 4
- Conclusion

6.2 Neil

- Safety Controller Algorithm
- Safety Controller Experimentation

6.3 Mo

- Safety Controller Algorithm
- Safety Controller Experimentation

6.4 May

- Introduction
- Wall Follower Algorithm

6.5 Haley

- Implementing Algorithm with Physical Robot
- Kp and Kd Tuning
- Driving Experiments 1 and 2