Lab # 3 Report: Wall Following

Team # 17

Devin Mccabe
Anish Ravichandran
Arnold Su
Anirudh Valiveru

6.4200 - Robotics: Science and Systems

March 11, 2023

1 Introduction

Starting this lab, our team set out to accomplish a variety of goals. We successfully adapted our wall follower simulation code from Lab 2 and tuned it with PID control parameters based on real-world data. Additionally, we implemented a safety controller system to ensure the robot does not incur damage when travelling near other objects. Aside from the written instructions of developing a functional wall follower and safety controller, we wanted to tackle the other aspects of interacting with the robot and with a team. The robot has many working systems and parts, and to seal the gaps in our knowledge, we wanted our own documentation of the system, with how-to guides and troubleshooting FAQs. Finally, we aimed to learn how to work with each other as a team, playing to each other's strengths and dividing up the tasks to lead to our success in the long-term over the semester.

2 Technical Approach

2.1 Wall Follower

One of the primary goals of our wall following labs over the past two weeks was to design and implement a functional implementation of wall following that would work in both simulation and on an actual racecar.
Our wall follower must be able to move along a wall at a fixed velocity, side (left or right) and desired distance, specified as parameters to the program. Note that these restrictions should hold regardless of the shape of the wall; for

example, if the robot encounters a corner or an oddly-shaped section of the wall, it should adjust its motors accordingly to continue tracking the wall's shape.

2.1.1 Design Details

To solve the problem of designing a wall-following robot, we chose to implement PID control, which is a type of negative feedback system. PID feedback control was chosen because of the nature of the problem at hand; since the goal of the problem is to keep the robot at a fixed distance away from the wall without moving, it makes sense to correct the robot's position based on its error from the desired distance with the wall.

We determined the distance of the robot from the wall by using LIDAR scans collected from our car's Hokuyo sensor. The sensor reads the laser data on each data as a $sensor_msgs/LaserScan$ object, whose attributes include a range of angles, as well as a measured laser distance at each of these angles.

By default, the Hokuyo sensor provides us with the data ranges in a list, as well as the minimum angle, maximum angle, and the angle increment between consecutive ranges in the `ranges` array. As a result, we should be able to convert any point in `ranges` into polar coordinates, with the vehicle at the center, by simply interpolating the possible angle measures for each point over all indices. To allow for fast iteration, our team decided to include the start and end angles for a wall scan as parameters to the ROS node, since the slice of data that we use for our range critically affects our wall following behavior.

After getting a list of our desired wall points as polar coordinates with respect to the robot, we convert these polar coordinates into Cartesian coordinates to then find a least-squares best fit line that represents the wall. It is important to find such a best-fit line (instead of simply using one point to the robot's side) to compute the error because of the high probability of noise on any individual measurement. Based on this best-fit line, we defined the "distance to the wall" as the y-coordinate of the closest point on our regression line to the robot origin. Since this measurement is generally fairly close to the actual distance, it works as a good heuristic whenever the wall is parallel to the robot. When we want to turn, however, the computed distance is smaller than the actual distance on an inside corner, and larger than the actual distance on an outside corner, which allows our controller to turn more or less based on the observed wall topology.

Below is the block diagram describing PID control given a target, taken from slide 45 in 6.4200's control lecture presentation.

In the context of this problem, $y(t)$ is defined as the sensed distance to the wall (as computed through linear regression), $r(t)$ is the desired distance from the wall (as specified by a parameter), and $e(t)$ is the difference between the desired and actual distances, and $u(t)$ is the motor command determining the angle of

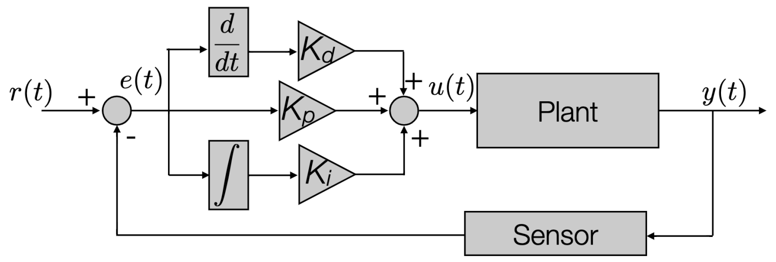


Figure 1: The PID Control Loop.

the front wheels when steering the robot. Note that a positive steering angle input to Ackermann steering corresponds to left steering, so we want to adjust the sign of $e(t)$ depending on the side that the robot should follow the wall on. Figure 2 displays a diagram of the wall following system.

While we did implement all three aspects of PID control in our code, much of our design process involved iteratively determining the optimal gains and angle range, which we delve into in section 3. At the end, we found that we could get a smooth and robust controller with a proportional gain of 1.5, and derivative/integral gains of 0, with an angle range from 30 to 50 degrees from the robot's orientation angle. Note that this implementation is equivalent to simple proportional control.

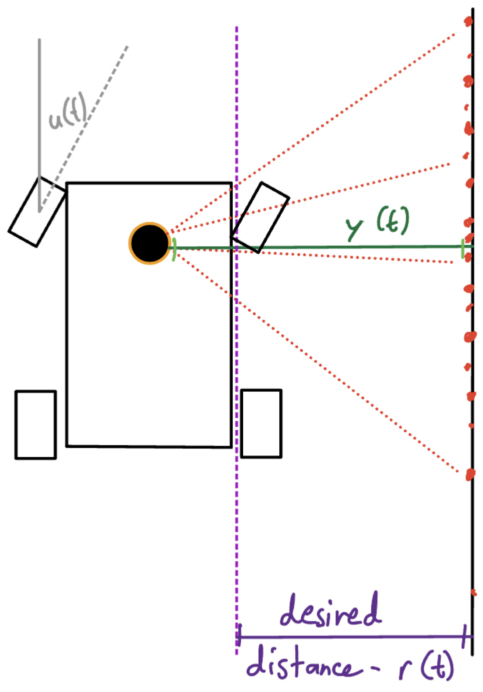
2.1.2 ROS Implementation

Since PID is a straightforward algorithm without many modules, we implemented it as a single ROS node located in the package's `src/wall_follower.py` file in Python. However, we specified several parameters for iteration, namely the side (left or right), desired distance, velocity, the three PID gains, and the start and end angle ranges for wall detection.

Scans from the Hokuyo sensor are read as LaserScan messages from the `'/scan'` topic, and motor commands are written to the `'/vesc/ackermann_cmd_mux/input/navigation'` topic, which is read by the command mux to be motor commands that are passed to the robot.

Lastly, we published the wall points to the '/wall' topic for the purposes of visualization of laser scans, in order to debug the wall detection code.

The final algorithm implemented for wall following is as shown:



$$e(t) = r(t) - y(t)$$

$$u(t) = K_p e(t) + K_d \frac{de}{dt} + K_i \int_0^t e(t')\, dt'$$

Figure 2: System Diagram.

2.2 Safety Controller

The goal of the safety controller is to prevent the car from colliding into objects while running programs. This is an important feature to have as the race car is both delicate and expensive, so there is great risk in operating the car autonomously where there is high chance of unexpected and unknown behavior. While there is already a ”dead man switch” in the form of the joy stick, having this safety controller in place adds an extra layer of protection and should be more reliable than a human operator.

The safety controller must also not cripple programs that we want to run. If the safety controller is too sensitive, the car is forced into deadlock and is prevented from moving. This introduces a design challenge in which the safety

```
Algorithm 1: PID Motor Controller
  Data: scan, desired_velocity (parameter), desired_distance
        (parameter), angle_start (param), angle_end (param), KP
        (param), KD (param), KI (param)
  while True do
      Split scans into left and right wall ranges based on angle_start and
       angle_end
      Convert polar coordinates to Cartesian Coordinates (w.r.t. robot)
      Compute least squares regression to get wall points.
      Find closest point on line to origin
      curr_distance ← y-coord of closest point
      error ← desired_distance − curr_distance
      integral ← integral + error ∗ loop_frequency
      derivative ← (error − prev_error)/loop_frequency
      $u$ ← KP ∗ error + KI ∗ integral + KD ∗ derivative
      Input $u$ as the input, desired_velocity into Ackermann steering.
      prev_error = error
  end
```

controller must be sensitive enough to prevent collisions, but also not overly sensitive where the car becomes too "scared" to function.

2.2.1 Initial Set-up

The race-car has a built-in command mux with different levels of priority shown in Figure 3. This means that when the race-car receives commands from different topics, it will only execute the topic being published to the highest priority topic. For the purposes of this lab, there are four topics to consider:

- "/vesc/ackermann_cmd_mux/input/navigation"
- "/vesc/high_level/ackermann_cmd_mux/output"
- "/vesc/low_level/ackermann_cmd_mux/input/navigation"
- "/vesc/low_level/ackermann_cmd_mux/input/safety"

which we will call "/navigation", "/output", "/low_level_navigation", and "/safety" for brevity. For the purpose of this lab our programs will be published to "/navigation". "/navigation" is piped into "/output", and "/output" is piped into "/low_level_navigation". "/low_level_navigation" and "/safety" are both published to the same mux, where "/safety" takes priority over "/navigation".

As usual, LiDAR scan data will be published to "/scan".

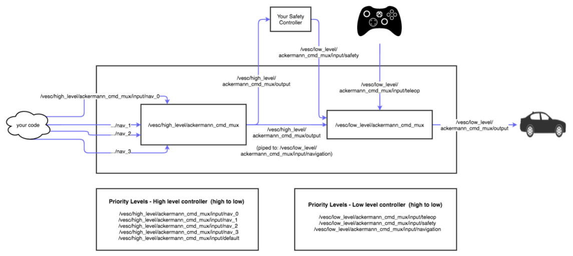

Your Safety Controller
/vesc/low_level/ ackermann_cmd_mux/input/safety
/vesc/high_level/ ackermann_cmd_mux/output
/vesc/low_level/ ackermann_cmd_mux/input/teleop
/vesc/high_level/ackermann_cmd_mux/input/nav_0
your code
.../nav_1
.../nav_2
.../nav_3
/vesc/high_level/ackermann_cmd_mux
/vesc/high_level/ 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 3: The Car Command Mux, borrowed from mit-rss.

2.2.2 Technical Approach

In designing the safety controller, we decided that using a "buffer zone" is a simple yet effective approach. The idea is to have an imaginary "buffer line" based around the car. If any object is closer to the car than this line, then the safety controller will activate and force the car to stop moving.

First, there were two choices for the shape of the line, curved or straight. We decided using a straight line was the better approach because the car itself is box-shaped, so a straight line will more closely mirror the car itself. While a curved line more easily scales around more than one side of the car, it forces the controller to be overly sensitive to properly cover the entirety of the car, since the car is not a single point.

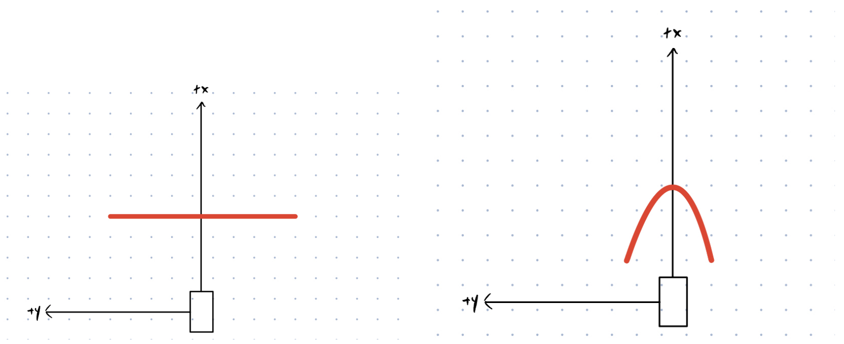


Figure 4: Different versions of buffer line.

Given our line will be a straight line, the next step was to determine the dimensions of the buffer line. In other words, deciding how far away from the car the line will be, which we will call l, and the length of the line, which we will call b. The level of risk/potential damage increases as the speed of the car increases. Therefore, the safety controller should be more sensitive the faster the car moves. And so the distance between the line and the car will be a linear function increasing with the car's speed. Let v be the car's speed in m/s. So $l = s \cdot v$ where s is some constant. s effectively controls the sensitivity of the controller, so s will be parameterizable to control sensitivity. Next, we decided to make b constant and parameterizable. Then we can adjust b so that it covers the entire breadth of the car, while not being too sensitive.

Finally, we decided which direction(s) the line(s) will lie on relative to the car. We decided to place a single buffer line in the direction of the car's velocity. In other words, if the car's steering angle is θ, then the center of the line will be d away from $(0, 0)$ in direction θ. This minimizes the necessary sensitivity of our safety controller because it will only be concerned about objects in the direction of movement. For example, if the car is moving forward, it will only be concerned if it approaches an object from the front. It also makes the safety controller more robust to sharp turns. The full design is shown in Figure 5. Notice this approach does make the car vulnerable external objects moving into the car from an angle that is not the general direction of movement. However, this should be little concern this kind of event should be rare and easily avoidable when our car is operating.

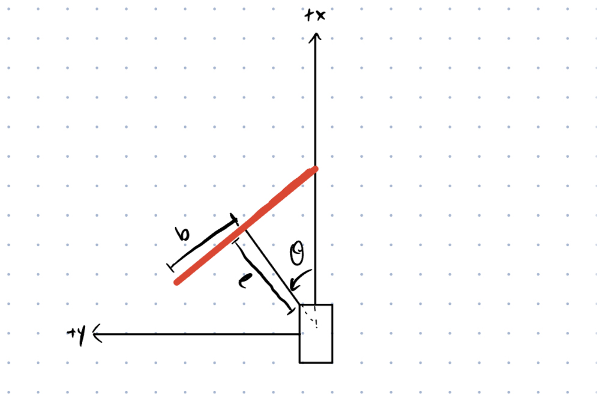


Figure 5: The safety controller design.

2.2.3 ROS Implementation

Given our set up, we first determined which topics our safety controller should listen and/or publish to. We want our safety controller to stop any programs that may cause the car to crash, so "/safety" was the clear choice for publishing commands to override any dangerous commands. Because we want our safety controller to be sensitive to the cars speed and turning, the safety controller is subscribed to "/output" to pick up the navigation-related commands are being sent to the car. Additionally, the controller needs to know how far away it is from object. Therefore, it will be listening to the LiDAR scan data from "/scan". A simplified rqt graph of the controller is shown in Figure 6

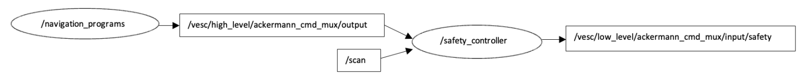

/navigation_programs
/vesc/high_level/ackermann_cmd_mux/output
/scan
/safety_controller
/vesc/low_level/ackermann_cmd_mux/input/safety

Figure 6: RQT Graph of Safety Controller Connections.

Note that scan data is parameterized by angle, so producing our buffer line will have to be iterative over angles. Let us develop a simple algorithm is going to iterate through each of the scan angles α, and calculate a threshold distance d as a function of α. If the range of the scan at α is less than d, then an object has breached the buffer zone and the safety controller publishes to stop the car.

Algorithm 2: Safety Controller (version 1)

```
$\alpha \leftarrow scan.angle\_min$;
while $\alpha \leq scan.angle\_max$ do
    $d \leftarrow threshold\_dist(\alpha)$
    if $scan.ranges(\alpha) \leq d$ then
        $publish$;
    else
        $continue$;
    end
    $\alpha \leftarrow \alpha + scan.angle\_increment$
end
```

To calculate $d = threshold\_dist(\alpha)$, let's start with the forward case ($\theta = 0$). We want our threshold distances to form a straight horizontal line such that is is $l = sv$ meters away from the car, and has length b. Thus, let the buffer line be defined as a line segment with the points $\{(l, -b),\ (l, b)\}$ in (x, y) coordinates. Our goal is to convert points along this line into polar coordinates (d, α) so relate threshold distance d and scan angle α. We know that $\forall \alpha$, $x = d \cdot cos(\alpha)$. Because we want $x = l$, $d = \frac{l}{cos(\alpha)}$. We don't want an infinitely long line, so we must bound α. The line segment $\{(0, 0), (l, b)\}$ is the line from the car towards the left most point on our buffer line. This line occurs at $\alpha_b = tan^{-1}(b/l)$. By symmetry, we know that at $\alpha = -\alpha_b$, a line will take us to the right most point $(l, -b)$ on our buffer line. And so we can define our function as:

$$desired\_dist(\alpha, l, b) = \begin{cases} \frac{l}{cos(\alpha)} & \alpha \in [-\alpha_b, \alpha_b] \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

However we want our buffer line to also shift according to our steering angle θ. Since we have already calculated our line in terms of angles α, all we need now is to shift our function by θ to rotate the line. Let dd_1 be the function defined above. And so *desired_dist* is actually a function of both α and θ,

$$desired\_dist(\alpha, \theta, l, b) = dd_1(\alpha - \theta, l, b).$$

Our final algorithm is as follows:

Algorithm 3: Safety Controller (final version)

```
Data: $scan$, $v$ (velocity), $\theta$ (steering angle), $s$ (param), $b$ (param)
$l \leftarrow sv$;
$\alpha_b \leftarrow tan^{-1}(b/l)$;
$\alpha \leftarrow scan.angle\_min$;
while $\alpha \leq scan.angle\_max$ do
    $d \leftarrow threshold\_dist(\alpha, \theta, l, b)$;
    if $scan.ranges(\alpha) \leq d$ then
        $publish$;
    else
        $continue$;
    end
    $\alpha \leftarrow \alpha + scan.angle\_increment$
end
```

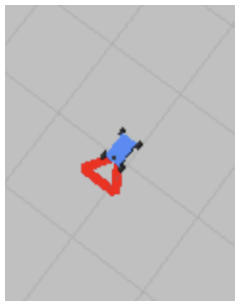


Figure 7: RViz Simulation showing the buffer zone.

3 Experimental Evaluation

We evaluated our wall follower controller's ability to reach and maintain a desired distance from a wall while operating under a range of velocities, starting angles, and desired distances. In evaluating the wall follower controller, the error, defined as the desired distance minus the actual distance to the wall, was the primary data point we tracked to determine the performance. From graphing the error with respect to time, we could determine the settling time, relative overshoot, and steady-state error of our controller. These three metrics, all obtainable from the error vs. time graph, determine the necessary gain values for our controller.

3.1 Impact of Velocity on Wall Follower Controller

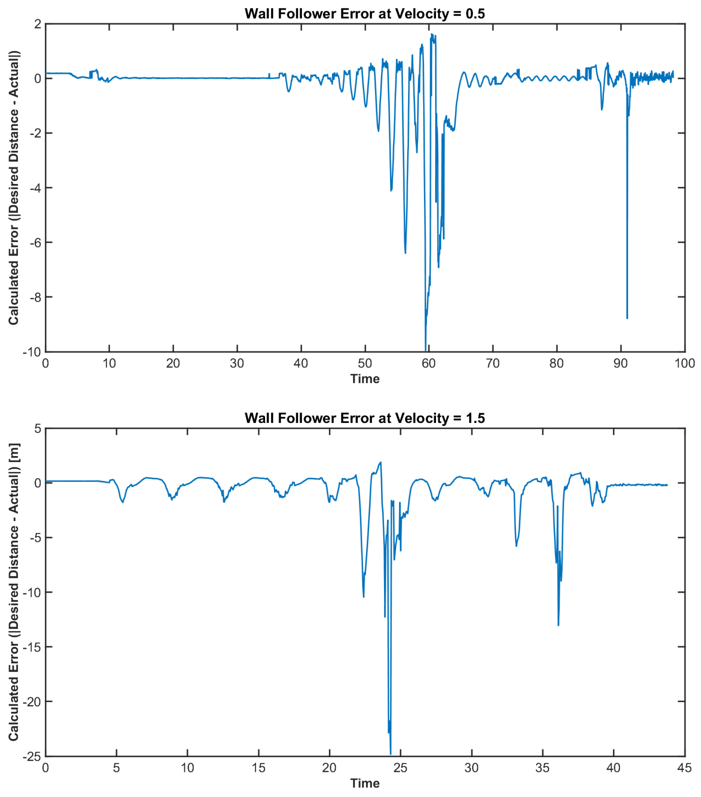


Figure 8: Measured error at a velocity of 0.5 and a velocity of 1.5 plotted with time. For each test, velocity was varied while the desired distance from the wall remained a constant 1 meter and a consistent starting angle of approximately 30 degrees with the wall was maintained.

As figure 8 displays, with greater set velocity values, the car exhibited less favorable driving characteristics as it had both greater overshoot and a longer settling time. The error of the controller when driving at a car velocity of 0.5 stayed less than a quarter of a meter. The car also settled into a straight driving line within seconds. The car had minimal steering angle adjustment after approximately 5 seconds of driving. At higher velocities, the car did not settle before it reached the end of the testing wall. When the velocity was set to 1.5, the settling time was greater than 20 seconds, over four times the settling time of the 0.5 velocity test. The amplitude of oscillations of error was similarly increased from lower velocity tests, with the magnitude of error increasing to greater than a meter.

While the car did not crash at all at higher velocities, the trend of increasing overshoot with velocity suggests our controller is best tuned for lower velocities. The controller had a very quick and efficient response for the lower velocities, but it showed room for improvement. In section 5, we discuss the adjustments and iterations we went through to improve our controller.

3.2 Impact of Starting Angle on Wall Follower Controller

In assess the effectiveness of our wall follower controller, we tested how our controller reacted to a full range of starting angles. We maintained the same desired distance of the previous set of tests of 1 meter and had a velocity of 1 for all tests. The range of angles covered the entire span of facing the wall to facing directly away from the wall. We considered a rotation of the clockwise from parallel with the wall as a negative change in angle and counterclockwise positive.

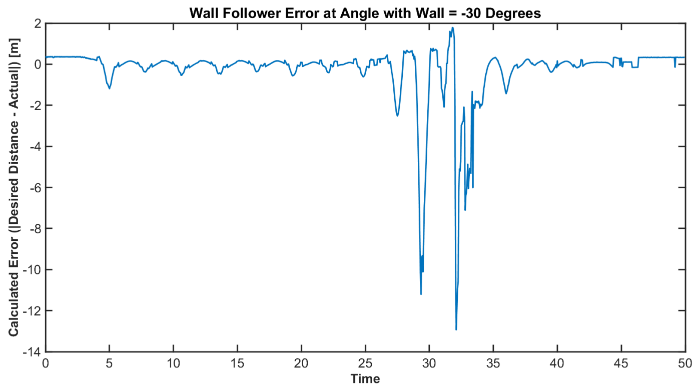
Figure 9 displays a comparison of the wall follower controller's response to a positive and negative starting angle. The controller did not exhibit any change in settling time or overshoot. The initial error value differed between the two due to the function that our controller uses to evaluate its "distance" to the wall, but this did not have any negative impact on the effectiveness of our controller's response.

In figure 10, the controller's response to a starting angle of 90 degrees displays its effectiveness at any starting angle. When the car was facing directly away from the wall, the controller still found the wall and then gave corresponding steering inputs in order to direct the car to the line at the desired distance from the wall. The response worked while facing the wall directly as well, having a similar response with similar peak error magnitudes and settling times.

3.3 Impact of Desired Distance on Wall Follower Controller

To evaluate the wall follower controller on its ability to maintain a variety of desired distances, we varied the desired distance parameter settings from 0.5 meters to 2.5 meters. The 2.5 meters was the upper end of the range as the hallways we were working in were little wider than this distance.

Figure 11 displays the trend of increasing overshoot for larger desired distances. With a desired distance of 0.5 meters, the controller had a settling time in terms of steering input of approximately 5 seconds. The distance of 2.5 did not fully



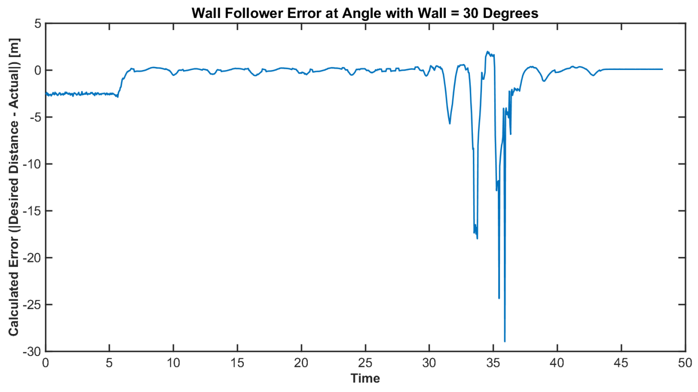


Figure 9: Measured error at both negative 30 and positive 30-degree starting angles. The desired distance for each test was 1 meter and the velocity parameter value was 1.

settle before it began to pick up the end of the wall and the upcoming turn. The increased distance caused our controller to have the error to exhibit a sinusoidal structure with a slowly decreasing magnitude, before increasing as the end of the wall increased the perceived distance to the wall. This set of tests varying the desired distance showed that our controller maintained functionality at large desired distances as well as smaller desired distances. The controller could be better tuned for greater distances in future iterations.

3.4 Evaluation of the Safety Controller

For the implementation of the safety controller, we crafted our test suite with the following parameters in mind: car velocity, turning angle while stopping, types of obstacles, and whether the car was traversing a concave (inner) or convex (outer) corner. When considering car velocity, we hypothesized that a

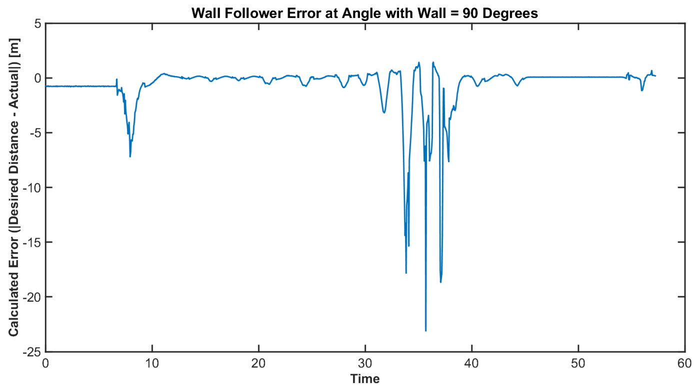
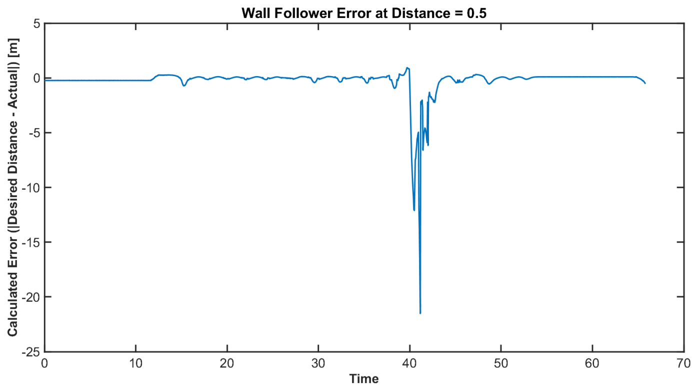


Figure 10: Measured error at a 90-degree starting angle. The desired distance for each test was 1 meter and the velocity parameter value was 1.

larger velocity would not have an effect on whether the car safely stopped in the presence of an obstacle, since our implementation would increase the size of the buffer zone when the car is moving faster. We hoped that turning angle and the type of corner would not affect how well the car stopped, and had no real idea whether the type of obstacle would produce a salient difference in performance.

Test	Velocity	Angle	Object	Test Type	Result
1	0.5	0.0	Opaque Box	Velocity Test	Pass
2	1.0	0.0	Opaque Box	Velocity Test	Pass
3	1.5	0.0	Opaque Box	Velocity Test	Pass
4	2.0	0.0	Opaque Box	Velocity Test	Fail
5	3.0	0.0	Opaque Box	Velocity Test	Pass
6	1.0	-30.0	Wall	Turning Test	Pass
7	1.0	-60.0	Wall	Turning Test	Pass
8	1.0	-90.0	Wall	Turning Test	Fail
9	1.0	0.0	Cone	Obstacle Test	Fail
10	1.0	0.0	Plastic Box	Obstacle Test	Fail
11	1.0	0.0	Reflective Wall	Obstacle Test	Pass
12	-1.0	0.0	Opaque Box	Backwards Test	Fail

The test results paint a bright yet complex picture. Passing a majority of the tests in the suite, it is easy to see how well the safety controller works in a real-world environment such as the Stata basement. However, at certain velocities, the car finds trouble stopping in time to prevent a collision. Test 4 is noteworthy because it seems anomalous - both tests 3 and 5, at lower and higher velocities, clearly pass, while an intermediate velocity fails. This demonstrates the nonlinearity of our safety controller system and showed us that we needed



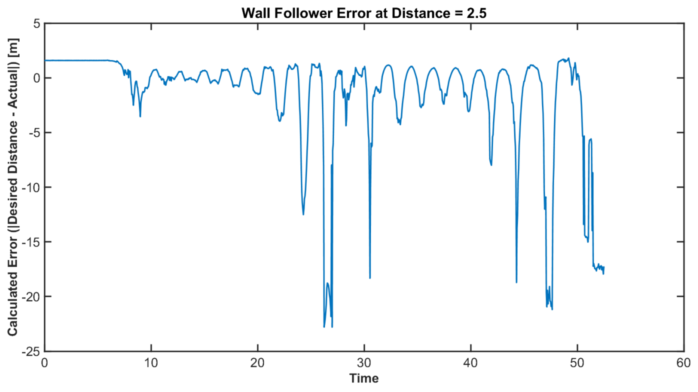


Figure 11: Measured error at desired distance values of 0.5 meters to 2.5 meters. The starting angle for each test was negative 30 degrees and the velocity parameter value was 1.

to further tune it. It is also worth noting that our criteria to pass a test were very strict. The car could not touch the object, even at very low speeds. In a real scenario, we can imagine that the car would not be harmed from a low velocity collision with an object, but for our purposes, this would classify as a failed test case.
For many non-traditional objects, such as those that are conical or translucent, we can see that the safety controller can benefit as well. Additionally, in extreme scenarios such as encountering an object while turning 90 degrees or driving backwards, the car fails to pass the test due to overengagement of the safety controller. Nevertheless, these are extreme cases that will rarely be encountered in the field, and therefore they are not a cause for concern.

4 Controller Imporvement and Iteration

Our tests showed that the controller we implemented was effective at following a wall and turning corners at all speeds and distances tested, however, the response of the controller exhibited some overshoot and settling time issues at the highest velocities and largest desired distances. The controller we were using was a proportional controller with a proportional gain of 5 and no derivative component. This proportional gain had the potential to be lowered to reduce overshoot and improve the performance of the controller. We varied the gain in a series of tests to determine the best gain for our application.

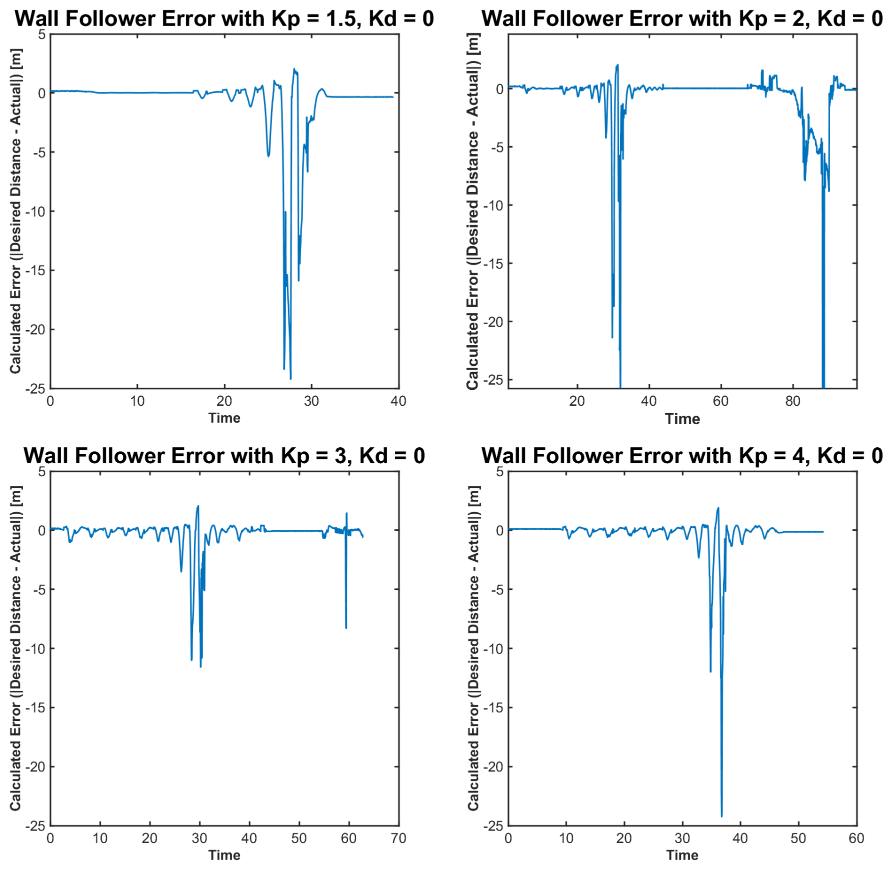


Figure 12: Measured error while the proportional gain of our controller was varied from 1.5 to 4. The starting angle for each test was negative 30 degrees, the velocity parameter value was 1, and desired distance was 1 meter.

We found that a proportional gain of 1.5 minimized the settling time and overshoot while still maintaining the desired distance of the car from the wall. Higher

Kp values created overshoot and increased the settling time, producing a sinusoidal response structure. We attempted to add a derivative term to our controller to reduce the oscillation magnitude, but the additional steady-state error and slow response time caused it to not turn corners well for the derivative terms that we tested.

5 Conclusion

Our team's primary takeaways from the wall-following lab lie in learning the high-level skills required to succeed in the rest of the class. Since the rest of the labs moving forward will be team-based while also requiring the car, much of the week was spent fine-tuning the systems that our team used for collaboration and task-management. One lesson that we learned in particular revolved around sharing the car with other teams who may require the same resources as we do; since it will be important for us to work with our pod-mates for the rest of the semester, making sure that we are highly communicative with each other about resource and time-management is critical for ensuring success for everyone.

We also determined that extensive documentation of anything we do on the robot, whether an experiment or an integration hurdle, would be crucial part of setting ourselves up for success over this week. For one, we decided early-on to utilize Notion as our task-management system, serving as a central hub for all important links, deliverable drafts, tasks, and documented issues to refer to later. While we did not end up referring back to any of the issues that we had run into, we foresee this being an important step in streamlining our setup in future labs.

Aside from logistics, both wall-following labs over the past two weeks served as a good exercise to understanding how to use ROS in a real application. While we had done the ROS intro exercises before, implementing a wall-following system in simulation before porting the code and tuning to an actual robot was a good way to increase our intuition about the ROS workflow. This includes visualizing the current state of the robot, using launch files to simplify development, and understanding the differences between the '/drive' topic for simulation and the VESC command mux.

Last, but not least, we walked through the process of implementing a low-level robot controller from scratch, learning the basic foundations of control theory in a hands-on foundation. While the final implementation was essentially some form of proportional control, implementing PID and trying various different gains was a good way to gain an intuition for how the different gains interact with each other.

There are also several future steps that we could take to improve the robustness of our design. For one, using some set of points directly in front of the robot to detect an inside corner may allow us to turn smoother by implementing a state-machine approach. We may also implement some roaming functionality for the robot to find a wall if there are none its vicinity. The last thing that we could try is some dynamic gain management to change gains based on aspects like speed or distance from the wall.

6 Lessons Learned

Anirudh Valiveru - Since the primary deliverable of this lab are the lab report and briefing, as opposed to passing some test cases, many of the lessons that I learned stemmed from the unique approach that our team had to take to meet these deliverables. For example, our team had spent a considerable amount of time discussing and determining the best data to collect in order to communicate our system in the best way. This goes hand-in-hand with the second important lesson that I learned about project management. Since this lab required a significant amount of work from all of us over a whole week, communicating our conflicts and splitting tasks according to our strengths and availability was crucial to our lab's outcome. In this vein, our team set up a Notion page to manage all of our collected information, experiments, and tasks, which was a great call to set up a solid system for collaboration in future labs.

Anish Ravichandran - A key aspect of working in a lab group is learning to fill different roles. I found that having multiple people do the same thing, such as writing code, performing tests, or setting up the car, can lead to inefficiencies because there are too many "chefs in the kitchen". Delegating roles early on and making decisions playing to each member's strengths was a lesson learned this time around. Another lesson, which is the key component to any class at this school, was time management, where we had to share our car with other teams, find times to work on the lab together, and overall stay efficient when working. We learned how to document issues so that they aren't repeated and negotiate timing with other groups, which are essential skills as we move on to future labs.

Arnold Su - In a technical context, through this lab I learned that there was a lot more "uncertainty" and raw experimentation than I was used to. As someone whose technical experience is almost entirely done on just the computer, I realized that computer simulations and tests were not very accurate when it is actuated in real life. Additionally, the idea of evaluation in real-life was relatively new, as most of my evaluations came in the form of coding test suites. From a communication aspect, I found that knowing we had to communicate something gave more purpose to our testing. It forced me to think about exactly what kind of data we want to collect, and how we should go about analyzing it for our deliverables. Finally, for collaboration I learned that it was hugely beneficial to sit down, slow down, and organize our respective tasks and responsibilities, as well as organizing time together. When each of our priorities/expectations are made clear, I think it eliminates lingering questions like ”what should I do now?”, as well easily splitting up the work so that it is balanced.

Devin McCabe - Through this lab, I learned how to more effectively organize myself and my team to achieve our goals and meet deliverables. We created a centralized site that broke down all important tasks for the lab and served as a hub for any information we might want to share with the group. We could then more effectively create task lists that kept our sessions focused and efficient. Our test planning and brainstorming took place on this shared site as well. We could keep track of the overall goals of the tests as well as all the small details like specific parameter values, names of recorded topics, and filenames of the raw data. One of the most important additions was an error tracker. We could create a centralized database to document errors that occurred. This will be instrumental down the line in remaining efficient as we work through more difficult labs and cannot afford to run into as many small errors that cost too much time. Organization is key and I have learned it will be absolutely essential to our success.