Lab 3 Report: LiDAR-Based Wall-Following for Autonomous Race Cars

Team 4

Adelene Chan
Alan Chen
Paul Gregory
Eghosa Ohenhen

6.4200 Robotics: Science Systems

March 14, 2025

1 Introduction (Paul, Adelene)

The focus of this lab was to develop a navigation system for a miniature autonomous racecar. Our goal was to have our racecar be able to follow a wall given a variety of different situations, stopping if necessary to avoid crashes and damage. Consequently, we developed two key algorithms: a wall-following algorithm that selects optimal driving paths based on predicted trajectories and LiDAR data, and a safety controller designed to override wall-following commands when an imminent collision is detected. We perform a number of tests on our racecar algorithms, measuring their abilities to achieve our goals.

We approached this problem by first building on foundational knowledge gained in previous labs.

The previous labs focused on getting familiar with development tools (working with Linux, Github) and Robot Operating System (ROS) which allow us to modularize various processes in the form of nodes which can pass around topics such as velocity commands or LiDAR scan results. In this lab, we transitioned from simulation to our physical racecar and were required to develop safety controller and wall follower nodes following a priority list for drive command topics. Another challenge that came with this transition was the difference in LiDAR scan data and car dynamics between the simulation and the actual racecar; a good simulation wall follower and safety controller would not necessarily perform well in the real world. The wall follower and safety controller developed in this lab are essential building blocks for developing a complete autonomous navigation system.

2 Technical Approach

2.1 **Wall Follower** (Alan)

The Wall Follower algorithm works by iterating all possible turning angles the racecar can make, evaluate each one, and pick the best angle. A visualization is shown in fig. 1.

Each angle is evaluated assuming the car will stay at turning for the angle for a plan-ahead distance (section 2.1.1).

The path that can go furthest ahead before hitting a wall (meaning that it is closer to a lidar point than a threshold; described in section 2.1.2) is picked; If there are ties (which usually happens), a scoring algorithm is used.

Each path is scored by discretizing the path into points, and scoring each point. Each point gets a reward for being at the right distance to the wall (section 2.1.3), and is penalized for being too close to walls (section 2.1.4). An additional penalty term is used to encorage the racecar to follow the same wall (section 2.1.5).

The points closer to the current is weighted more (section 2.1.6).

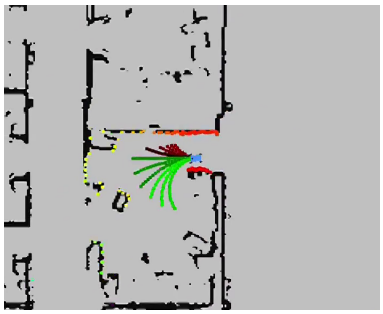

Figure 1: Wall Following algorithm
A subset of paths consider are shown. Green paths are scored higher, and red lower.

2.1.1 plan-ahead distance

The plan-ahead distance is given by

```
plan_ahead = max(2.0 * meters, current_speed * (1.0 * second))
```

Points on the next plan-ahead distance are included to calculate score and the distance before hitting a wall. The plan-ahead distance is increased if the car is faster than `2.0 * meters / second`.

2.1.2 minimal allowed distance

The planned path ends if the racecar is closer to a lidar point than a threshold. Pathes that are longer is always chosen over shorter ones.
The threshold is given by

```
# cur_dist: the distance traveled to the point being evaluated
assert 0 <= cur_dist and cur_dist < plan_ahead
min_allowed_dist = (
    current_speed
    * (1 / 12 * second)
    * min(cur_dist / plan_ahead * 2, 1.5)
)
```

The threshold is increased when further away, so that a path is less likely to become invalid as the car moves and deviates slightly from the computed path.

2.1.3 reward when at the right distance

A reward is given to accomplish the goal of staying a certain distance away from the wall. The reward is shown in fig. 2 and is given by

```
# y: the distance to the closest lidar point that is on the desired SIDE
y = dist(closest_sided(cur), cur)
reward = 1 / ((y - DESIRED_DISTANCE) ** 2 * 4 + 1)
```

The reward is maximized at the desired distance.

2.1.4 penalty when close to walls

A point on the path is penalized if its distance to a lidar point is too small.
The penalty function is shown in fig. 3 and is given by

```
# x: distance to the closest lidar point
x = dist(closest(cur), cur)
assert x >= min_allowed_dist
if x < min_allowed_dist * 2:
    f = lambda x: exp(1 / (2 * (x / min_allowed_dist - 1.0))) / 2
    penalty = f(x) - f(min_allowed_dist * 2)
else:
    penalty = 0
```

To discourage sudden jumps in path planned, the penalty function is chosen to be a continuous function.

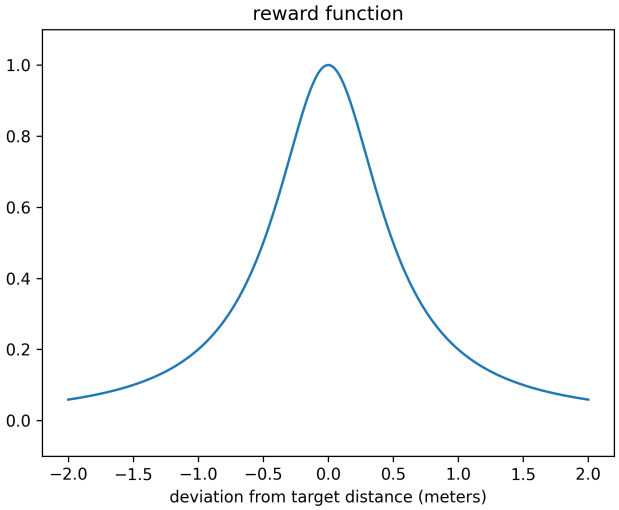

Figure 2: reward function (section 2.1.3)

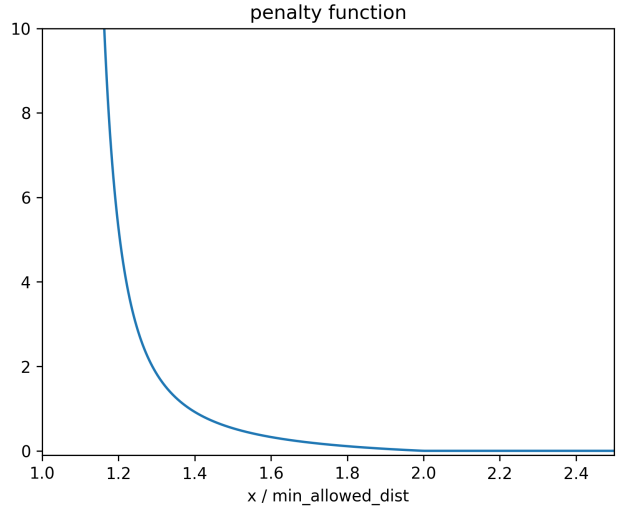

Figure 3: penalty function (section 2.1.4)

2.1.5 penalty for following a different wall

The racecar, when instructed to follow the left wall, might choose to take a right turn at an intersection. Therefore, a penalty term is added if the racecar starts to follow a different wall. It's value is given by

```
# cur: point being scored
# prev: previous point in the path
# closest_sided(x): lidar point closest to x that is on the desired side
dist_to_prev = dist(closest_sided(cur), closest_sided(prev))
# plan ahead distance is divided into 20 points for scoring
assert dist(cur, prev) == plan_ahead / 20
threshold = plan_ahead * 0.15
if dist_to_prev > threshold:
    penalty = ((dist_to_prev / threshold) - 1) ** 2 / 2
else:
    penalty = 0
```

This penalty term, as a function of `dist_to_prev`, is shown in fig. 4.

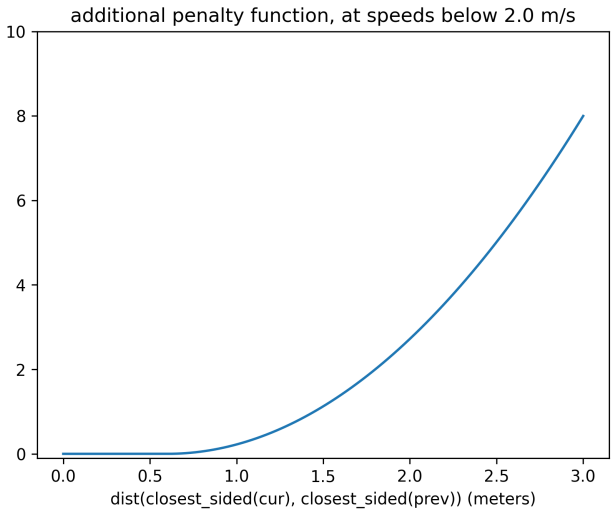

Figure 4: penalty function (section 2.1.5)

2.1.6 summing rewards

score of a path is a sum of scores of each point, with a decay factor.

```
score_path = sum(
    score_point(p) * (2 / 3) ** (i / len(path))
    for i, p in enumerate(path)
)
```

The decay factor considers points closer in future to be more important. The weigthing is show in fig. 5.

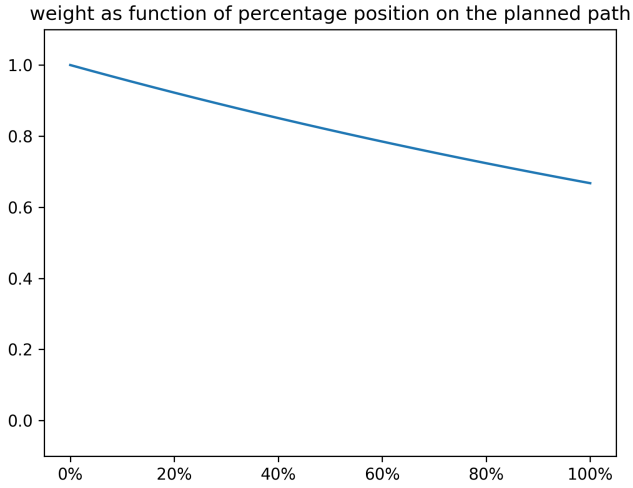

Figure 5: decay function (section 2.1.6)

2.2 **Safety Controller** (Adelene)

The objective of the safety controller is to stop the car in the event that the car is about to crash into a wall or obstacle. It is important that the safety controller is robust yet not too conservative so that it does not stop when trying to follow a wall.

In designing the safety controller, we decided that we needed the safety controller to handle various speeds and obstacle types as shown in Figure 6. While running the racecar, we also observed that the car actually requires different stopping distances for different speeds. When the car is slow, it should be okay to have a smaller stopping distance. This matches the kinematics theory that distance has a squared relationship with velocity as shown in equation (1).

$$x = \frac{v_i^2}{2a} \tag{1}$$

Safety Controller Function: The safety controller takes in the LiDAR scan and the low-level ackermann command being sent to the racecar. The node processes the LiDAR scan by getting distances within a certain angle range in the front based on the current steering angle. For example, the racecar may observe a 45 degree sweep towards the right if the previously sent steering angle is directed towards the right. To evaluate whether to stop, the node sees if the average distance from an n number of minimum LiDAR points is less than the calculated threshold.

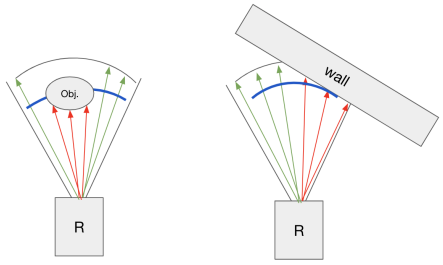

Figure 6: Two possible obstacle scenarios: obstacle (left) and wall (right). The distance threshold for stopping is shown in blue.

To calculate the threshold, the node uses the current velocity command sent to the racecar (from the low-level Ackermann command) and inputs it into an experimentally determined function that outputs a threshold distance. The function was determined by measuring approximate stopping distances starting at several speeds, 0.5, 1, 2, and 3 m/s. Figure 7 shows the experimentally measured stopping distances against the speeds and a linear fit was applied.

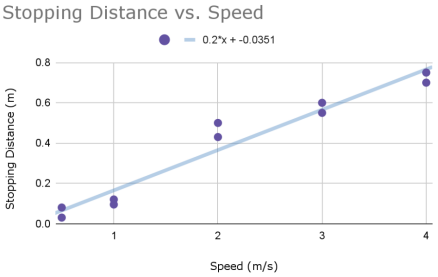

Figure 7: This graph demonstrates the collected data and the line of best fit associated with the data. The trendline determined how the stopping distance threshold was calculated.

This empirically derived formula provided a starting point for the velocity to stopping distance. The final controller, shown in Figure 8, factors in the distance from the LiDAR to the front of the car and an extra finite buffer distance as an offset of $0.2m$. There are two primary conditions, one for a front wall like situation getting the mean of distances in a front slice of the LiDAR and, and one for a smaller obstruction getting the mean of n minimum distances in the same front slice of data.
Finally, to evaluate our safety controller, we conducted several tests to check whether the physical racecar would stop in several scenarios, like a wall or a smaller obstruction (cone). The table below shows the tests conducted, varying speed and obstacles. The results indicate that the safety controller is reliable at smaller speeds, which is within the 2.0 m/s speed requirement in the current

```
def scan_callback():
    # Filter cone of LiDAR scan data
    front_sweep_angle_deg = 45 #degrees

    front_filter = (angles < front_sweep_angle_rad/2 + current_steering_ang) & (angles >
    ranges_filtered = ranges[front_filter]

    # Find min n ranges in front sweep
    num_min_ranges = 2  # number of points
    min_ranges = np.sort(ranges_filtered)[:num_min_ranges]

    # Determine average of all front ranges and min points
    avg_of_front_ranges = np.mean(ranges_filtered)
    avg_of_n_min_ranges = np.mean(min_ranges)

    # Calculate stopping range based on current velocity
    stopping_range = 0.2*current_velocity+0.2

    # Condition 1 - average of front filtered ranges
    if avg_of_front_ranges < stopping_range:
        send_safety()

    # Condition 2 - average of some # of the min ranges
    elif avg_of_n_min_ranges < stopping_range:
        send_safety()

def send_safety():
    drive_msg = AckermannDriveStamped()
    drive_msg.drive.speed = 0.0

    # publish message
    self.safety_publisher.publish(drive_msg)
```

Figure 8: This pseudocode demonstrates how the LiDAR scan data and drive command was processed and used to determine when the car should stop.

labs. However, at faster speeds such as 3.0 m/s, the racecar does not always stop before the wall. There remains room to develop a different function on the speed, such as something that matches the theoretical kinematics or adjust the front angle range the racecar processes as it gets closer or further from an obstacle.

Speed	Obstacle	Stopped?
0.5 m/s	Wall	Yes
1.0 m/s	Wall	Yes
3.0 m/s	Wall	Not always
0.5 m/s	Cone	Yes
1.0 m/s	Cone	Yes
3.0 m/s	Cone	Not always

3 Experimental Evaluation (Eghosa)

3.1 Technical Procedures

We designed our tests with our two key goals in mind, namely minimizing the robot's error when wall following at a desired distance d and also preventing the robot from crashing into the wall. We first broke down the wall following problem into three subdomains:

1. **Angle**: The angle of the initial placement of the robot, measured between the side of the robot and the wall it is following.
2. **Side**: The direction of the wall (left or right) that the robot is trying to follow.
3. **Speed**: The average velocity at which the robot is moving. We chose to test at a speed of 0.5 m/s, 1.0 m/s, and 2.0 m/s.

With each test modifying one or more of these three aspects, we chose a distance-based metric to determine robot performance in each test. We used a utility function, s, based on the magnitude of difference in current distance d_{min} and desired distance d_{des} of the robot in respect to the detected wall (loss l).

$$s = \frac{1}{(1+l)^2} = \frac{1}{(1+|e|)^2} = \frac{1}{(1+|d_{min} - d_{des}|)^2}$$

The goal is to maximize s, as s is highest when the robot is perfectly following the line (error e is zero). We decided to represent each test by plotting the time of the test with the distance error and s.

3.2 Results (Eghosa)

Here are some of the performance vs time graphs generated during multiple wall follower test runs.

3.3 Discussion (Eghosa)

We faced some issues implementing our experimental evaluation due to the brute force wall follower. Unlike systems using linear regression, our robot tested all possible paths and selected the best, making it difficult to accurately

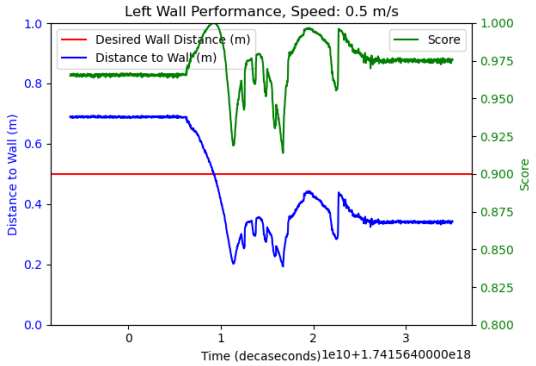

Figure 9: Left Wall, Speed 0.5 m/s

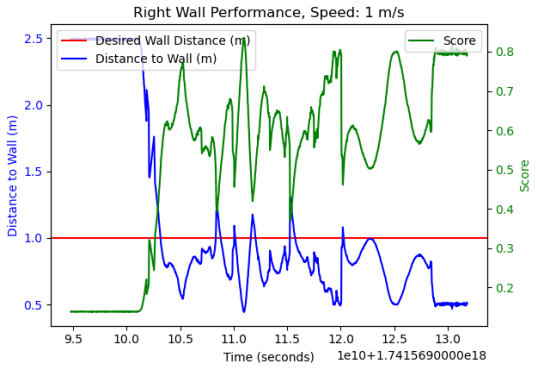

Figure 10: *Right Wall, Speed 1 m/s, Desired Distance 1 meters* In this test, the robot was placed far away from the wall, slightly oriented towards the wall. The test was conducted inside of the classroom, hence the noise.

estimate the distance from the wall and compute error metrics. To resolve this, we introduced a separate ROS node, score_metric_publisher, to calculate and publish the performance score and error. This node required parameters such as the side the robot was following and the desired distances from the wall.
However, discrepancies arose in the recorded metrics, as the scoring parameters were not updated when the wall-following parameters changed. Consequently, we had to replay the ROS bags to align the metrics with the robot's actual performance.
Analysis revealed that the robot performed well on straight paths, maintaining

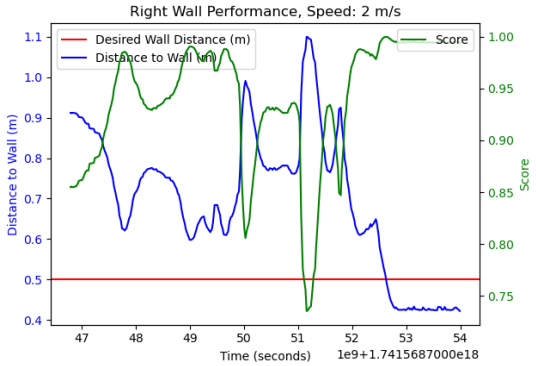

Figure 11: *Right Wall, Speed 2 m/s, Desired Distance 0.5 meters.* This test was conducted inside of the classroom, hence the noise.

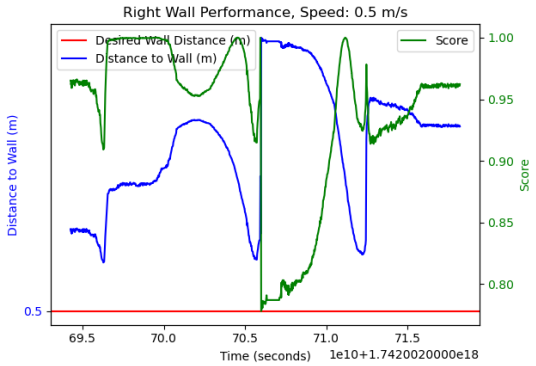

Figure 12: *Right Wall, Speed 0.5 m/s, Desired Distance 0.5 meters.* This test was conducted on the track outside of the classroom.

a stable distance from the wall, as shown in Figure 9. However, the performance degraded on curved paths, where the robot successfully turned but the error increased significantly, particularly during sharp 90-degree turns, such as in Figure 12's test. This could be due to the robot's aggressive distance maintenance during turns, leading to unexpected proximity to the wall when encountering tight corners. This limited its ability to anticipate the wall, potentially triggering the safety controller to stop the robot, a scenario shown in Figure 11 and 12's stabilization near the end of the run.

These results highlight the robot's strength in straight-line following but also

reveal limitations in more dynamic, curved environments. Improved scoring mechanisms and control strategies are needed to enhance performance in such scenarios.

4 Conclusion(Paul)

In this lab, we created and tested algorithms for wall-following and safety control on an autonomous racecar. We succesfully transitioned from simulation to the real world, adjusting for differences in lidar data received and car control. For our wall follower, we implement a brute force method that iterates over possible future paths, choosing the one that best follows the wall. We test our wall follower based on initial angle, side of the wall, and how fast our racecar is moving. We find that the racecar is able to stay a constant distance away from the wall, however it converges to a distance that is not the desired distance. Future work should be done in order to rectify this issue and have the car stay the specified distance away from the wall. Our safety controller uses different distance thresholds for stopping with different speeds to give the car enough time to stop at high speeds. In the testing of our wall follower, we find that at lower speeds performance is exceptional, however at higher speeds the car does not always stop when needed. In future design phases, we plan to integrate more rigorous unit testing to ensure robustness.

5 Lessons Learned

(Adelene)
Throughout this lab, I learned about the importance of being organized and communicating frequently with our team. There are many tasks, from technical code development to briefing and report aspects, so it was critical that we maintained a timeline and kept each other updated. Additionally, unexpected bugs or additional failure modes are introduced on physical hardware and it was helpful to learn about different ways to analyze our tests from using rosbags to rviz for visualization. I enjoyed meeting my team and learning to coordinate and work together. Getting started on a team project can often be tough, but in general it has been exciting to collaborate and learn from my teammates.
(Paul)
In lab 3, I learned about some of the extra things needed to consider when transitioning out of simulation, as the real world introduces much more potential risk with many more possible bugs, as well as accounting for differences in data. I also learned a lot about how to present. Before this lab, every slideshow I ever made was just bullet points, and in this lab I learned that this is not an effective presentation strategy. This lab also provided valuable experience working on a team and taught me the value of communication for effective collaboration.
(Alan)
While our wall follower works, there are some major bugs in the version we

evaluated. One major cause of this is lack of testing; some bugs would be immediately detected and avoided if we had sufficient testing. There is also a overreliance on sim, causing the racecar to significantly underperform on higher speeds, as compared to the sim. This is primarily due to the lack of time spent on the real racecar, and inefficient use of time with the real racecar. We would do better by investing into improving the development workflow on the physical racecar.

(Eghosa)

I learned a lot about keeping in touch with a team this lab and also about the difficulty of migrating multiple products into one. I have worked on github repos with multiple people before, but usually we start from scratch, working together to implement an idea. However, for this lab, we have each came in with our own working implementations and side projects and had to integrate them together not only via Github, but also when sshed onto the robot's WiFi. Trying to push our code from the robot to Github was interesting since we were essentially on the same machine, so a lot of the usual abstractions used when collaborating with Github couldn't be used. This was a very valuable insight for me.