Lab #3 Report: Wall Follower

Team #6

Natalie Muradyan
Tiffany Horter
Nisarg Dharia
Andrew Manwaring

6.141 - Robotics: Science and Systems

March 4, 2022

1 Introduction - Natalie Muradyan

In recent years, autonomous cars have become more popular and gained higher demand. Many companies have explored the idea of self-driving cars, and there are already some models of cars that can drive in the streets with no assistance.

In this lab, our team explored some of the key concepts in autonomous driving. Our goal was to design a car that could safely follow a wall without external control.

In the previous labs, we have already designed a simple wall follower that, given a side (either left or right) and a distance, ensures that the car follows the wall on that side and keeps the specified distance from it. For lab 3, our first challenge was to integrate the code from the last labs and implement it on the actual car. This task turned out to be particularly tricky for our team since our car operates a Velodyne LiDAR, which was both shifted by around 90° and also did not get data from objects within 0.5 meters. Therefore, we modified the codebase a lot and tested it on the car to make sure we were getting accurate data to follow the wall.

The second challenge of the lab was designing a safety controller. The idea is that before adding complicated algorithms and testing on the car, it's important to have a way to avoid crashes. Since our car costs more than $4.500 and similarly real-life autonomous cars are quite expensive, and hardware malfunctions can hinder testing and distort the data, such a safety controller is essential to the design. Our team had two main ideas to approach the problem: the sector

method and the curved line method. These methods helped us parse the data received from the Velodyne LiDAR and identify any obstacles that could lead to crashes. After testing both methods, our team decided to pursue the curved line method as it was less conservative.

This lab introduced many interesting ideas that we see in the current self-driving cars. It reminded us to not only think about performance but also ensure fault tolerance.

2 Technical Approach

2.1 Pre-processing Scan - Tiffany Horter

Velodyne LiDAR Difference from Simulation

As our team's robot had a Velodyne LiDAR, there were discrepancies between the LiDAR scan that worked in simulation and the LiDAR results we received on the robot itself which affected the ability of the robot to understand its environment. One such difference is that the Velodyne LiDAR was mounted at a 90 degree angle from the expected 0 degrees pointing forward as shown in the figure below, causing the scan angles to be off. The red arrow on the right represents the LiDAR's expected 0 angle, while the simulation's expected -90 to 90 range are represented by a red half-circle above the puck on the right.

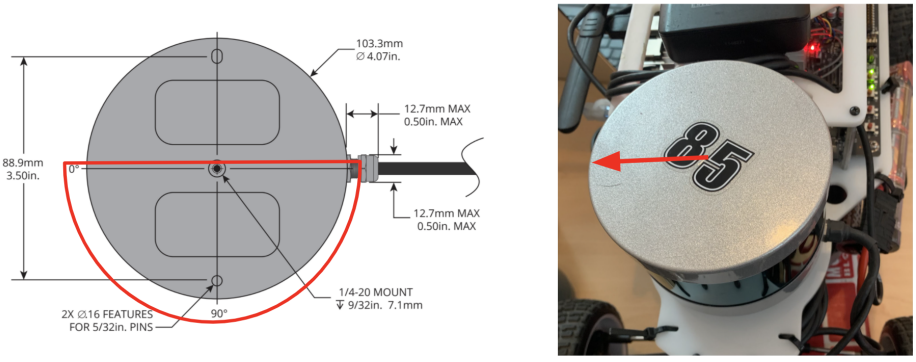

Figure 1: Velodyne schematic showing angles

Another challenge with the real LiDAR data was that the Velodyne LiDAR scans slower than the /scan topic that publishes the data. As such, some of the data for each scan would come back as infinite as there had not been time to scan through the entire range of the scan. In addition, all distances for the LiDAR were doubled and the Velodyne LiDAR in our testing was unable to

see any object within 45 centimeters of it. These differences were a significant challenge for our team to resolve as the majority of the team's time was spent debugging the LiDAR data. This is another example of the overall robotics challenge of making the simulation code and reality work together.

Resolving the differences through pre-processing the scan data

To solve these challenges, we decided to publish a new pre-processed node of that scan data so that the robot would have accurate and understandable information about its environment. To do this, we took in the scan data of the "/scan" topic which contained the LiDAR data and performed several transformations on that data to resolve the issues enumerated above.

We fixed the issue where the scanner did not have time to complete a full rotation by combining two scans. By keeping the prior scan and every other time a scan was received conservatively combining the two scans by taking the minimum range of the two scans, we were able to retrieve a scan range that corresponded to the full scan. In addition, we fixed the doubled distances in the original data by halving the values of the range measurements for each element of the LiDAR scan. We also performed minor data cleaning by removing the remaining infinite distance points to eliminate those that were simply too far away or too reflective for the LiDAR to detect correctly.

To fix the issue with the skewed angle, we chose to transform our data to match the angle ranges we expected from the simulation. First, we created a new message type to store the angles as well for clarity and the ability to have a non-consecutive set of scan angles. Then, the angles were transformed according to the following relation to shift the 0 angle to the front of the car and have all angles be in the range from [-pi, pi]:

$$\theta = \theta - \pi/2$$

$$\text{if } \theta < -\pi \text{ then } \theta = \theta + 2 * \pi$$

After this transformation, we resorted the data by angle to more easily be able to slice the LiDAR scan data in the future. Finally, we published this processed data to a new node to use in all subsequent programs, such as the wall follower and the safety controller.

By preprocessing the data, the conflict between the robot's input and processing that worked in simulation was resolved, improving our ability to understand where the wall is.

2.2 Safety Controller - Nisarg Dharia

In order for our racecar to be usable in a wide range of scenarios, we needed to make sure it was able to keep itself out of harm's way. As such, we devel-

oped out safety controller with the purpose of preventing collisions between our racecar and any obstacles in it's path. To execute this effectively, we broke the process down into 3 steps: projecting the path of the racecar, scanning for obstacles along this path, and reacting appropriately depending on the nature of the obstacle. In doing so, we were able to make a safety controller that was robust enough to protect the vehicle, without being so restrictive as to limit the racecar's abilities.

Projecting the Path

Our team knew that we wanted our safety controller to look both directly ahead of the racecar and to the side in which the car was turning, if applicable. Thus, our initial approach was to draw two straight lines, one from either side of the robot, and look for any obstacles within this sector. The first line would be straight ahead from the wheel located opposite the turning direction of the racecar, while the second line was angled proportional to the steering angle, as illustrated in the figure below.

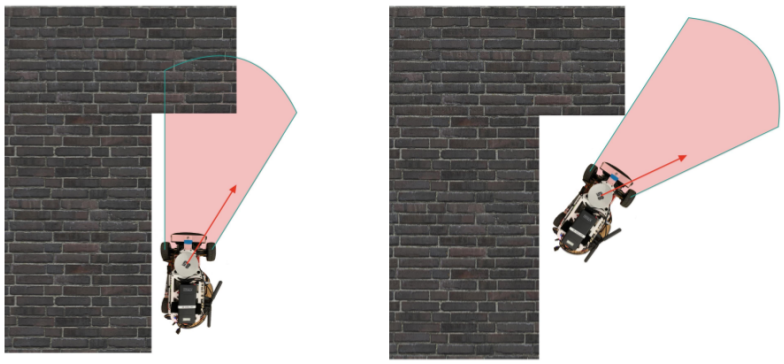

Figure 2: Before (left) and after (right) of the racecar's scanning area while executing a turn

However, we quickly realized that this method was too conservative, as there were many occasions where the car was already turning away from an object, but would still stop due to the scan directly ahead. Figure 2 shows an example of this, as the turning racecar would have stopped when sensing the wall directly ahead (shown on the left), even though it's turn would have safely avoided the wall had it continued (shown on the right).

In order to fix this issue, we developed a new method that projected a more exact path of the car based on it's steering angle. First, we used the equation

$$R = L/tan(\theta) \tag{1}$$

to determine the car's turn radius, where L is the length of the racecar's wheelbase and θ is the racecar's steering angle. Once computed, we used this turn radius to determine the circular path that the car would follow if the steering angle remained constant, given by

$$x^2 + (y + R)^2 = R^2 \tag{2}$$

where using the $y + R$ term offset the circle by it's radius, putting the car along the edge of the circle rather than at it's center. Finally, the Ackermann drive stamp calculates steering angle from the middle axis from the car, so we offset the turn radius by 25 cm on either side to account for the width of the car, giving the equations

$$x^2 + (y + R)^2 = (R + 0.25)^2 \tag{3}$$

$$x^2 + (y + R)^2 = (R - 0.25)^2 \tag{4}$$

As a result, we were able to project the path of the right and left wheel independently, as shown on the left of the figure below.

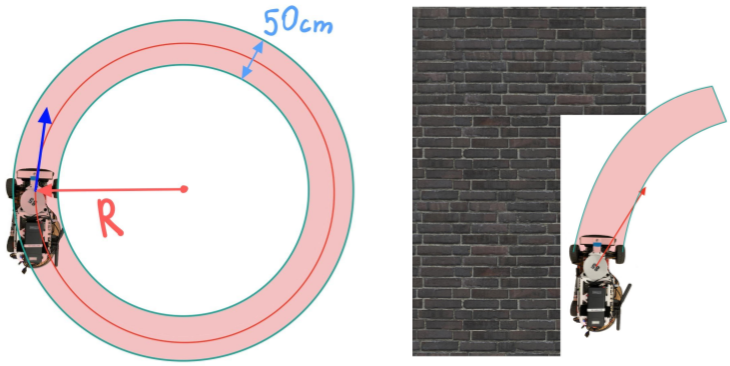

Figure 3: The racecar uses it's turn radius to determine the circular path it's on (left), and projects this path onto the area in front of it (right)

As indicated in the image on the right of figure 3, this new approach no longer suffered from the overly conservative stopping issue as the first approach, but was still able to look for obstacles straight ahead and on the turning side of the racecar as desired.

Scanning for Obstacles

Once we had our racecar's path projected, we needed to scan for any obstacles along this path. To do this, we first filtered our LiDAR scan data to only look at points between $-\pi/2$ and $\pi/2$ radians, since points outside of this range are

located behind the car and would not be a collision risk. For each of the remaining points, we then converted the data from polar coordinates to x and y values. Next, for each x value we calculated the corresponding y value of the left and right lanes by plugging the x value into equations (3) and (4). Finally, we checked to see if the y value of the data point fell in between the calculated y values of the left and right lanes.

If the data point was not in between the lanes the point was ignored, as it was not in the path or the racecar. However, if the point did indeed fall between the lanes, then it was tested for one final distance check. In this check, the straight line distance between the racecar and the point was compared with the safety protocol distance, calculated as

$$\text{safety distance} = 0.3 * \text{velocity}^2 + 0.5 \tag{5}$$

If the point was closer than this calculated distance, the racecar deemed an obstacle to be in it's path and the safety controls were triggered. (Note that the safety distance contained a velocity squared term as stopping distance scales with the square of velocity, as well as a minimum stopping distance of 0.5 meters, which was approximately the minimum range of our LiDAR).

Reacting to Obstacles

If an obstacle was detected, we needed to respond to avoid a collision. Initially, we decided to simply set velocity to 0, but we noticed that in some scenarios we just wanted to slow down, not stop. In particular, if an object was moving along at a constant speed in front of the car, we would want to maintain distance from the object at a steady speed instead of constantly switching between stopping and full speed. As a result, we decided to scale speed down when an obstacle appeared, and then scale back up slowly as it disappeared. This was done by slowing down the car according to the equation

$$\text{new\_velocity} = 0.5 * \text{current\_velocity} - 0.1 \tag{6}$$

and then speeding back up when an obstacle was no longer in range according to the equation

$$\text{new\_velocity} = \text{current\_velocity} + 0.2 \tag{7}$$

Through the use of these two equations, we were able to still stop fast enough to avoid collisions with stationary objects, while also being robust enough to slow down and follow mobile obstacles.

2.3 Wall Follower - Andrew Manwaring

In a first step of gaining a broad understanding of its environment, the racecar was tasked with identifying and following walls. The goal of the wall following

node is to find, map, and track a path in the racecar's environment at user input of desired distance away from the wall and side of tracking. Our implementation uses on carefully sliced and linearly regressed ranges of the previously mentioned processed scan data from the Velodyne LiDAR system on-board our vehicle. These ranges are used to obtain a minimum distance to the front wall and side wall. If the front wall is not blocked, the racecar will maintain desired_distance from the side wall using a PD controller. A blocked front wall will cause a hard turn to continue wall following on the correct side.

Slicing Scan

The wall follower subscribes to the processed scan data, feeding the data into the wall follower function. The processed data is initially stripped to the relevant slices, notably the side and front slices of the racecar are saved, as shown in figure 1. These scan slices are used to inform the racecar's next drive commands.

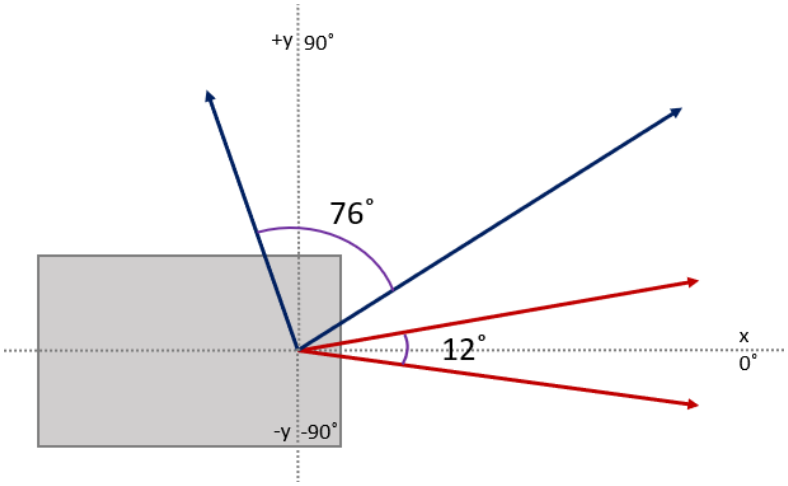

Figure 4: front and side slice visualization for a vehicle tracking the wall from the left side

The car scans the side starting at ± 30 degrees from the +x axis in the front, and looking back to ± 106 degrees, spanning a total area of 76 degrees of its side. It is important that the racecar looks slightly ahead to predict the wall ahead, while still looking behind to correct the robot if it is pointed away from the wall. In the front, the car scans from -6 to 6 degrees, obtaining a narrow but telling view of the car's future. The front slice remains narrow to avoid wall detection in the front when the car sees an object to it's side.

Finding minimum distance to wall

After cutting up the scan, the sliced ranges are filtered, and fed into a linear regression to output the minimum distance within the slice to the car.

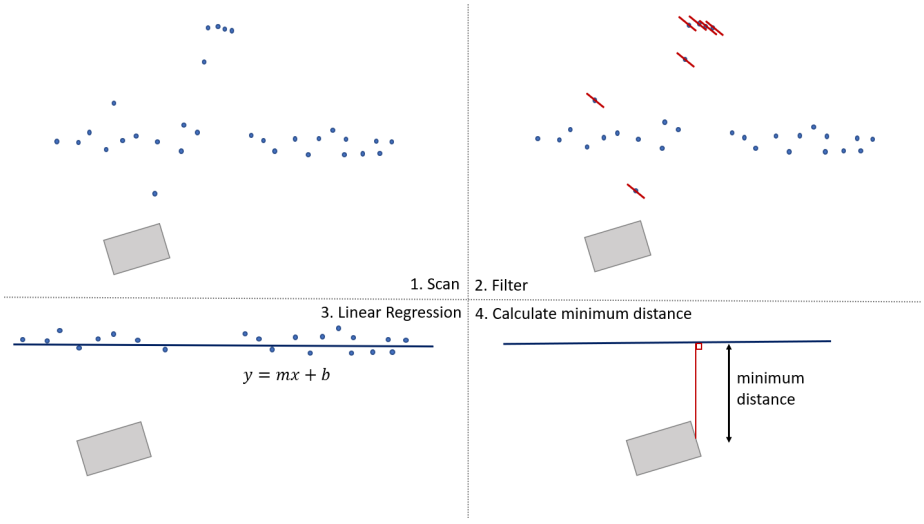

Figure 5: the steps of slice processing in order to measure the minimum distance to a wall in our wall follower implementation

Starting with the filtering (step 2), all ranges over 2 standard deviations away from the mean range are removed to remove noise, mostly encountered in the simulator. Also, any point that is over 3· desired_distance is removed. This allows the racecar to drive past doorways or small openings in the wall without turning into the room. If this filters out all of the data, then the racecar is quite far from a wall, so the distance filter allows data up to 10· desired_distance to pass through, so that the racecar can try to recover its path. The front wall does not get filtered in this way because we only car about the front wall if the distances are small.

Next, both the front and filtered side slices are converted from polar to Cartesian coordinates for linear regression (step 3). This further smooths out noise that may be present in the scan data. In an ideal world, the distance of the car to a wall would just be the minimum point of a scan, but noise in the data would cause significantly smaller distances to be reported if that method was used. Linear regression allows for a trend in the data points to be found for estimating the distance of the car to the wall. Using numpy module's polyfit, an equation is found for the front and side walls. These line equations are passed into the closest point on a line equation to get the minimum distance between the walls and the car (step 4).

Following the wall

If the front distance is below a threshold, then the car needs to begin its turn at max turn angle in the opposite direction of tracking to continue wall following on the correct side. This threshold is desired_distance + $0.3 \cdot$ speed, so that as the car's speed increases, it begins turning earlier to compensate. If the front of car is clear, then the car begins the PD control. The PD controller was found

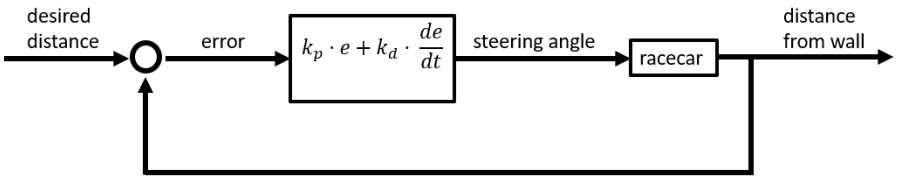

Figure 6: the basic PD Controller design found in our wall follower implementation

to be plenty sufficient for the control necessary to follow walls. In the above figure, the error is calculated as the side distance to the wall - desired_distance. This provides a negative error if the car is too close to the wall and positive if too far. This allows for the errors to map directly to the steering angle when multiplied by a factor of -1 for left side and 1 for right side tracking.

The derivative of the error in the PD controller is approximated using the error in the previous scan and the amount of nanoseconds that have elapsed since the previous scan. Accordingly, the derivative must be multiplied by a factor of 10^{-9} to return the error to helpful numbers. With the error and derivative, a k_p=0.5 and k_d=300 were settled upon. This lead to regular steering angles between -0.34 and 0.34, only exceeding these max turn limits in instances of extreme error. Otherwise these variables scaled the PD out nicely between these angles.

With a steering angle calculated, a AckermannDriveStamped type message is published to the .../nav_0 topic. This way any command published by the wall follower can be overwritten by the safety controller or the human operator holding the controller. When the car is placed in an area where it can see the desired wall, this implementation allows for effective tracking of the wall on the desired side at a set desired distance with at varying speeds.

3 Experimental Evaluation

3.1 Safety Controller Evaluation - Natalie Muradyan

We had two stages of testing: in simulation and real life. We also divided our tests into subcategories to make sure we covered the full range of possible situations our robot could be in. Here are the categories we used when testing.

- **No obstacles, going straight.**
  In this test, we aimed the robot parallel to the wall with no obstacles in front to check if the safety controller records any false obstacles and stops. In the simulation, we let the robot go over the whole room, and it did not stop or record any false obstacles. In real life, we let the car drive around a long hallway with no obstacles, and our controller again did not record any obstacles.

- **No obstacles, going at an angle towards the wall.**
  For this test, we aimed the car toward the wall at an angle to see if it would interpret it as an obstacle or know it could avoid crashing. When running our curved line method, our car successfully recognized that it would turn away from the wall, and it did not stop. In real life, we used the lid of the box as a wall to ensure our robot would not crash into a wall in case things went wrong, and we aimed the robot at an angle. Our robot was able to recognize that it could avoid hitting the wall by taking a turn instead of stopping, and the test was successful.

- **No obstacles, going away from the wall.**
  In this case, both in simulation and in real life, our robot was able to find the wall quickly and, instead of stopping, start following it.

- **Unavoidable obstacles.** For this test, we tried adding obstacles that the robot could not avoid hitting while it was running. In the simulation, we let the car go directly into a wall, and it would recognize that there's a wall and stop. In real life, we were able to test a wider range of things. We first added obstacles in front of the car, and it would stop. Then we removed the obstacles to see if it would recognize that and continue following the wall. Our robot passed these tests multiple times.

- **Avoidable obstacles.**
  This test case is very similar to the "going at an angle towards the wall" test case since they essentially test the same thing: whether the robot can recognize it can turn instead of stopping. The difference is that in real life, we did some more tests with adding objects spontaneously and seeing how the robot handles them. For example, we added a brick in front of the car at an avoidable distance, and it didn't stop but instead bypassed the brick. We tried doing this with multiple objects and distances, and all our tests were successful.

3.2 Wall Follower Evaluation - Andrew Manwaring

In a large room or simulation it is easy to see that the wall follower is maintaining the set distance from the wall, but when comparing between simulation and reality in a small room, the qualitative performance is hard to determine. Our team developed a few tests and metrics for evaluating performance based on the robot's perceived error. There are shortcomings with this approach, however it remains useful for comparison of simulation to reality.

Our main metric for comparing this error was designed based on the scoring for RSS Lab2. Our team uses a calculated error score out of 1 to gauge the performance of the wall follower, with a numbers closest to 1 the most accurate. At each call to wall follower's callback function, the error is added to a total error metric, and divided by the number of callbacks to obtain an e_{avg}. The following is the equation for error percentage:

$$e_{percentage} = \frac{1}{1 + e_{avg}^2}$$

To compare the simulation and reality, a test was run on similar areas in a room and simulation, allowing for side-by-side comparison of percentage error and actual error at turns. Additionally, the error was graphed concurrently, as these paths were taken. Below figures are the two paths taken by the simulated racecar and racecar along with their respective error graphs. Due to constraints of the physical space, the test was run at a speed of 0.6.

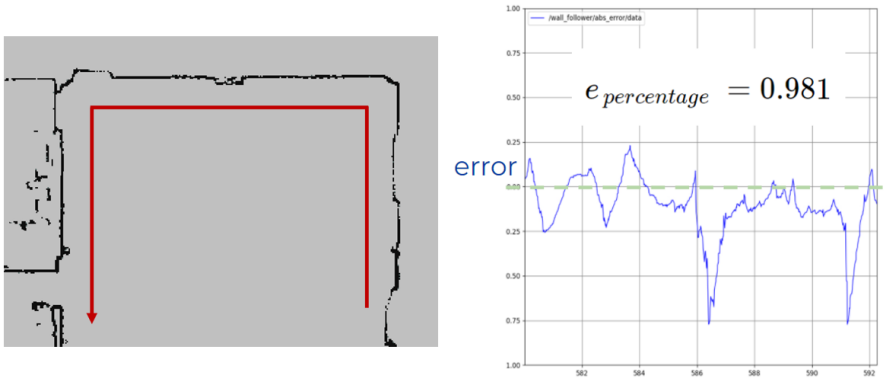

Figure 7: The path simulated path taken by the racecar, along with it's error along that path. The error percentage was of the simulation along this path was 0.981.

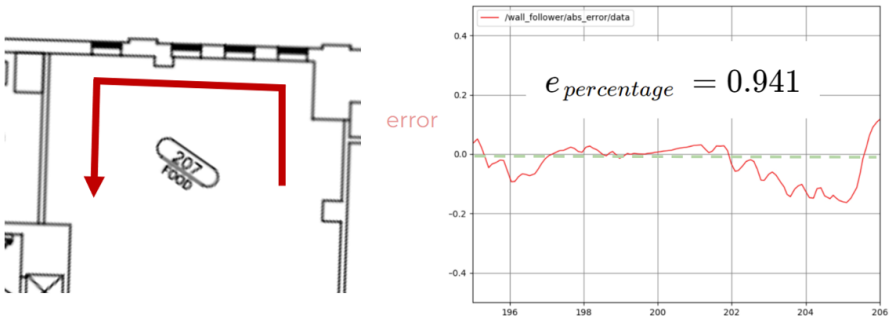

Figure 8: A floorplan of the real life path taken by the raceca, along with it's error along that path. The error percentage was of the racecar along this path was 0.941.

A few notes are apparent from comparison of the simulated and real racecar in figures 7 and 8. First, both the simulated and real racecar have respectably high error percentages. This demonstrates capability of our wall follower implementation both in simulation and reality. Additionally, the error on the physical car is much smoother than on the simulation, highlighting a major difference between the simulation and physical world - the wheels can not change direction instantly. This leads to slower reactions to corners and overall step response time. It follows that the performance of our racecar in reality is worse than the simulation due to these slower reaction times and differences in tuning between the physical world and simulation.

Wall follower was evaluated in a number of other qualitative ways as well. For instance, we ran multiple tests on the car at varying speeds. We found as the speed increased as high as 3, the car had much greater error around corners and occasionally lost the wall for some time. In the future we would like to mitigate this by tuning the PD variables and angle slices for higher speeds. The racecar was tested repeatedly to reach the working form currently, and in these tests we honed in on the significant disadvantage of the Velodyne LiDAR's minimum distance. With a minimum distance of 0.5m, the team had to set our desired distance to 1m or greater to allow for the safety controller to run without interrupting the wall follower. This meant that the car needed to be tested in large spaces. The team also hopes to improve the car's ability to navigate through tighter spaces in future evolutions of the racecar.

The wall follower performed well in both simulation and reality. The simulation's error percentage was notably better than the physical racecar, but this

tracks with the physical racecar's time to turn wheels and change speed.

4 Conclusion - Tiffany Horter

Through this lab, we were able to drive the physical car through tele-op, implement our wall follower code from simulation onto the real robot, and create a safety controller which stopped the robot when it was at risk of crashing. We achieved a high accuracy for the wall follower in simulation of 98.1% and 94.1% on robot. For our safety controller, it successfully prevented crashes in all tested situations. Both the wall follower and the safety controller were highly effective.

There are still several improvements that could be made to improve the performance of this phase, such as tuning. To improve wall follower accuracy, continued tuning of the parameters would help eliminate the "wiggling" of the robot as it tries to maintain a constant distance from the wall. One hardware related limitation that likely cannot be fixed for the wall follower is the range is limited at close range to seeing objects that are greater than 45 cm away. To allow the wall-follower to work at distances closer than 0.5 m, we would probably have to change the LiDAR. In the next design phase, we will be able to use computer vision instead to detect closer objects and follow a line.

5 Lessons Learned

5.1 Andrew Manwaring

Throughout this lab I learned the importance of starting early and testing often. Our team started our work on the code early and continued to refine it, however, we did not begin robot testing until we felt that it was perfectly running in simulations. If we had tested the two concurrently, we would've discovered bugs early and they would've been much easier to troubleshoot rather than switching back between the model and script. This is also a lesson in team work as it would have allowed more members to work on a challenge at the same time.

Additionally, I learned more about the ROS debugging system and about how to spot bugs more quickly. Processes like rospy.loginfo and quick publishers did the trick of finding those hidden errors. The only applicable prior knowledge I have had from before this class was Python and control feedback loops. I have learned so much about Linux, github, and ROS, so with each lab I am gaining a lot of proficiency in these topics.

In creating our brief, I saw how much our performance improved as we ran through the slides more and more. We honed in on what we wanted to say, eliminating time wasting fluff.

5.2 Nisarg Dharia

This lab helped me refine numerous technical skills necessary for a wide range of application in robots. For one, my familiarity with frameworks such as ROS and packages such as Numpy improved significantly, as I was able to research and incorporate a number of features into my code. Among these were the use of message filters in ROS to subscribe to multiple topics at once, as well as the use of masks in numpy to filter arrays quickly and efficiently. In addition, I also learned a lot about various pieces of hardware and some of their limitations. In particular, I was educated on how our Velodyne LiDAR worked by sending out lasers and measuring their return time to calculate distance, and how it's rotation rate being slower than the /scan topic's publishing rate led to incomplete data.

I was also able to identify some areas of improvement as a teammate. Specifically, our initial delegation of tasks was done hastily and without foresight, and led to some dissatisfaction among teammates later in the lab. In the future, I hope to be more involving of teammates and forward thinking when making these decisions to avoid conflicts and keep the team satisfied.

Most importantly though, on the intangible side of things, this lab reminded me how important it was to be persistent when facing large and complicated problems. There were numerous times where something broke or went wrong, and I often felt compelled to give up and make excuses. However, thanks to my teammates and some looming deadlines, I was able to persevere and was rewarded with the wave of satisfaction when everything finally worked.

5.3 Natalie Muradyan

After this lab, I refined some of my technical skills. I learned many things that were very specific to the lab, such as how to debug the robot when it doesn't connect, but also some things that are helpful in other areas, like different approaches to detecting obstacles, designing a website, etc.

I also developed some soft skills and some non-technical skills. I refined my skills in designing presentation diagrams, graphical representation, and making more appealing slides.

Overall, this lab was very creative and helped me think about how autonomous machines work and what goes into the process. It is very interesting to see the whole process of designing a car, and of course, things start from testing and safety.

5.4 Tiffany Horter

From a technical point of view, this lab taught me a great deal about ROS and reinforced my understanding of Numpy through all of the matrix manipulation required to interpret the data. I also learned about more about how to implement control theory and utilize LiDAR technology. Beyond that, I became familiar with the differences between simulation and reality. One major lesson that I learned was the importance of testing on the robot early in the process – many of our issues came from only implementing for simulation in the beginning and then much later discovering that the robot interpreted its input differently or there were unaccounted-for errors on the physical robot.

In working on the communication aspects of this class, including the briefings, I have been working on brevity. This has been a constant challenge for me, and having a hard 8 minute cut-off provided excellent motivation for keeping the briefing brief. In addition, I worked on integrating the new method of designing slides mentioned in class into my typical workflow. While at first it was challenging, especially adding full sentences to the top of slides, I believe that now my slides are more understandable to my audience. Breaking up the project into smaller chunks would have helped to collaborate more easily and prevent the silo-ing of knowledge between areas of the project.

6 CITATIONS

VLP 16 User Manual (left side of Figure 1 found on pg 30)