Lab #3 Report: Wall Follower

Team #11

Quincy Johnson
Aileen Ma
Dylan Goff
Matthew Jens
Anika Cheerla
6.141/16.405

March 5, 2022

1 Introduction (Quincy)

Our team's objective for lab 3 is to design and integrate a robust autonomous wall following algorithm with a physical robot such that the robot is able to follow a wall from a desired distance away, in addition to implementing a safety controller to override any autonomous action before a collision can occur. While accomplishing these tasks, several constraints had to be taken into account: the robot's LIDAR (Light Detection and Ranging) sensor has a limited scan range from angles $-\frac{2\pi}{3}$ to $\frac{2\pi}{3}$ radians (240^o), the max/min steering angle of the car is ± 0.34 radians, and the max speed is 4 m/s. Along with these constraints, it is also important to take into account noise from the input, along with the unpredictability of the environment being navigated. With these considerations and the end goal in mind, the challenges of the lab can be defined in three stages. First, the robots needs to detect a wall by slicing up the LIDAR scan data to include only the part of the scan corresponding to which wall side the robot is following and then filtering this data to remove any outliers interfering with wall detection. Second, the robot car needs to accurately follow corners and noisy walls via modulating the steering angle of the robot car through the use of a PD controller. Lastly, the robot car needs to be equipped with a safety controller to protect the robot from potentially damaging collisions with objects and people, while also not being so invasive that the functionality of the wall follower is impaired. This lab not only tests our ability to design and implement a robust controlled autonomous system, but also our ability to effectively collaborate as a team to implement, test, and document our final design. Ultimately, the work that we accomplished in this lab will serve as the foundation for future labs.

2 Technical Approach (Anika, Aileen, Dylan)

2.1 Wall Follower (Anika and Aileen)

In this section, we detail how we made an autonomous controller that drives the racecar forwards while maintaining a constant distance from a wall on either the left or right.

2.1.1 Finding a wall

The first step in implementing the wall follower was developing an accurate wall detector. At each timestep, the LIDAR takes 1000 measurements spanning across a 270° field of view, with only the 90° region directly behind the robot being unaccounted for. The LIDAR packages and sends the data as an array, which we then need to process for further use.

First, we slice the data based on the desired wall side for the car to follow. For example, when the wall is to the left of the car, the robot will only consider LIDAR points from the front of the car to the leftmost point.

After checking to make sure that a range of points is being returned, we needed to process the data. First, we were concerned about particulates on the LIDAR itself that would obscure the data. These data points would cause the LIDAR to pick up distances that were abnormally close, skewing our line of best fit and misrepresenting the wall we intended to detect. To remove these points, we decided to apply a mask that would check to see if the distance was less than 0.1m, and if so, to assign a weight of 0 to that point. In doing so, we could effectively remove any outlier caused by particulates or other insignificant factors we would not want to use in calculating a line of best fit. For all other points, we assigned a weight based on the inverse square distance from that point to the car. This would assign a greater priority to points closer to the car than points farther away. This type of weighting was particularly useful in following walls disappearing around the corner (an inner turn), especially in narrower hallways.

After applying a mask and assigning weights, we used the numpy polyfit feature to calculate a line of best fit belonging to all these points. This performs a least squares regression on the points, generating a line representing the wall our robot intends to follow. This method forces the car to follow the wall that is closest to it if there are multiple walls in its field of vision and also removes distant outliers from the data.

2.1.2 PD Controller

Our robot makes use of a PD (proportional-derivative) controller in order to follow the wall autonomously and respond to changes in the environment. At every time step, the robot receives the error term $e(t)$, representing the error difference between the robot's actual distance from the wall and the robot's desired distance from the wall at a given time t. We then formulate a new steering angle input $u(t)$ to be passed in to our robot following the formula

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

The term $K_p e(t)$ represents the proportional term, which directly modulates the future steering

angle based on the current error. This is the dominating term affecting the motion of the robot car itself; however, it cannot be used alone because it is not sufficiently responsive to changes and will produce drastic overcorrections. The coefficient K_p represents the proportional gain, and its value is determined from experimentation.

The term $K_p \frac{de}{dt}$ represents the derivative term, which estimates the future error and dampens the effects of the proportional term. In order to calculate $\frac{de}{dt}$, our robot subtracts the previous error term $e(t-1)$ from the current error term $e(t)$. While it may seem oversimplified that our derivative term only consider the prior timestep, it proves surprisingly sufficient in the design of a robust PD controller. The coefficient K_d represents the derivative gain, and its value is similarly determined from experimentation.

It was necessary to determine the values of K_p and K_d through experimentation. First, we start with K_d set to 0, and slowly increase K_p by increments of 0.3 until we start seeing oscillations in the motion of the robot. This is most prominent when the robot encounters an obstacle which requires a sudden change in trajectory, such as turning a corner. At this point, we increase K_d until the overcorrections are dampened and oscillations disappear. We assumed that K_p would be greater than K_d because the proportional term should dominate the derivative term, and both terms would likely fall between 0 and 5.

2.1.3 Front wall logic

When there is a wall immediately in front of the robot, we chose to have the robot steer in the opposite direction of the wall it is following to avoid a collision and continue following the wall. The robot first detects if the wall is within a buffer away from it, where the buffer is determined by the speed of the car. If it is, the system sends the max steering angle in the direction opposite to the wall the car is following as the next steering command. However, note that this logic does not work if the wall is at an angle steeper than the max steering angle. In this case, our safety controller would kick in to avoid a collision.

2.2 Safety Controller (Dylan)

The hardware used for the racecar is expensive; a high-speed collision could result in hundreds or even thousands of dollars worth of damage to the robot and must be avoided. These collisions can be caused by controller failure, resulting in a collision with a wall, obstacle, or person. Thus, it is important to implement a controller that can sense when collisions are imminent and halt the car before disaster occurs.

2.2.1 Mathematical Approach

By predicting the position of the car a short amount of time into the future, we can determine whether or not a crash is imminent. To do this, we take the car's current velocity command V and predict where the car will move after a short amount of time dt, and thus predict the change in LIDAR distance readings $\Delta D(\theta)$ as shown in Figure 1.

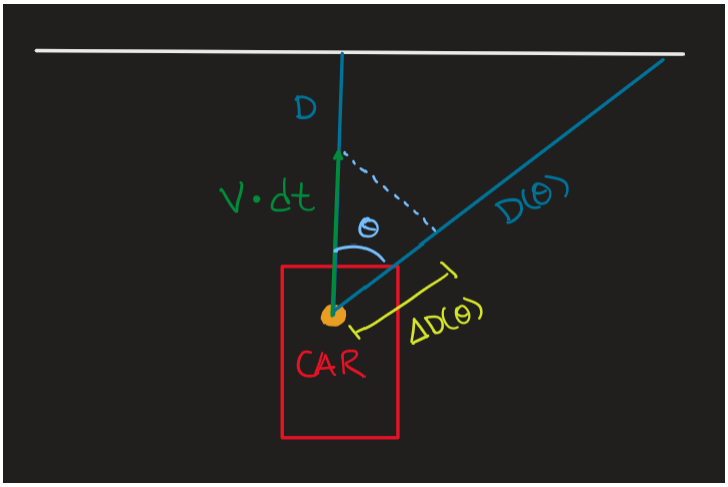

Figure 1: Geometric representation of predicted change in future LIDAR distance readings

Following basic geometry from Figure 1, we can calculate the change in distance after time dt for each corresponding LIDAR angle θ as

$$\Delta D(\theta) = V * dt * cos(\theta) \tag{2}$$

where $D(\theta)$ represents the current LIDAR distance reading for each angle and $\Delta D(\theta)$ represents the projected change in $D(\theta)$ over dt seconds.

However, this is not sufficient to detect collisions, as the LIDAR measurements are not taken from the edges of the car. Thus, we must add a correction that takes into account the distance from the LIDAR sensor to an imaginary bounding box around the car that we define to be the edge of the car. The geometry for this bounding box is shown in Figure 2. In this figure, L denotes the distance from the LIDAR to the front of the bounding box, W denotes the distance from the LIDAR to the left/right sides of the bounding box, and E denotes the distance from the LIDAR sensor to the bounding box as a function of θ.

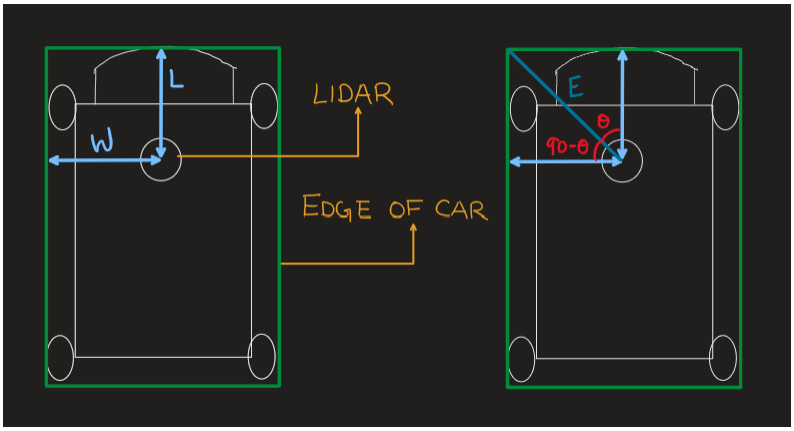

Figure 2: Bounding box for the car.

From Figure 2 we can infer functions for E from simple trigonometry. Note that, due to a geometric issue with the formation of right angles between the center of the LIDAR and the bounding box, E is a piecewise function as shown in Equation 3.

$$E(\theta) = \begin{cases} \frac{L}{cos(\theta)} & 0 \leq \theta \leq \frac{\pi}{2} \\ \frac{W}{cos(90-\theta)} & \frac{\pi}{4} < \theta < \frac{\pi}{2} \end{cases} \tag{3}$$

The predicted distance from the car bounding box to any obstacle after time dt, then, becomes

$$D(\theta)_{t+dt} = D(\theta)_t - \Delta D(\theta) - E(\theta) \tag{4}$$

Lastly, we must account for the direction the robot is moving in. To do this, we check the difference in values obtained from the LIDAR at the previous time step $D(\theta)_{t-dt}$ and the current LIDAR readings $D(\theta)_t$. This gives us $\frac{dD(\theta)}{dt}$, which denotes how much quickly we are moving toward an object located at angle θ from the car. We assume that the LIDAR data will have some level of noise B associated with it, making it such that $\frac{dD(\theta)}{dt} \neq 0$ even when not moving. Thus, we make the assumption that the racecar is moving towards an object when $\frac{dD(\theta)}{dt} > B$. For all $D(\theta)$ that satisfy this condition, we apply the control scheme as shown in Equation 4. Then, when $D(\theta)_{t+dt} < 0$, the safety controller detects an imminent crash and subsequently intervenes by outputting a command to stop the robot.

Note that the robot will not always be moving in a straight line (the wheels can turn), and thus we are approximating the robot's trajectory with a linearization over a small time step dt. Since the time step is small, though, we neglect the error incurred by the linearization.

2.2.2 Hardware Implementation

When the safety controller outputs a command, it is necessary for it to override any command being sent by a navigation controller such as the wall follower. The racecar has topics for commands from the manual controller, safety controller, and navigation controller as shown in Figure 3.

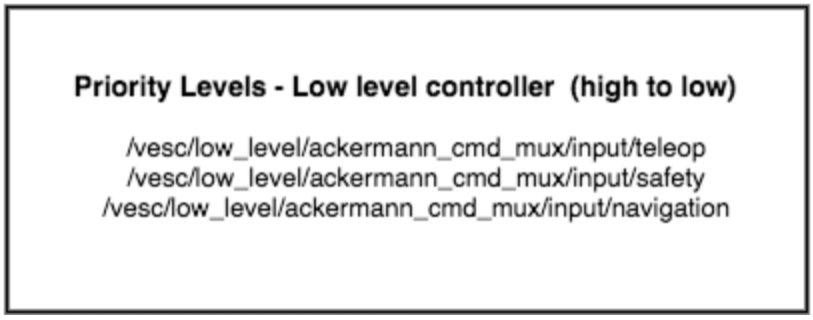

Figure 3: Command priority in descending order. Topics in order from top to bottom: manual commands, safety controller, navigation controller

From Figure 3, we can see that the safety controller (which publishes to the topic ending in .../safety) is higher in priority than the commands published by the navigation controller (topic ending in .../navigation). However, manual commands override all controllers since manual commands are published to the topic ending in .../teleop. The controller itself has several checks in order to prevent collisions:

if *Too close to object* **then**
  Stop;
else
  Pick points car is moving towards;
  Check if car will crash into objects it is moving towards;
  Stop if crash is imminent
end

Algorithm 1: Safety controller logic

The former check is chosen based on a small distance to an object that is assumed to be unacceptable regardless of speed. For example, this will come into effect if the car starts too close to an obstacle. The latter check is more robust, detecting any approaching obstacles and halting the car as needed.

2.3 Determining Outliers

Similar to the issue we encountered in wall following, the issue of LIDAR obfuscation can cause major issues with collision detection and consequently affect the safety controller. The LIDAR will assume there is an object directly in the path of collision and cause the safety controller to take over. As a result, we filter out any values that are within an experimentally determined distance (in this case, we choose 0.1m like we had previously done with the wall follower).

3 Experimental Evaluation (Matt and Anika)

3.1 Wall Follower

3.1.1 Quantitative Evaluation

In order to measure both the robustness and accuracy of the wall follower, we tested it on three course cases detailed with the desired car path as shown in Figure 4. These tests were conducted with both the wall follower and safety controller running.

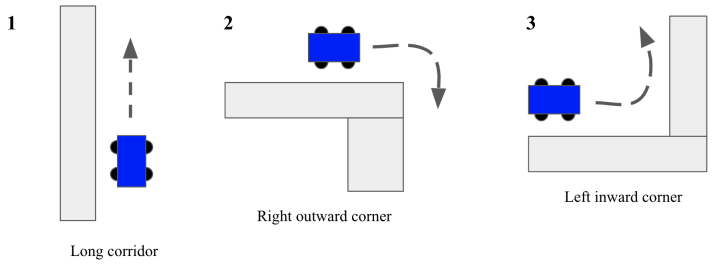

Figure 4: The three different course configurations we tested our wall follower code on.

We tested each case at 3 different speeds: 0.5 m/s, 1.0 m/s and 1.5 m/s, with a desired distance from the wall of 0.5m. Figure 5 details all of the results collected.

Speed/Case	1	2	3
0.5 m/s	0.0318 ± 0.0217 m/s	0.0631± 0.0633 m/s	0.0929 ± 0.0100 m/s
1.0 m/s	0.0797 ± 0.0630 m/s	0.1363 ± 0.1070 m/s	0.0900 ± 0.0263 m/s
1.5 m/s	0.0626 ± 0.0179 m/s	0.3407 ± 0.2257 m/s	0.1306 ± 0.0229 m/s

Figure 5: Mean and standard deviation of the error of each course and speed configuration.

In order to evaluate the performance of each of the runs, we calculated the mean and standard deviation of the distance of the car from the desired distance of the wall at each time step. These results are shown in Figure 5. The mean error mostly increases with speed along each of the cases. This makes sense as at higher speeds, the PD controller is not as able to correct for small errors. We also noticed that the car was slightly wobbling at higher speeds, which does account for the increase in error and could be fixed with more gains tuning. Course configuration 2 seems to be harder than the others, especially at higher speeds. Quantitatively, we noticed that the robot takes a much wider turn when turning an outward corner than when turning an inward corner. This could be due to either the front wall checking code from Section 2.1.3 kicking in during an inward

corner causing a closer turn. It could also be due to inaccuracies in the intermediate lines generated by the linear regression code from 2.1.1, as both walls of the robot may be detected by the system.

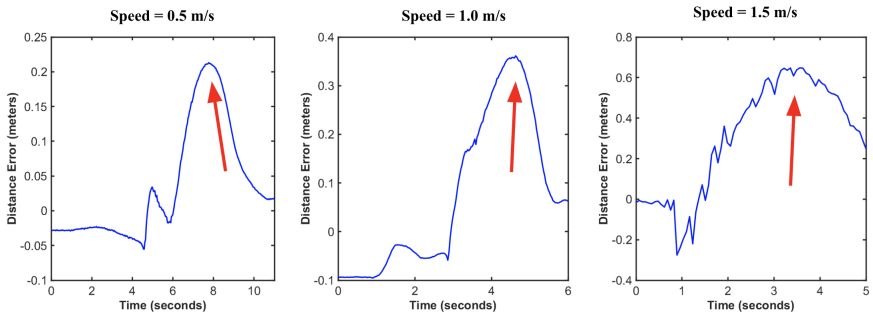

Figure 6: Error from the desired distance over time during the Course Configuration 2, the right outward corner.

In Figure 7, we show the error over time at various speeds for the second course configuration. The red arrows indicate the peak of the turn the robot had to take to maneuver around the corner. As the speeds get higher, this peak has a higher distance from the desired line as the car needs more space to execute a feasible turn.

3.2 Safety Controller

3.2.1 Quantitative Evaluation

To evaluate the safety controller, we determined that we must perform two types of tests. The fist test entails the safety controller being triggered from a still object that is in the robot's path. The second test requires the robot to stop when an object is moving in front of the robot's path. Another important element of evaluating the safety controller is testing the robot at varying speeds.

Each test also requires the correct recordings so the right analyses can be performed. The recordings include a video of the robot, a rviz recording, and the distance from the object to the robot at the end of the test.

We started with the still object tests and ran two trials of 0.5 m/s and 1 m/s, with a desired distance from the wall of 0.5 m. After completing those tests, we then tested the safety controller with a moving object (a moving person). Figure 7 displays the results.

Speed/Case	Still Object	Moving Object
0.5 m/s	0.0762 ± 0.001 m	0.1016 ± 0.001 m
1.0 m/s	0.0381 ± 0.001 m	0.0254 ± 0.001 m

Figure 7: Results from testing the safety controller with still and moving objects at varying speeds. The measurements in the table are the distances between the object and the robot's bumper at the end of the test.

Evidently, the safety controller performed the best at the lowest speed of 0.5 m/s due to the substantial buffer between the object and robot, but it still worked at 1 m/s given that our safety controller accounts for the robot's current speed. At even higher speeds, however, the safety controller stopped but softly bumped into the object. This error could be attributed to the safety controller not being sensitive enough to the car's inertia. Higher speeds require more time for the robot to slow down. Our team is actively working towards improving the controller so that it can work at any desired speed.

3.3 Qualitative Evaluation

We qualitatively demonstrated the performance of the robot on a large obstacle course complete with inward and outward turns and obstacles up against the walls, as shown in the videos here: https://tinyurl.com/team11-vids. We wanted to show that the robot could execute a series of difficult courses and that the safety controller would override the wall follower code when an unexpected obstacle came up.

4 Conclusion (Matt and Aileen)

We achieved this lab's primary objective by implementing an accurate wall-following algorithm on a physical robot and producing a working safety controller. There were three key milestones that our team reached this week in order to reach this goal.

First, we were able to successfully integrate various members' wall following algorithms to optimize for the most efficient implementation. We overcame the difficulties of dealing with faulty LIDAR scans, hardware freezes, and inconsistent testing to produce a robust algorithm in the end. Second, we were able to collaborate to build the safety controller. Making use of forward propagation to stop when a future collision is expected, our robot will carry forth our algorithm to prevent damaging crashes. Third, our team made strides in our ability to communicate effectively. We set up a Facebook messenger group chat, a shared Google drive and calendar, and a team Github repository. With these tools, we were able to facilitate an environment where we could collaborate and delegate tasks. We also compromised with one another, such as by staying late before or after class in order to finish some component of the robot.

Given the tight time constraint for the lab, however, there remain a couple of areas upon which we would like to improve. Our robot was very successful at lower speeds, but at higher speeds, both

the wall follower and the safety controller struggled to make tight turns or to stop before hitting an object. To address these issues, we are planning to continue tinkering with the parameters of our existing program while also thinking of new ways to improve performance. These improvements will ensure that our robot is ready for next week's exciting challenges of utilizing computer vision techniques to detect and recognize objects. Building upon our team chemistry, we are looking forward to settling into roles and routines that allow us to grow while also providing stability in our schedules.

5 Lessons Learned

As is common for a team transitioning to working with one another, we spent the first week settling into roles and routines. Not only did we discover the ways in which our technical skills complement one another, but we also realized that gaps in our schedule that often made it difficult to meet as a team. It was especially crucial to learn that we should start working on the lab even earlier as a team in the future. Because collaboration is an important part of the process, it is necessary to build in buffer time for everyone to meet without pressure (or neglecting other aspects of their life).

Moreover, we also learned that it is important to approach labs with a plan and work through it systematically. Although we broke into teams to attack the issues, we also should have planned out tests and benchmarks to indicate when we had successfully resolved a problem. Furthermore, we realized that our testing was not robust enough in earlier stages of the lab, which forced us to start over many times. For example, we should have tested our initial K_p and K_d values with incrementally increasing velocities while keeping other factors the same. Because we did not do so, we ended up needing to re-tune the parameters multiple times when we encountered issues later on in the lab with unexpected robot behavior.

Finally, we learned that we need to check in frequently in order to ensure that members are on the same page and not getting blocked. By sharing knowledge, we can overcome hurdles much more readily or learn when it is time to ask a TA for help.

5.1 Aileen

Aileen faced some of her fears head on with regards to not only learning a new tech stack, but also working with a team without feeling intimidated. In particular, she realized the value of approaching labs with a plan and timeline in mind.

5.2 Matt

Matt found that the integration of hardware and software was particularly challenging. When there is an issue to be resolved, the interface of software and hardware causes the number of sources of error to become much larger. He also discovered the importance of communicating well with a team and managing expectations ahead of time is crucial to efficient production.

5.3 Anika

Anika discovered that it is necessary to allocate more time than expected when moving from simulation to hardware. Although integration initially seemed like a seamless transition, it quickly became clear that there were multiple differences that snowballed into larger issues as time went on. She also learned about the difficulty in coordinating teammates' schedules and the importance of establishing clear meeting times early on.

5.4 Quincy

Quincy learned about the importance of complementing others' skills and being able to ask for help. Additionally, he realized that it is important to have a centralized location for goals and deadlines. In the coming weeks, he is looking forward to better coordination with teammates on meeting times and juggling commitments.

5.5 Dylan

Dylan found that considering the edge cases to algorithm development is crucial to reducing future errors. Additionally, he realized that the transition from simulation to hardware can produce unexpected results, and it is important to gain insights from both team members and staff members when such errors occur. From a CI component, he learned that various people have different preferences when it comes to scheduling and it is important to establish clear meeting times and boundaries such that people feel comfortable working together.