Lab 3 Report: Wall-Following on the Racecar

Team 8

Monica Chan
Cristine Chen
Kyle Fu
Asa Paparo

RSS

March 15, 2025

1 Introduction

Author: Kyle Fu
Editor: Monica Chan

This lab focused on designing two key components of an autonomous robotic system: a safety controller and a wall follower. The safety controller ensures safe operation by preventing collisions with walls and pedestrians, allowing us to test our robot in autonomous mode without risk. The wall follower enables the robot to navigate autonomously along the walls of structures like the Stata basement. Figure 1 shows the hardware and software components involved.

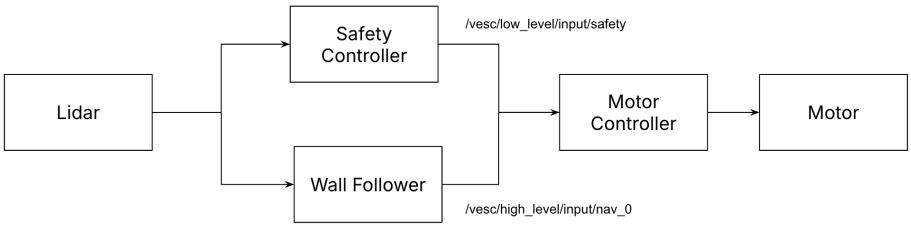

Figure 1: High level overview of the lab. The hardware components include the LiDAR, motor controller, and motor. Safety controller and wall follower software modules publish drive commands to the motor controller.

The primary goal of this lab was to develop and tune PID controllers while ensuring robust and modular design principles for safe and reliable autonomous navigation. The safety controller was required to detect imminent collisions and stop the robot in time, using a kinematic-based approach that calculates stopping distance based on the robot's velocity and obstacles ahead. To ensure reusability across future labs, we implemented the safety controller as a standalone ROS package.

For the wall follower, the robot needed to follow walls while going straight, making turns, and maintaining a target distance from either the left or right wall. Our approach involved extracting wall points from LiDAR data, fitting a line to those points, and using a PD and P controller to track the distance and relative angle to the line. This provides a balance between responsiveness and stability in navigation.

This lab reinforced key concepts in control design and safety-critical autonomy, contributing to the broader development of reliable robotic systems in RSS.

2 Technical Approach

Author: Monica Chan
Editor: Kyle Fu

2.1 Safety Controller

Our safety controller is designed to prevent collisions with objects, walls, and people using a time-based kinematic approach. Figure 2 provides an overview of how the controller ensures safe stopping when an obstacle is detected within a critical range.

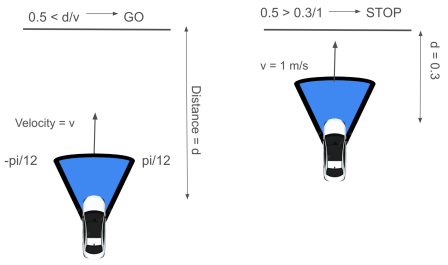

Figure 2: Explanation of safety controller function. The right image shows a situation where the safety controller would stop the robot, since the time to collision is under 0.5 seconds.

The system enforces a minimum stopping time of 0.5 seconds, meaning if the ratio of the current distance to an obstacle and the robot's commanded velocity falls below 0.5 seconds, the robot will command its velocity to zero. Equation 1 formalizes this condition:

$$t_{stop} = \frac{d}{v} < 0.5sec \tag{1}$$

where d is the measured distance to the nearest object and v is the robot's current velocity.

To prevent unnecessary stops while maintaining safety, we evaluate obstacles within an angular range of $-\pi/12$ to $\pi/12$ in front of the robot. This range was determined through empirical tuning, particularly by observing the controller's performance in the wall-following task. We found expanding this range made the controller overly conservative, often causing the robot to halt unnecessarily in tight spaces or sharp turns, as demonstrated in Figure 3.

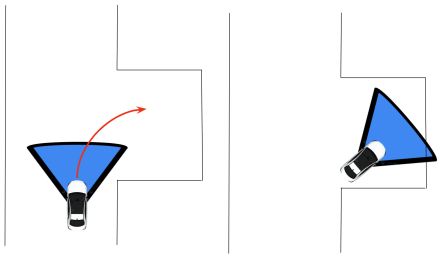

Figure 3: Illustration of robot failing a turn when the considered range of points is too large, causing the robot to be overconservative in stopping.

In addition to collision avoidance, the safety controller includes a watchdog timer to handle sensor failures. This watchdog monitors incoming LiDAR scan messages and ensures the robot stops if no data is received. This prevents undefined behavior that could arise from hardware malfunctions or communication failures. We structured the safety controller as a separate ROS package, ensuring modularity and reusability for future autonomous navigation tasks.

2.2 Wall Follower

Our wall follower maintains a desired speed v and specified distance $D_{desired}$ from either the left or right wall while going straight and making left or right turns. Figure 4 provides an architectural overview, illustrating how the robot adjusts its trajectory based on LiDAR data. We break the process into 3 parts: filtering the lidar points, the distance controller, and the angle controller.

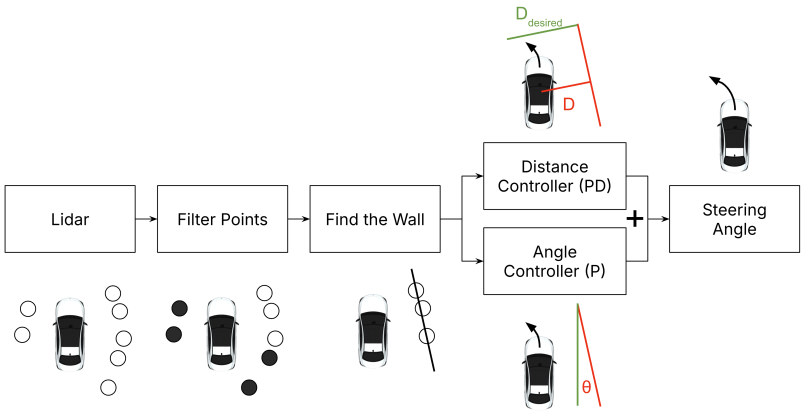

Figure 4: Overview of our wall following process. We begin by filtering the points and fitting a line to the points that make up the wall. Then we implement a distance and angle controller to calculate a new steering command.

2.2.1 Point Filtering

In the same way the safety controller only considers a portion of the total points, not all points returned by the LiDAR are necessary for effective wall-following. The primary goal of our point selection system is to retain only the points that define the wall while filtering out irrelevant measurements. Figure 5 illustrates our point filtering algorithm.

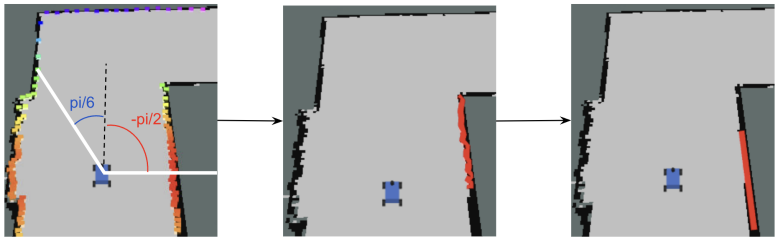

Figure 5: Process of filtering irrelevant points and finding the wall. We first remove all points outside of our angular range (white lines). Then we filter out points with a large gap, such as on the turn, which removes the points on the front wall. We finally fit a line to estimate the wall.

We begin by filtering points based on their angular range relative to the front of the robot. The chosen range depends on whether the robot follows the right or left wall:

- Left wall: Consider points within $[-\pi/6, \pi/2]$
- Right wall: consider points within $[-\pi/2, \pi/6]$

These ranges were empirically tuned using both our simulation environment and the physical robot to ensure robust wall and corner detection.

After selecting points within the desired angular range, we filter out points that are not part of the wall by removing large gaps in the detected wall structure. This is implemented using Algorithm 1, which iterates through the points and filters out any that are too far apart from the previous point (either exceeding a predefined max gap, or being disproportionally far apart given its distance from the robot). These checks ensure gaps solely due to the LiDAR's scanning resolution are not filtered, but big gaps in the wall are.

```
pts ← get_scan_range()
final_pts ← []
for each pt in pts do
    last_pt ← final_pts[−1]
    if last_pt is not None then
        gap ← euclidean_distance(pt, last_pt)
        robot_dist ← euclidean_distance(last_pt)
        if gap > max_gap or (robot_dist > desired distance and gap >
          robot_dist/η) then
            break
        end
    end
    final_pts.append(pt)
end
```

Algorithm 1: Point Filtering for Wall Detection. $max\_gap = 0.6$, $\eta = 2.5$ are empirically tuned scalars.

Once we have filtered the relevant points, we fit a line to them using NumPy's polynomial fit function. This line serves as the reference wall, allowing our controllers to compute an appropriate steering command.

2.2.2 Distance Controller

We implemented a Proportional-Derivative (PD) controller to maintain a desired distance from the wall, similar to our approach in Lab 2 (Wall Following in Simulation). Using the line fitted to the wall, we compute an error function, as defined in Equation 2.

$$e(t) = D_{desired} - D_{wall}(t) \tag{2}$$

$$u(t) = K_p e(t) + K_d \frac{d}{dt} e(t) \tag{3}$$

where $e(t)$ is the error function and $u(t)$ is the distance controller's steering command. The steering output of the PD controller, given by Equation 3, ensures the robot remains at the target distance from the wall.

We opted for a PD controller instead of a full PID controller because our testing—both in simulation and on the physical robot—showed minimal steady-state error, rendering the integral term unnecessary. We tuned the distance controller empirically, increasing K_p until the system oscillated slightly, then increasing K_d to prevent oscillations. Our physical robot used $K_p = 0.75$ and $K_d = 0.1$. We note that the ideal PD values were higher in simulation due to different car dynamics.

2.2.3 Angle Controller

During simulation testing, we observed a distance-only controller performed well on inward turns, but struggled with outward turns.

- Inward turn: A turn in the same direction as the followed wall (e.g. a right turn while following the right wall).
- Outward turn: A turn opposite to the wall's direction (e.g. a left turn while following the right wall).

The distance controller failed to anticipate outward turns, reacting too late and causing the robot to overshoot the turn, as seen in Figure 6.

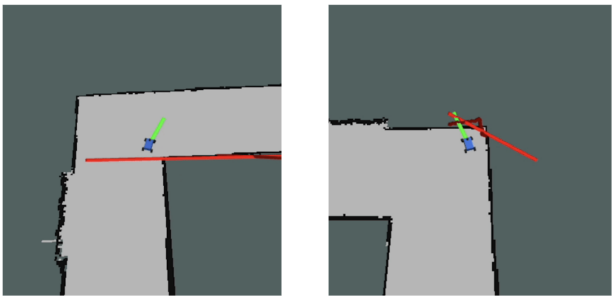

Figure 6: Motivation for the angle controller. This robot simulation runs only a distance controller, causing our robot to turn late on outward turns and collide with the front wall.

To handle outward turns, we introduced a secondary angle controller that encourages the robot to maintain a parallel orientation to the wall. With this

controller, when approaching a corner, the robot begins its turn earlier to maintain the correct heading. After experimenting with control strategies, we found a Proportional (P) controller produced the most stable performance.

The angle controller's output, given by Equation 5, is combined with the distance controller's output to generate the final steering command, as shown in Equation 6:

$$e_\theta(t) = -\theta_{wall}(t) \tag{4}$$

$$u_\theta(t) = A_p e_\theta(t) \tag{5}$$

$$\theta_{commanded}(t) = u(t) + u_\theta(t) \tag{6}$$

where θ_{wall} is the angle between the robot and the wall, A_p is the proportional coefficient for the angle controller, and $\theta_{commanded}$ is the commanded steering angle of the robot.

We tuned the angle controller after tuning the distance controller, increasing A_p until the robot handled both inward and outward turns well. Our angle controller on the physical robot used $A_p = 0.2$.

3 Experimental Evaluation

Author: Cristine Chen
Editor: Asa Paparo

3.1 Safety Controller

We tested our safety controller first by positioning the robot on a brick and running the wall follower code. Next, we placed the bin lid in front of the robot's line of sight to assess its response. Figure 7 illustrates our initial process for evaluating the safety controller. When we start the wall follower code, the robot's wheels spin rapidly, as evidenced by the slight motion blur in the wheels. However, as the bin is brought closer to the robot's line of sight, the safety controller activates. The robot detects the approaching obstacle and responds by locking its wheels in place.

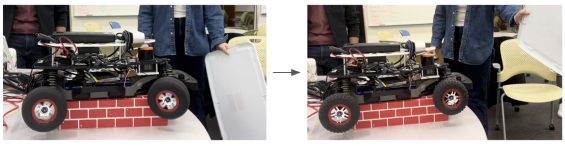

Figure 7: Set-up for the safety controller test on the brick.

After confirming that our safety controller worked as expected in the brick test, we moved to the hallways of Stata basement to conduct further trials. We ran the wall follower code on the robot and quickly used the bin lid to block its path to evaluate its response to unexpected obstacles. Figure 8 illustrates how the robot successfully detected the obstruction and came to a stop without making contact.

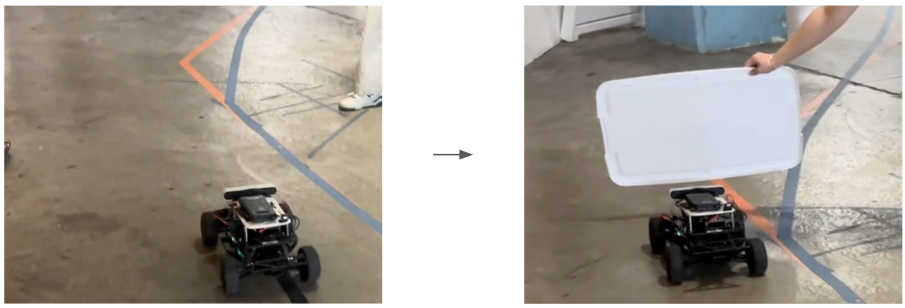

Figure 8: Set-up for the safety controller test in action.

We assessed our time-based safety controller's effectiveness across different velocities through tests at $v = 1$ m/s and $v = 2$ m/s. The setup of our experiments is shown in Figure 9. To ensure consistency, we marked a designated spot in the robot's path with tape to indicate where the bin lid would be placed. The robot is positioned at varying distances from the designated spot from where it would be run. Then, using the wall-follower, it proceeds along its path, encounters the obstacle, and stops. The distance from the marked spot is subsequently measured. Each velocity was tested across three trials, with the results presented in Table 1 and Table 2.

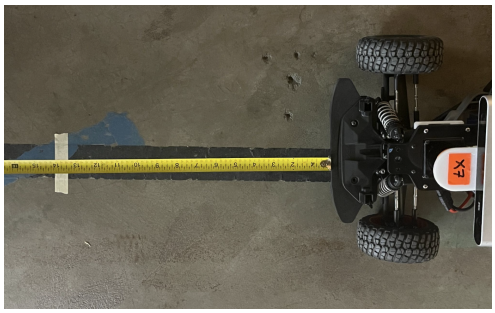

Figure 9: Setup of the experiment for trials recorded in Tables 1 and Table 2.

Trial Number	Stopping Distance (m)
1	0.298
2	0.339
3	0.307

Table 1: Stopping distance of robot across three trials with $v = 1$ m/s.

Trial Number	Stopping Distance (m)
1	0.393
2	0.352
3	0.368

Table 2: Stopping distance of robot across three trials with $v = 2$ m/s.

We found the average stopping distance to be 0.315 m for $v = 1$ m/s and 0.371 m for $v = 2$ m/s. This aligns with our expectations for a time-based safety controller, as a higher velocity results in an earlier stop to compensate for the increased speed. Additionally, there were no outliers in our trials, with stopping distances clustering closely around the mean. The standard deviation was approximately 0.022 m, and the range was 0.041 m for both tested velocities, indicating consistent performance across trials. Based on the data, our safety controller successfully achieved its objective of enabling the robot to avoid collisions while maintaining its functionality.

3.2 Wall Follower

After observing that the robot could follow a straight wall with minimal oscillations, we sought to evaluate its performance during turns. We assessed the robot's performance while making left and right turns in the hallways of the Stata basement at $v = 1$ m/s and $v = 2$ m/s in Table 3 and 4, respectively. We measured both the average minimum and global minimum distances from the wall using ROS bag data collected during the robot's turns.

Turn Type	Avg. Minimum Distance From Wall (m)	Global Min. Distance From Wall (m)
Left Turn	0.928	0.893
Right Turn	0.836	0.694

Table 3: Minimum distance from the wall during turns at $v = 1$ m/s

Turn Type	Avg. Minimum Distance From Wall (m)	Global Min. Distance From Wall (m)
Left Turn	0.655	0.648
Right Turn	0.382	0.157

Table 4: Minimum distance from the wall during turns at $v = 2$ m/s

We also plotted the error, defined as the difference between the robot's minimum distance from the wall and its desired distance of 1 m, during left and right turns at $v = 1$ m/s and $v = 2$ m/s in Figures 10-11 and Figures 12-13, respectively. A positive error indicates that the robot was further from the wall than its desired distance of 1 m while a negative error signifies that the robot was too close to the wall.

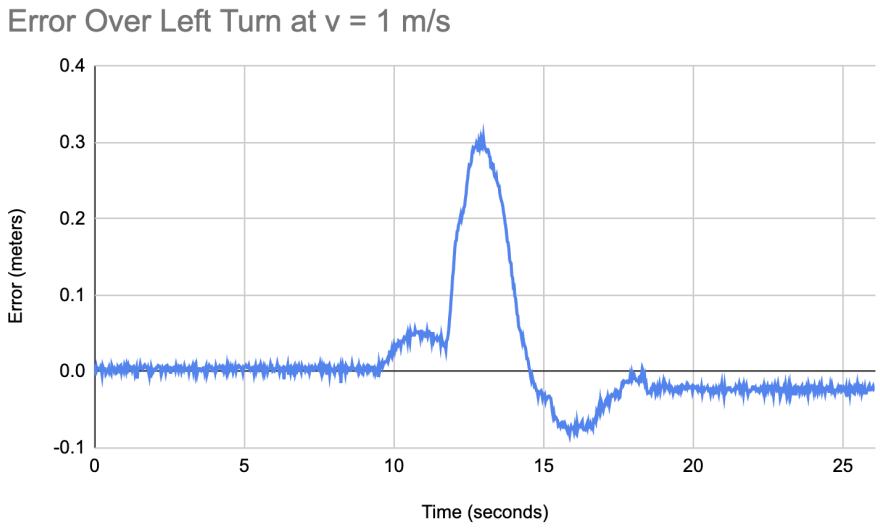

Figure 10: Graph of error over time during the robot's left turn at $v = 1$ m/s.

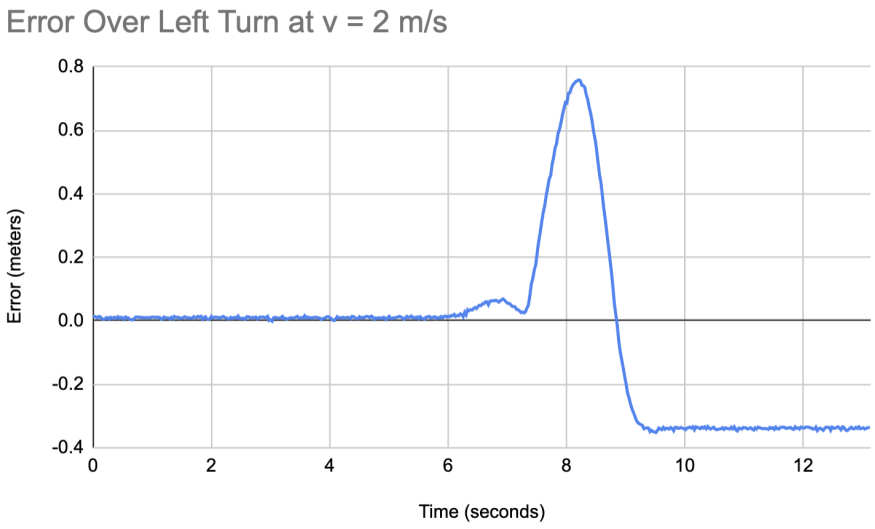

Figure 11: Graph of error over time during the robot's left turn at $v = 2$ m/s.

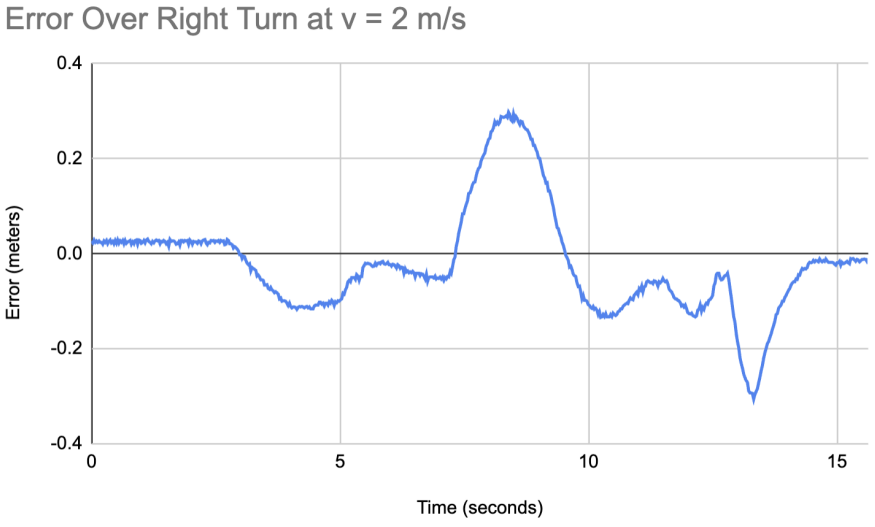

Figure 12: Graph of error over time during the robot's right turn at $v = 1$ m/s.

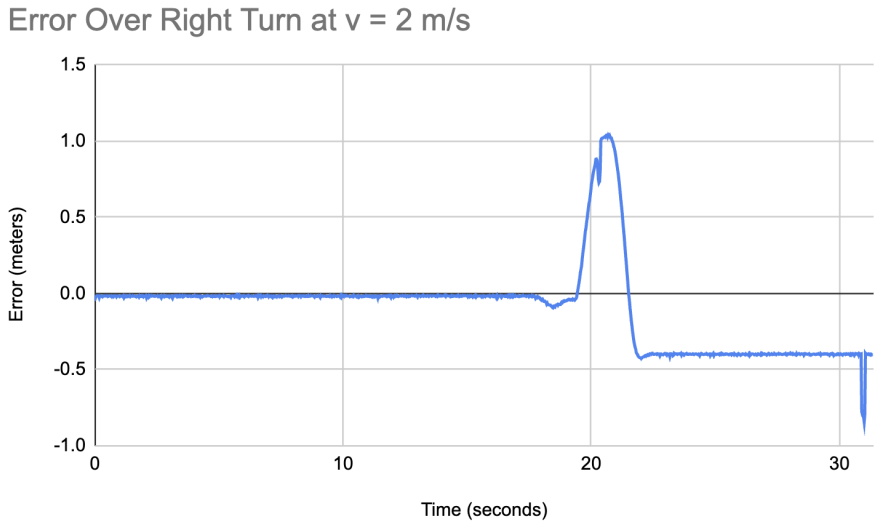

Figure 13: Graph of error over time during the robot's right turn at $v = 2$ m/s.

From the data, it is evident that the robot performs worse when making right turns compared to left turns, exhibiting a larger error and more oscillations during its right turns. Additionally, the robot's performance is significantly degraded at the higher velocity of $v = 2$ m/s, where it fails to successfully complete the turn towards the end, requiring the intervention of the safety controller to prevent a collision with the wall. While there is still room for improvement, our wall follower successfully achieved our objective of consistently tracking the designated side of the wall with minimal oscillations while maintaining the desired distance at $v = 1$ m/s.

4 Conclusion

Author: Asa Paparo
Editor: Cristine Chen
In this lab, we successfully implemented robust wall following and safety controllers that met the qualitative criteria we outlined. Specifically, our wall follower can follow either wall at a constant velocity and perform inward, outward turns without oscillation. The safety controller has been tested to avoid crashing into various targets at multiple velocities. In the future, we intend to improve our line following algorithm to be more rigorous than the sum of a distance and angle controller, as this will allow for more consistent results when following tape during subsequent design phases. We also aim to tune our PID values

to accommodate higher velocities, as we primarily used $v = 1$ m/s for the assessment of our system. Higher velocities have shown to cause difficulties in making turns, and optimizing the PID parameters will help improve the robot's performance at these speeds.

5 Lessons Learned

5.1 Monica Chan

5.1.1 Technical

I learned that using RViz for debugging is highly beneficial, as it helps identify issues that may not be immediately apparent. For example, while collecting ROS bag data for turns, I noticed that one turn had an unusually small minimum distance. By visualizing the sensor data in RViz, we discovered that the LiDAR was detecting wires, which caused the discrepancy. This experience reinforced the importance of using visualization tools to diagnose unexpected behaviors in the system.

5.1.2 CI

I learned that having a shared Google Drive was extremely helpful, as it kept all materials organized in one place. Additionally, maintaining a brainstorming document allowed us to record ideas and decisions, ensuring that everyone was on the same page. This improved communication, facilitated information sharing, and helped establish a clear, shared understanding of tasks and objectives.

5.2 Cristine Chen

5.2.1 Technical

I learned how to set up a launch file and configure it with parameters from YAML files. Additionally, I discovered that storing PID tuning parameters in the YAML files is a best practice, as it allows for easier adjustments without the need to modify the code directly.

5.2.2 CI

I learned that maintaining comprehensive documentation for each part of our system is essential. Since our team members have different schedules, having a shared resource allows anyone attending office hours to access the necessary information without waiting for others to respond. This is especially important given that we worked on different aspects of the project, ensuring continuity and efficiency in our collaboration.

5.3 Kyle Fu

5.3.1 Technical

I learned there are major differences between the simulation and the physical racecar. In particular, the PID parameters had to be tuned down a lot on the physical racecar due to different car dynamics. When we first used the simulation values, the car oscillated a lot, requiring us to dampen parameters. We tuned the distance and angle controllers separately to efficiently find good parameters. It's much faster to iterate in simulation than on the physical racecar. Also, I learned that robotics is full of approximate algorithms, and that it's basically impossible to account for all edge cases in a robotics system.

5.3.2 CI

I learned having a consistent, weekly meeting time outside of class helped with keeping our team on track, without the added decision fatigue of having to coordinate on a new time every meeting. I also learned the importance of keeping solution approaches understandable, which helped when explaining code to team members and documenting it in both the briefing and the report. I practiced making clear assertions, along with the right amount of technical detail for the audience, in briefing slides and reports.

5.4 Asa Paparo

5.4.1 Technical

I learned that the loop times of the VESC low-level controller were lower and less consistent than expected. I originally intended to implement a watchdog as an additional component to the safety controller, which would stop the robot if the commands were not being published at consistent intervals. However, I found that this approach inadvertently caused the robot to become paralyzed, as commands were not being published consistently.

5.4.2 CI

I learned that having a shared communication channel was invaluable for collaborating with teammates. We created a Messenger group chat when our team was initially put together, and we have used it very frequently since then. It facilitated coordination for meetings during office hours and tracked each team member's progress. Additionally, it proved useful for asking questions and receiving advice and opinions from others.