Lab 3 Report: Designing an Autonomous Controller for Robust Racecar Navigation

Team 7

Pranav Arunandhi (editor)
Thomas Fisher
Brady Klein
Calvin Maddox
Gustavo Ramirez

Robotics : Science and Systems

March 10 2023

1 Introduction

The goal of this lab is to implement a wall follower capable of autonomously locating and maintaining a constant distance from a wall on either its left or right side, chosen on the fly, while driving forwards. Our implementation achieves the above objective while being robust to uneven surfaces and small errors in the LIDAR data.

Our controller is able to recover from small deviations from the desired state; being too far, too close, or too angled by implementing proportional-derivative (PD) control.

In addition to the wall follower controller, we also develop a safety controller that prevents the racecar from crashing into obstacles. The safety controller uses data from the LIDAR sensor to detect potential collisions and apply emergency brakes to stop the vehicle if necessary. When designing our safety controller, we ensured it is robust but flexible- capable of preventing collisions without being overly cautious. The safety controller developed in this lab is intended to be used for all future labs, enabling us to focus on developing more advanced control algorithms in the future without having to worry about safety concerns.

Beyond learning how to implementing a wall-follower algorithm using PID control techniques, this lab allowed us to familiarize ourselves with the physical

car, the procedures for uploading and adapting our code to the racecar platform, streamline our debugging process, understand the data structures received from the racecar sensors, design effective ROS architectures, and collaborate effectively, all of which are essential skills for future projects in this class and in the field of autonomous systems.

2 Technical Approach

2.1 Wall Follower Design - Brady Klein

In addition to the safety controller, our ROS package contains a wall follower node which allows it to navigate around its environment. This ROS node works as both a subscriber and a publisher. The subscription topic is `/scan`. This topic contains LaserScan messages which hold information surveyed from the mounted LIDAR device and describe the distances to points around the car. The node's publishing topic is `/vesc/ackermann_cmd_mux/input/navigation`. This message contains information about how the car should steer in order to avoid obstacles detected by the LIDAR laser scans.

The first step of our wall follower's callback function is to take the LIDAR scan metadata and construct a NumPy array of the angles relevant to the scan's ranges. Once we have the associated angles, we can splice both this angle array and the scan's ranges in order to filter out laser scan data which is not relevant to detecting the wall we're following. After splicing the data, we can use the points detected in order to fit a regression line which approximates the wall we wish to follow. We pass this regression to a feedback controller which calculates error and determines an appropriate action to make - in our case, the desired steering angle for the car. Finally, once the action has been defined, we create an `AckermannDriveStamped` message which contains all the relevant steering information that the car will need to navigate.

2.1.1 Input

The LaserScan message contains many useful fields which describe the obstacles around the car's LIDAR sensor. Four of these fields will hold particular importance for various methods of our wall follower class: `angle_min`, `angle_max`, `angle_increment`, and `ranges`.

2.1.2 Calculate Angles

In order to make our LaserScan data more useful for following walls, we first create an array which defines all the angles at which the LIDAR surveyed a range. To do this we simply iterate from the scan's `angle_min` to `angle_max` using the `angle_increment`. Although a simple calculation, the created array

will allow us to construct a geometric model of our environment which will be helpful later.

2.1.3 Splice Ranges and Angles

In our package's parameters, we have a specified side for which we should follow the wall and a desired distance and velocity at which to follow it at. Because we are only concerned with walls to one side of the car, we can actually splice our data in order to consider only the data relevant to that side. In order to accomplish this, we must find the center index of the scan ranges. Once this index is found, we splice our list to contain only points before or after that index. This will effectively cut our scan data in half, keeping only the information relevant to the desired side.

2.1.4 Fit Wall Regression

Once we have spliced our data into the relevant subset, we can now begin to build our geometric model of the environment. Taking an angle and the associated range, we use simple trigonometry in order to graph the point on a two dimensional plane. Once all points have been mapped, we filter out any points which do not fall into a window of twice the desired distance. If a point falls outside this window, we remove the point from our array before the regression is calculated.

If the array is non-empty after this filtering step, we use the remaining points to fit a regression line which represents the wall we are trying to follow. This regression line will give us two important metrics: the angle from the wall and the distance from it. Our controller will use these metrics in order to determine a steering angle for the car to take.

If the array is empty after the filtering step, we raise a `NoDataPointsException`. If this exception is raised, the controller will set the action to zero. This effectively means that if the car cannot identify any relevant data points on which to fit a regression, it will drive straight until a regression can be made.

2.1.5 Controller

For our wall follower, our team experimented with both a PD controller and a PID controller. We eventually determined that the PID controller did not provide any significant advantages for our controller at this time. We found that the integral term was best suited for counteracting persistent convergence errors that may result from disturbances to the system. As no relevant external forces were acting on our car, we found that we could achieve the same navigation functionality without the integral as we could with it.

Our package has both a K_p and K_d parameter defined. We can use these

parameters in conjunction with a calculated error to determine an action to take. Our action equation is as follows:

$$\texttt{action} = \texttt{side} * (K_p * e + K_d * \theta * \texttt{velocity})$$

In this equation, `side` is a parameter equal to either -1 or 1, depending on which wall we are trying to follow - right or left, respectively. The error e is calculated as the difference between desired distance to the wall and the observed distance to the wall, which we can retrieve from our regression line. θ refers to the angle between our current heading and the desired direction. Finally, `velocity` is another parameter of the package and is in the range $[0, 4]$ m s^{-1}.

2.1.6 Output

The final step before publishing is to construct an `AckermannDriveStamped` message in order to format our navigation instructions to be sent. This message has several fields, but for our purposes we can set `steering_angle_velocity`, `acceleration`, and `jerk` all to zero, which indicates to the physical controller that any changes to `steering_angle` or `velocity` should be made as close to instantaneously as possible. The most important field is the `steering_angle`. We set this field equal to the action which our PD controller calculated. Finally, we set the drive speed to our velocity parameter and stamp the message before publishing.

2.2 Safety Controller Design - Calvin Maddox

The second part of our team's technical approach involved designing a safety controller for our vehicle. Our primary goal in developing this controller was to prevent any collisions that could feasibly be avoided by the car (i.e. obstacles that are detectable by the car – we can not account for objects dropped from above or placed behind for example). Our secondary objective was to accomplish this first goal while minimizing the possible disruption to the behavior and performance of any program running on the car. In support of this aim, we developed a two-step process that checks for conditions both currently in the car's path, as well as along the car's projected path, in order to determine whether any actions are necessary on the part of the controller.

2.2.1 Nodes

We initialize three key nodes in order to implement our controller: one input node I_d which reads drive instructions sent to the car, one input node I_s which reads the LIDAR scans, and one output node O_d which writes drive instructions in the event that the safety controller has to override the instructions read by I_d.

2.2.2 Safety Controller Process

Each time a scan is read by I_s, the data is saved in the safety controller object, to be used when determining the car's position relative to the world around it. Thus, when a drive instruction is read by I_d, the first step is to access this most recent scan and pull out the list of values representing the different distances measured at each angle. Our controller then completes two steps of verification to ensure that the drive instruction does not cause a collision. First, the distance from the front of the car to the closest object is compared to the angle that is being sent to turn to in order to determine if the car has room to take this action, which we'll call the "bumper check". Second, the angle the car is turning to is compared with the distance to the nearest object at that angle, and it is determined whether the car could stop in time to avoid a collision if set to that angle, which we'll call the "brake check." If either of these checks fail, then O_d will send an instruction to override the current drive instruction. Both of these checks are described in more detail below.

2.2.3 Bumper Check

The main crux of the bumper check lies in determining the turning radius of the car and ensuring that it does not exceed the distance from the front of the car. In order to calculate turn radius, we divide the wheel base of the car (the distance from the front to rear axle) by the tangent of the turn angle θ that is part of the input drive instruction (this causes an issue when θ is 0, but this case is covered by the brake check, so we make sure that $\theta \neq 0$ first):

$$r = \frac{\texttt{wheel\_base}}{\tan(\theta)}$$

If this value is greater than the distance in front of the car, we know that corrective action is required, as the car's projected path would lead to it running into a wall or obstacle. However, in order to support our second goal of minimizing impact on the actions that the drive instruction is attempting to take, we do an additional check to see if the course can be corrected before instructing the car to brake. This involves taking the maximum turning angle (which we found to be around 40 degrees, or .73 radians) and using it as θ in our turn radius calculation. If this radius falls below the distance to the nearest obstacle, we know that by correcting the steering to have a tighter angle, we can avoid a collision, so O_d sends a drive instruction with the same values as the one received by I_d, but with this higher turn angle in the desired direction. On the other hand, if the turn radius calculated is still greater than the distance to the nearest object, then we know that corrective action can not stop a collision, and so O_d sends an instruction to the car to brake (an `AckermannDriveStamped` message with 0 in all fields).

2.2.4 Brake Check

If the bumper check passes, the car then performs a "brake check" to ensure that it has sufficient time to brake were it to perform the instruction received by I_d. We first find the projected distance, d, to the car's heading by taking the scan from I_s and finding the distance measured at the angle θ from the instruction received by I_d. We then calculate a time to obstacle, t_o as follows:

$$t_o = \left(\frac{1}{v}\right)^{c_p} \cdot d$$

where v is the velocity sent by the instruction, and c_p is a constant we have introduced in order to tune performance (we found a value of 3 for c_p led to desirable performance). We compare this time with a time to brake t_b calculated as follows:

$$t_b = \frac{c_b}{v}$$

where c_b is another constant we introduce to tune performance (we found a value of .25 for c_b led to desirable performance). If we determine that $t_b \geq t_o$, then O_d immediately sends a brake message, as the car must immediately start braking in order to avoid colliding with the object in its path.

This two step process helps to ensure that our car can perform any actions assigned to it as well as possible while still remaining safe and avoiding crashes. In the event that we believe that we can amend the instruction to avoid a crash we do so, otherwise we instruct the car to immediately brake, meaning that we always accomplish our first goal of avoiding crashes, and can often do so while still accomplishing our second goal of keeping alterations to the instructions at a minimum.

3 Experimental Evaluation - Thomas Fisher

There were a variety of methods that our team employed to test the functionality of our wall follower implementation and to gauge its accuracy on both a qualitative and quantitative level. This section describes those methods and presents evidence in support of the conclusions our team drew from our results.

3.1 Visual Assessment

The most basic but also most fundamental form of verification that our team used to determine whether our wall follower algorithm was able to successfully navigate a range of environments was visual confirmation. By observing the car's speed and turning angles as it traversed a section of wall, our team was able to either verify that the car's behavior was consistent with what we expected of our algorithm, or to identify an issue in the car's movement that would prompt an investigation into the possible cause.

While difficult to quantify, our team's visual observations were the largest contributor towards adjustments of the wall following algorithm out of all of the assessment metrics presented. This qualitative feedback is instantaneous, easily reproducible, and intuitive, and was the first metric our team turned to when assessing performance.

3.2 Simulated Assessment

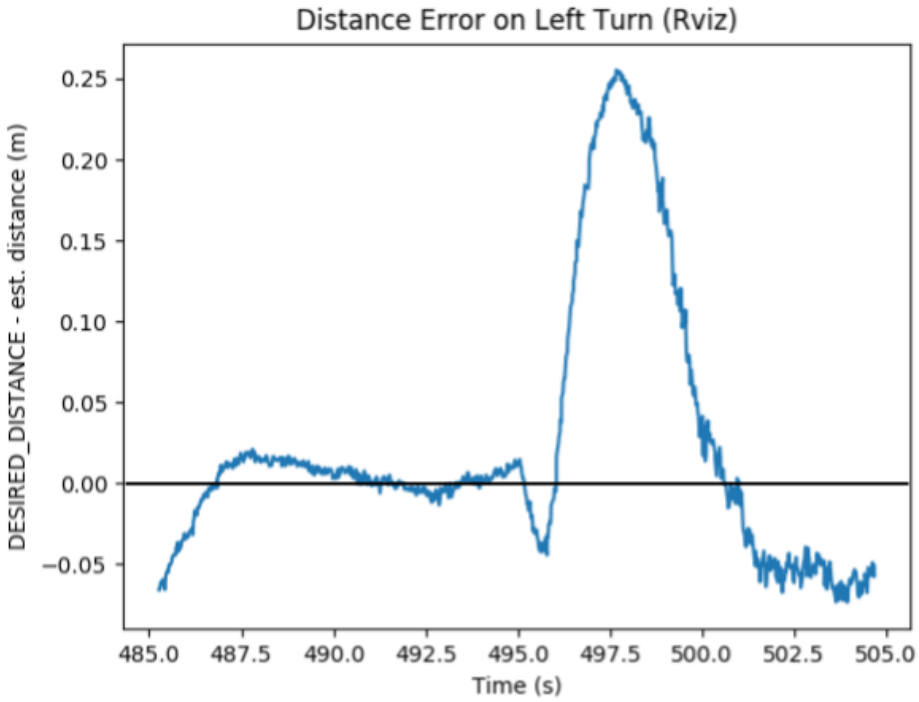

Figure 1: Left turn error in simulated environment

Our team also tested the performance of our wall following algorithm in a simulated environment, with RViz for visualization. Testing over simulation is highly accessible and allows for detailed visualizations of LIDAR data and wall regressions in real time against a known baseline. Our team tested our implementation of the wall follower in simulation using a range of different speeds and desired distances to maintain from the wall. Since these simulations can run on each team member's computer individually, simulated tests provide an opportunity

for parallelization that is much more time efficient than real world tests run on the car.

Figure 1 is a graph of the distance error as our car takes a turn in the simulation environment. As anticipated, the near-zero error takes a large jump as the corner is rounded, before falling back to smaller values as the car continues following a straight wall.

We were also able to utilize the visualizations in RViz in combination with our real world runs, as it allowed our team to verify that our algorithm's use of LIDAR data was functioning as we expected. In Figure 2, the points along a wall can be compared with the wall the algorithm has created using regression (red line). As expected, the regression-generated wall is straight and forms an excellent approximation for the actual wall.

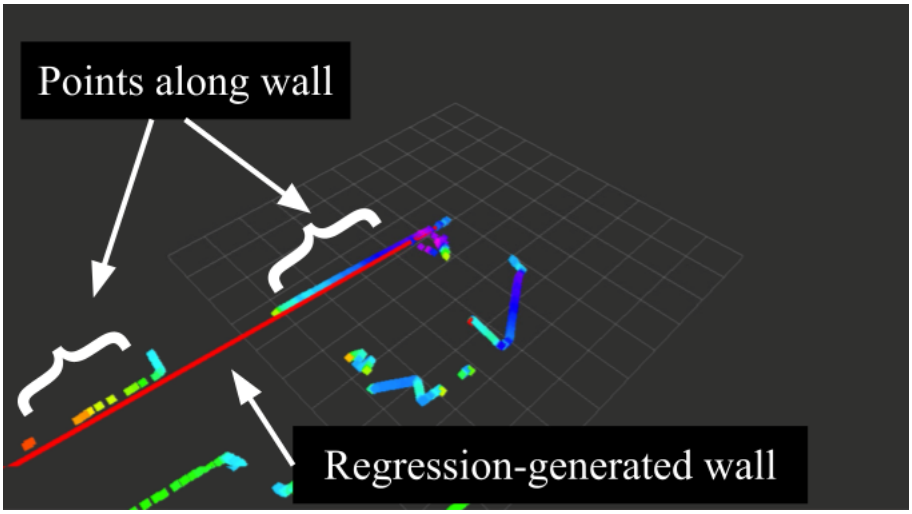

Figure 2: Verifying sensor data with RViz

Importantly, our team did notice that success in RViz was not necessarily a guarantee of success in the real world, even for similar wall structures and drive parameters. Good performance in RViz after making an algorithm adjustment was an indication that the same change was worth pursuing in a real world run, rather than a confirmation of better performance in itself.

3.3 Analytic Assessment

Our team recorded rosbag data during real world runs to gain quantitative insight into our wall follower's performance. Our team created a dedicated analysis topic for publishing distance error while the wall follower is running. This removed the need to record all published topics, most of which had no analytical value, during a run.

In Figure 3, we see a plot of distance error versus time over the duration of a short run of the car along a straight wall. The error values are small and consistent, which are both positive signs that are algorithm is working as expected and correcting deviations quickly.

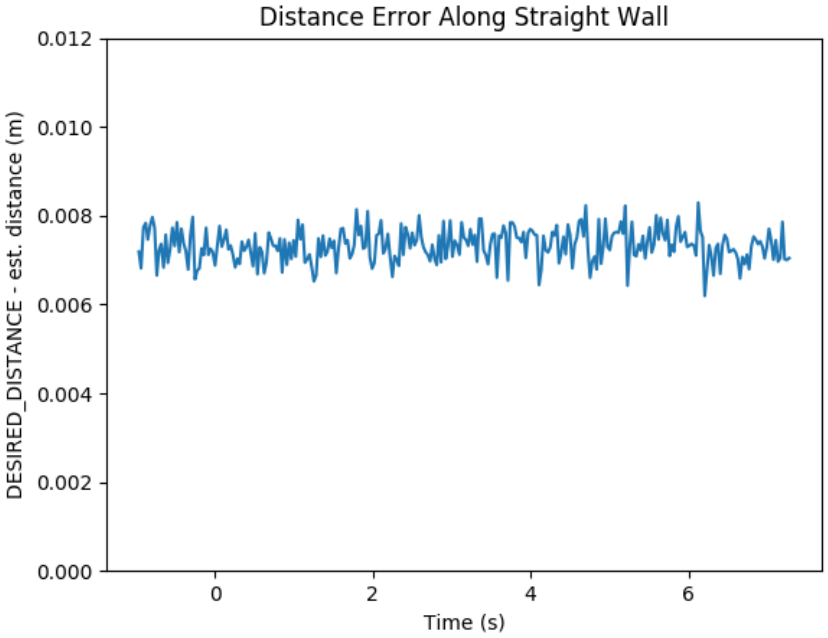

Figure 3: Distance error along a straight wall

These types of graphs allowed our team to accurately determine whether a small change to our algorithm was in fact helpful or not. Small increases in accuracy that are too difficult to discern visually can be seen clearly using rosbag data. An example of how are team used this data to improve our algorithm is seen in Figure 4, which graphs the distance error over time during a right turn. Our team was having difficulty determining at what point in the turn the car was

deviating from its expected distance from the wall. Analyzing the movement of the car second by second allowed us to identify that the issue involved a secondary turn after the main turn to be offsetting the car from its expected position.

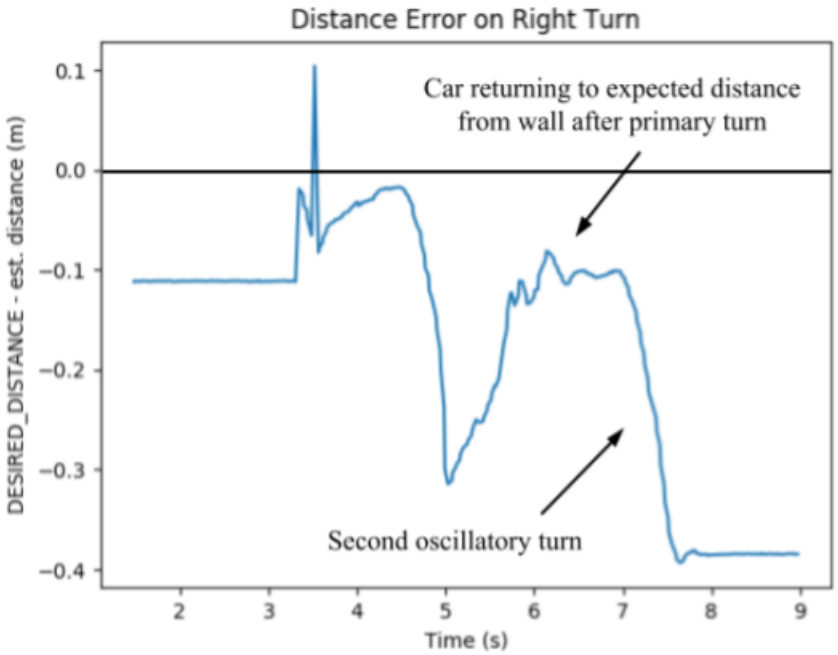

Figure 4: Identifying an algorithm issue using rosbag data

3.4 Future Evaluation Metrics

Our team discussed the possibility of including additional quantitative evaluation metrics that could be aggregated and plotted similarly to the distance error graphs above. Due to time constraints and the minimal benefit these quantities would afford for this lab, we decided not to implement them as of yet. We are noting them here to demonstrate the types of analytic measures we may potentially add to our assessment suite in the future.

- Generate velocity data from wheel rotations and turn angle to allow for publishing a velocity error metric.
- Generate a measure of how our turn handling aligns with what we'd expect for a given section of wall. This metric would be generated using the

r squared value from our regression and comparing it to the change in steering angle over time.

- Generate acceleration data to gauge the effectiveness of our safety controller. Acceleration could be easily determined once a velocity metric is in place.

4 Conclusion - Pranav

We were able to successfully develop a wall follower and safety controller for our autonomous robot. By determining relevant metrics that we would want to achieve and benchmarking our implementations against these metrics, we claim that our wall follower and safety controller are well-made to achieve their respective goals. The car is able to navigate arbitrary unfamiliar spaces at the commanded velocity and direction, and to avoid situations that could lead to a crash, either by turning out of the way or stopping.

Our next steps would be to continue verifying our implementation by stress testing it in difficult situations, such as tight turns in narrow hallways, wide spaces such that the car starts far away from a wall, and paths that have obstacles on the way. We would also like to perform more data analysis to further improve the robustness of our claim that the implementations are successful, as is outlined in Section 3.4.

To make a more optimal implementation, we would also like to perform a more formal parameter test that sweeps across the range of reasonable values for parameters like $K_p, K_d, c_p,$ and c_b. Optimally tuning these parameters would benefit us by potentially allowing for a shorter development cycle in our next project (line following), as our implementation for that project will also likely depend on an optimal controller.

5 Lessons Learned

Presents individually authored self-reflections on technical, communication, and collaboration lessons you have learned in the course of this lab.

5.1 Pranav

I really enjoyed collaborating with my other team members to take our varied approaches to implementing wall following in lab 2 and coming to a single, final implementation that would be as effective as possible in real life. Having open channels of communication facilitated this, as we were continuously discussing ideas and logistics to keep the process moving smoothly.

I also enjoyed editing the report at large. It was fun keeping the style consistent and making sure that we conveyed the information we wanted to as effectively as possible.

5.2 Thomas

Besides providing an excellent introduction to many aspects of ROS and our racecar hardware, Lab 3 was a learning experience in how our team worked together to deliver a result that none of us could have created on our own.

There are two main takeaways that I've gained from this first week of in-person meetings, virtual conversations, and collaborative documents. The first is that every minute the group can spend together is invaluable, so a clear plan should be made beforehand to maximize the productivity of that resource. Second is that dividing tasks into modular components is key to effectively managing time while keeping the results from different individuals organized.

Looking forward to refining these skills in future labs!

5.3 Brady

5.4 Calvin

I learned a lot about working both with the racecar, and as a team throughout the course of this lab. On the racecar side I learned a few key things to help streamline the process of testing our code; using ROS params was invaluable to help save us time tuning specific parameters that were necessary to make sure our robot worked sufficiently, and working through a shared github repository also saved a lot of time that would have had to be spent using scp to transfer files around through our computers. For working with my team I felt like by the end of the lab we had found our footing a bit better as far as being able to modularize tasks and I'm looking forward to bringing this knowledge to the next few labs.

5.5 Gustavo

Beyond gaining familiarity with the technical aspects of the wall follower algorithm, PID control, and the ROS development framework, this lab taught me invaluable lessons about how to collaborate effectively and ensure that a full technical deliverable was completed under a tight deadline.

As a team, we learned how to streamline our debugging process, taking advantage of tools like github, RVIZ, and rosparams. We also became familiar with each other's working styles and determined what were the best ways to communicate and delegate tasks.