Lab # 3 Report: Wall-Follower and Safety Controller

Team # 10

Diego Delarue
Jack Lewis
Jason Lee
Vlada Petrusenko

6.4200 - RSS

March 11, 2023

1 Introduction (author: Jack Lewis)

After the simulated wall-following of Lab 2, our next step toward building an autonomous car is implementing the wall-following on a physical car. To make sure we are doing this safely, we also implement a safety controller to run in the background and stop the car if it is about to hit anything.

As part of the project, we will look at the differences in the approach and mindset between dealing with a real-life vehicle and a simulated car. The primary difference we will be looking at is the need to implement a safety controller independently of the other programs in order to make sure the car will not be in a collision, which was not an issue in the simulation. If the simulated car ran right through a wall, we just reset it and let it try again. With the real car, though, we don't want the car running into things at all. While it has some padding to protect it and the things it runs into from minor collisions, we would rather not take any chances, especially with high-speed collisions.

These programs will be implemented as ROS nodes in such a way that different levels of control can be executed. We will also take advantage of ROS's publisher-subscriber architecture to allow each node to receive and broadcast all necessary information.

At the end of this lab, we seek to have a car that can find and follow walls on either side, at a given speed and distance from the wall. If in the process of doing this, it is about to run into something, whether the object was already there or appears there while the car is moving, the car should stop until it can proceed without hitting the object.

We find and follow walls by using LIDAR data, eliminating outlier points, finding a line of best fit for the remaining points, and using PID control to control the steering angle of the car to move the car to the specified distance from the wall and parallel to the wall.

We keep the car from hitting things by looking at LIDAR data, looking for readings really close to the car that are anywhere in front of the LIDAR scanner, and stopping if we see any. We also look at the readings that are directly in front of the car and stop if any represent an object that we would hit if we continued at our current speed for the next half-second.

2 Technical Approach

2.1 Wall Following (author: Vlada Petrusenko)

The first part of the project is to use the approach from the simulation and map it on a real-world robot so that it goes along the wall at a predefined velocity and distance from the wall. This is quite an integral part of any autonomous robot, given that it has to be aware of its environment and go along some route without hitting obstacles. While just following the wall might be not the main use case of the autonomous machine, it is definitely a base compound of any other navigation task built on top of it.
Some of the main technical challenges while following the wall are keeping a consistent distance from the wall and not fluctuating in a wave-looking shape, making turns on both sides without crashing into the wall, and finding a wall when getting lost. Some of those actions require completely opposite PID parameters, so the extra challenge is to tune them so that the system as a whole is robust and can reasonably pass all of those challenges.

We started off by simulating the anticipated and desired robot behavior, and constructing a steering control system so that the robot is able to detect the wall and follow it independently of its shape and turns. We also had a robot provided by RSS staff that can be controlled by a joystick or by the ROS code. The main sensor is the LIDAR sensor that scans the environment and returns the distance to any object in a predefined sector.

In terms of technical approach, the wall-following algorithm consists of the following steps:

0. Fetch all the global parameters from ‘params.yaml‘ file (organized this way to facilitate ease of changing relevant parameters.

1. Set up a Subscriber node collecting all the data from LIDAR scanners. The resulting data describes the robot's field of vision from minimal to maximal angle in polar coordinates (which means that distance to the object is provided and respective angles can be calculated from received data).

2. For the received data truncate it so that only relevant data is considered. While this is a parameter that can be changed to achieve better performance, the current choice is data from 11.25 to 90 degrees from the front axis of the robot to the side where the wall is being searched.

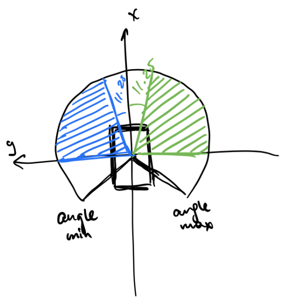

Figure 2.1.1. Range of vision of the LIDAR scanner and the data used to detect the wall (marked green and blue for right and left wall respectively).

3. Convert data from polar to Cartesian coordinates. Namely, for every polar point (r, θ) convert it to polar through $(r \cdot \cos\theta, r \cdot \sin\theta)$. That conversion is a relatively well-known fact, which also can be proved by the definition of cosinus and sinus (for reference, see the figure below).

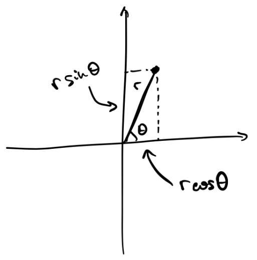

Figure 2.1.2. Converting polar coordinates into Cartesian.

4. Filter out outliers. There might be various ways to achieve a clean state of the data so that there are only points from the wall (it was especially critical during the simulation where not all the data was thoroughly recorded and it was common for accidental outlier data points from other walls to be recorded. The approach we used was to iterate through the data points, assume that the very first one belongs to the wall, and check for every next one whether it is at a reasonable distance from the previous one ($0.1 \cdot \mathit{DesiredDistance}$ in our

implementation, but can be adjusted if needed). Consider the point "belonging to the wall" if and only if it's within the closeness distance to the previously considered point, disregard otherwise. This algorithm might misidentify the wall once in a while, but those false identifications don't affect the steering much, while corrected right identifications improve steering a lot.

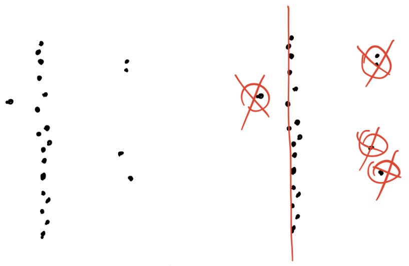

Figure 2.1.3. Identifying and discarding outliers.

5. Use least square regression to fit the remaining data points on the line. Consider this line to be approximating the wall. Generally speaking, least square regression is an algorithm that finds the line that fits data points the best - by minimizing the sum of squared distances from the predicted line. Formally speaking, the algorithm finds m and b, such that for a set of points $(x_0, y_0), ...(x_i, y_i), ...(x_n, y_n)$, the values of m and b are such that $\Sigma_0^n((y_i - (m \cdot x + b))^2)$ is minimal. In the implementation, the algorithm to find such m and b is to use that $A^T Ax = A^T y$, then for finding y it's enough to compute the matrix $A^T A$, invert it, multiply by A^T and then by y to get $\hat{x}$ which is the solution to the least square regression problem.

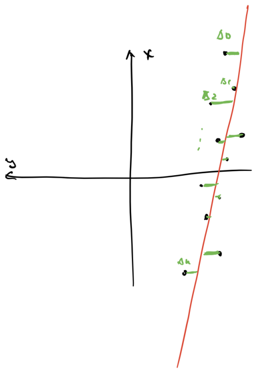

Figure 2.1.4. Least square regression tries to minimize the sum of squares of the distance to the line (marked green on the figure).

6. Use the PID controller to adjust the steering angle of the car. PID controller is being used for cases like autonomous driving where there is a feedback loop that helps to adjust the result. PID controller works by setting up Proportional, Integral, and Derivative coefficients so that they adjust input parameters to get

the desired output (in our case adjust the steering angle to get the desired distance and angle to the wall). There might be several approaches to construct the reference point, but in our implementation, we use $m + b$ and trying to match it to $DesiredDistance$, because m is anticipated to be 0 in a perfect case scenario, and we are aiming for it, and —b— should be exactly $DesiredDistance$. So this arithmetic construction optimizes for the correct configuration, although it's obviously not the only correct way of comparing PID output. Use the Publisher node to control the driving mechanism of the car by publishing calculated m and b to the car control system.

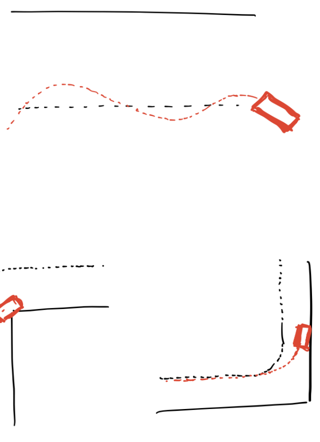

Figure 2.1.5. Possible issues with the suggested solution: overshooting, not making the turns on time, etc.

2.2 Safety Controller (author: Diego Delarue)

The safety controller was incorporated into this lab as a consequence of having a real-life car. This is out of a need to protect the equipment and make sure that autonomous vehicles are able to avoid danger. At its most primitive level, the goal of the safety controller is to protect the car by preventing it from getting into collisions. However, the project is leading up to the final competition, so we do not want the car to be "too scared" to crash and slow us down.

By the time we started to brainstorm how to address the safety controller, we had already successfully implemented the PID wall follower in simulation as well as in real life. Due to this fact, the biggest challenge with the safety controller was to think of a good design rather than the implementation. The main concept we chose is to have two conditionals such that if one is true the car stops as shown below by the pseudo-code.

```
ScanTolerance = closest range allowed without stopping
FutureTolerance = closest predicted range allowed without stopping
t = chosen reaction time in s
ranges = array of LiDAR ranges
v = velocity given in driving command
theta = steering_angle given in driving command
alpha = front angle of car to consider (shown in Figure 2.2.1)
range_min = smallest possible LiDAR range

# make sure no range is too small (too close to the wall)
if min(ranges) < max(ScanTolerance, range_min):
    STOP CAR

# make sure car will not crash after
# ReactionTime seconds at the given velocity
if min(ranges[theta-alpha: theta + alpha]) - v * t < FutureTolerance:
    STOP CAR
```

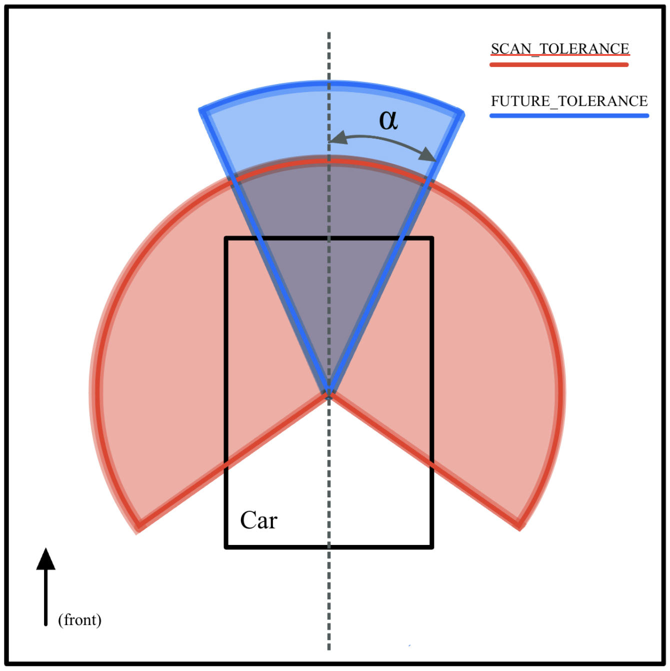

Figure 2.2.1 Safety controller will stop car when within the shaded ranges

The initial values we assigned to our variables were:

$$\begin{aligned} t &= 0.5 \text{ seconds} \\ \alpha &= 0.34 \text{ radians as the maximum turn angle} \\ ScanTolerance &= 0.25 \\ FutureTolerance &= 0.5 \end{aligned}$$

Once we conceptualized the safety controller, the next issue to address was to decide how to implement it in a way that overrides any other system that might be implemented on the car. We wanted our safety controller to be independent of any other system that could potentially be implemented in the car so we can guarantee that no matter the program, the car's well being will be protected. By taking advantage of the command mux with multiple levels of priority installed in the car (Figure 2.2.2), we were able to let the safety controller publish to the highest priority topic such that if either of the two conditions is triggered no other system could override the controller's commands. As shown in the image below, the structure that was implemented in the command mux is designed to let the safety controller publish to $'/vesc/high\_level/ackermann\_cmd\_mux/input/safety'$ and have the highest priority.

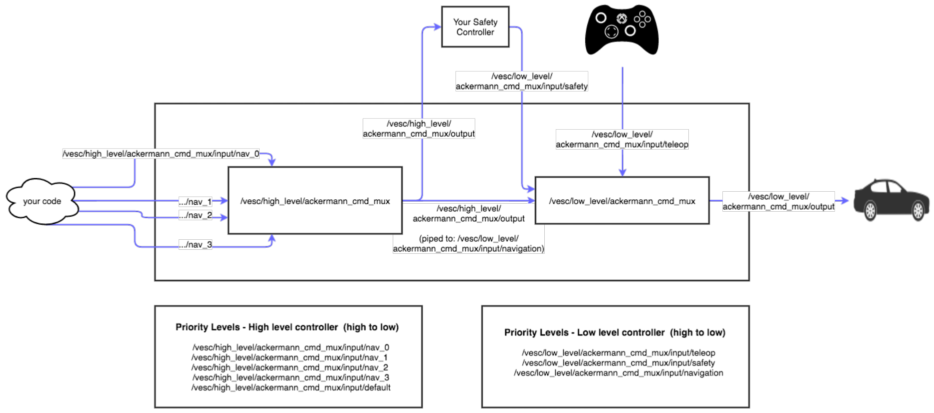
Your Safety Controller
/vesc/low_level/ackermann_cmd_mux/input/safety
/vesc/high_level/ackermann_cmd_mux/output
/vesc/low_level/ackermann_cmd_mux/input/teleop
/vesc/high_level/ackermann_cmd_mux/input/nav_0
your code
.../nav_1
.../nav_2
.../nav_3
/vesc/high_level/ackermann_cmd_mux
/vesc/high_level/ackermann_cmd_mux/output
(piped to: /vesc/low_level/ackermann_cmd_mux/input/navigation)
/vesc/low_level/ackermann_cmd_mux
/vesc/low_level/ackermann_cmd_mux/output
Priority Levels - High level controller (high to low)
/vesc/high_level/ackermann_cmd_mux/input/nav_0
/vesc/high_level/ackermann_cmd_mux/input/nav_1
/vesc/high_level/ackermann_cmd_mux/input/nav_2
/vesc/high_level/ackermann_cmd_mux/input/nav_3
/vesc/high_level/ackermann_cmd_mux/input/default
Priority Levels - Low level controller (high to low)
/vesc/low_level/ackermann_cmd_mux/input/teleop
/vesc/low_level/ackermann_cmd_mux/input/safety
/vesc/low_level/ackermann_cmd_mux/input/navigation

Figure 2.2.2 Overview of command mux with safety controller topic assigned highest priority.

Given the decision to have the safety controller publish to the highest priority topic, all that was left was to decide how to implement the ROS structure with the right subscribers to be able to operate the safety controller. The two pieces of information needed to check for the two conditions are (1) the LiDAR data and (2) the driving commands sent to the robot. For this, we set up two subscribers to '/scan' and '/vesc/high_level/ackermann_cmd_mux/output' for the respective requirements. All of this could be implemented inside a single ROS node which we named 'safety_controller'.

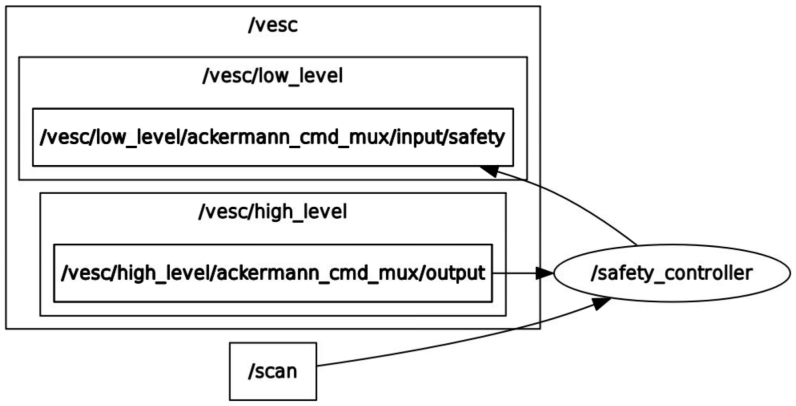
/vesc
/vesc/low_level
/vesc/low_level/ackermann_cmd_mux/input/safety
/vesc/high_level
/vesc/high_level/ackermann_cmd_mux/output
/safety_controller
/scan

Figure 2.2.3 rqt_graph of 'safety_controller' node interacting with the racecar in simulation.

As you can see in Figure 2.2.3, our safety controller is successfully able to subscribe to the '/scan' topic for the LiDAR scans and '/vesc/high_level/ackermann_cmd_mux/output' for the driving commands. Once we have this information as a sensor_msgs/LaserScan Message and ackerman_msgs/AckermannDriveStamped Message respectively, we can access all the data we need to implement our safety controller with ease.

3 Experimental Evaluation (author: Jason Lee)

We now present both qualitative tests and empirical data that were evaluated to determine the performances of the wall-following algorithm and the safety controller.

3.1 Wall Following

The main qualitative assessments we made of the wall-following algorithm were mostly that of whether or not the car could accurately navigate hallways on different combinations of parameters. These parameters were:

- Following the left or right edges of the hallway (binary)
- Speed of the car (0.5, 1, 2, and 4 m/s)
- Distance kept from the wall (0.25, 0.5, and 1 meter)

We noticed that of all combinations, the highest speed tested caused the most pronounced fluctuations across a central line parallel to the wall. This is likely due to the frequency of steering angle updates being constant across all tests, causing the car to cover more horizontal distance while travelling. Additionally,

after initial tests we were able to remove a distance of 0.25 meters from consideration as the car would clip corners, a problem only exacerbated by higher speeds.
The final qualitative assessment we noticed was that our car often had difficulties following hallways that were interspersed with black objects or colors. Instead of continuing to follow the wall or navigate around obstacles, the car would attempt to turn into the black objects or patches of paint. We hypothesize this is due to the inherent nature of LIDAR scanning using electromagnetic pulses to determine distance and darker colors absorbing more of these frequencies. While we are unable to propose a solution with the current technology we have, we believe that the incorporation of camera data and vision modules will alleviate this issue as edge detection and perspective can help us distinguish between an object and a separate path.

We were able to collect a rosbag of the car's navigation using the wall-following algorithm to take quantitative measurements, but were unable to perform analysis due to time constraints. Our goals for this analysis are outlined in Section 4.

3.2 Safety Controller

Due to the battery and time constraints with the car, all observations made and presented in this section are qualitative in nature. Our initial test of the safety controller involved whether or not the car autonomously stops when it encounters an obstacle it is unable to navigate around and would otherwise collide with. These experiments were conducted using a large plastic sheet, which the car successfully stopped in front of at the various speeds previously tested with the wall-following algorithm. While we anticipate there is some variation between the theoretical and empirical stopping distances as well as a correlation with car speed, we were unable to collect this data.

Additionally, we performed tests to ensure the priority system between manual input, the safety controller, and autonomous navigation was implemented properly. Firstly, while constantly holding the controller trigger for autonomous navigation, we placed an unavoidable obstacle (the previously mentioned plastic sheet) in the car's path, then removed it once it was approached by the car. The car correctly stopped once it reached the obstacle, then continued travelling after the sheet was removed without the need to restart the navigation. We were also able to force a collision using manual input, proving it had priority over the safety controller.

Finally, when re-navigating the same paths as the wall-following algorithm test suite but with the addition of the safety controller, we noticed that while the car continued to turn towards black objects and dark-colored portions of walls, the safety controller was actually able to prevent collisions with these obstructions.

This leads us to believe that while LIDAR data on dark colors is inconsistent and inaccurate from a distance, it grows more accurate the closer the sensor is to these objects, since the sensor data is used in both the wall-following algorithm and the safety controller.

4 Conclusion (author: Jack Lewis)

After adapting our wall-follower from the simulated car to the real car, we were able to quickly find walls and follow them relatively stably. There is still more we could do to improve this stability, especially at higher speeds, but the car didn't ever suddenly lose the wall for no reason. The car also couldn't see black objects, at least until it was very close to them. This could likely be compensated for in future labs by also looking at camera data, with which we could better identify the difference between black objects and empty space.

After confirming that our wall-follower worked, we implemented our safety controller, which was able to reliable stop the car from running into objects, even quite small ones. After any obstacles had been removed, the safety controller then allowed the car to continue driving.

We would like to perform additional quantitative tests to limit-test and calibrate our wall-following algorithm and safety controller. First, we would like to collect the horizontal distance of the car following a straight wall, and quantify the variation with respect to the goal distance. Collecting this information across variable speeds would enable us to plan for an optimal combination of car speed and goal distance while minimizing the likelihood of an interruption from the safety controller. Additionally, we plan to conduct an experiment on the safety controller in which we drive the car perpendicular to a wall and compare the actual stopping distance to the theoretical distance. Similarly to the previous experiment, these measurements across different speeds would help us set a safe threshold while not impeding normal autonomous navigation of the car.

Going forward, we will add the ability for the car to use data from the camera to decide where to drive, which we may also add to the wall-follower so it can properly detect black objects.

5 Lessons Learned

- Diego: I learned how to use ROS efficiently to be able to have multiple nodes subscribe to a topic and handle them in different ways (as was the case with wall_following and safety_controller). I also learned how different levels of priority can be introduced like in a command mux to be able to have different systems be at different priorities (such as the safety_controller stopping the car even when wall_follower still wants the car to move. Finally, I learned how to be able to start coordinating with your own team as well as with others when it comes to reserving and testing with the racecar.

- Vlada: I learned that with the scaling of the number of people involved, it gets gradually harder to communicate things and arrange meetings. From the technical perspective, I learned a lot about ROS functionality and node structuring, as well as experienced the differences between the simulation and the physical car. Also I learned that batteries discharge really quickly :)

- Jason: Through the help of TAs in debugging sessions, I learned a number of indicators and commands that tip what part of the hardware or software is causing issues when running programs on the car. Additionally, hardware is very finicky, and consistent communication both with my team and with other teams is essential for the project to go smoothly.

- Jack: I learned that there are a lot of ways that the car, its code, and its controller can go wrong, even if all of the code you wrote is right. If one of the cables has not been plugged in properly, or a command has not been run, or an environment has not been initialized properly, there can still be bugs that are often quite hard to find the source of.