6.4200/16.405 SP23 - Lab #3 Report: Implementing an Autonomous Wall-Following Vehicle with a Safety Controller

Team #20

Aaron Zhu
Jessica Rutledge
Chuyue Tang
Nihal Simha
Shara Bhuiyan

6.4200/16.405 Robotics: Science and Systems (RSS)

March 11, 2023

1 Introduction (Aaron / edited by Jess)

Wall following presents a straightforward method for an autonomous race car system design, where the vehicle traces the contour of a nearby wall or object at a predetermined distance. Previously in RSS (MIT's "6.4200/16.405 Robotics: Science and Systems" course), we were tasked with developing an algorithm for an autonomous vehicle that could follow a wall in a simulated environment. Now, we move away from simulations to work on a real-life race car, modifying our simulated wall-following code and implementing safety precautions to eliminate the danger of crashing.

Adapting a wall-following algorithm from a simulation to a physical car and implementing a robust safety controller presents a substantial challenge. Transferring the simulation onto the vehicle requires comprehensive knowledge of the hardware systems provided. Additionally, thorough tuning is required to ensure the desired results from the simulation are replicated in the real world. Finally, constructing the wall follower in the real world can result in more severe implications such as collisions (unlike simply passing through the wall as in the simulation). This emphasizes the necessity for rigorous fine tuning to ensure a robust wall-follower. The primary difficulty for the safety controller was balancing between the imperative of protecting the vehicle's hardware, while ensuring the vehicle's desired functions were not impeded.

In this lab, we build upon our previous lab code to present a working, autonomous vehicle system. Given a wall-like object, the car uses LiDAR laser scan data to move parallel to it at a set distance while autonomously executing appropriate safety precautions. Overall, we demonstrate, through experimentation, that our algorithm robustly operates in a variety of real-world scenarios.

The contents of this paper are:

- A comprehensive description of our system design, including our wall follower and safety controller
- An experimental analysis of the measures used to determine the success of our system.

We detail our technical implementations of the lab in Section II and evaluate our experimental results in Section III. Our conclusions are presented in Section IV and our personal lessons learned are described in Section V. Videos and images associated with this lab are available at: `https://drive.google.com/drive/folders/1IMhf-vWsCvO8wsje-Tpm68kaOgo3Tk51?usp=share_link`.

2 Technical Approach

This section details the design of the wall follower and safety controller.

2.1 Wall Follower (Nihal and Chuyue / edited by Shara)

Initially, a code review was conducted where each team member's wall follower code was evaluated to understand the different techniques that were used to obtain a solution. The resulting work is a combination of different code, rigorously tested both in simulation and on the actual car, to obtain the most accurate wall-following controller.

The goal of the wall follower is to obtain Light Detection and Ranging (LiDAR) measurements and produce a steering angle to drive the car while maintaining a desired distance from the wall. This was achieved through the implementation of a Proportional-Integral-Derivative (PID) controller, which is visualized in Figure 1. The desired distance d_r is specified to the controller. The race car moves, producing a change in the distance to the wall and a new LiDAR measurement L, which is processed by the sensor block to obtain a measurement of the distance from the wall y. This is then subtracted from d_r to obtain an error e which is processed by the PID controller block to produce a steering angle δ_s that the car must drive to follow the wall. Sometimes, this value can be larger than the achievable steering angle and the saturation block is used to limit this angle.

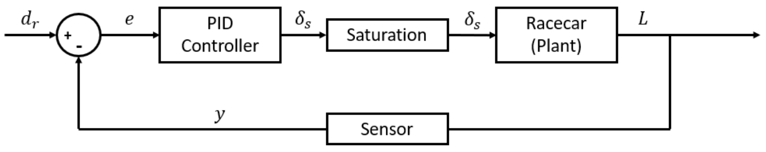


Figure 1: Wall follower control system. A PID controller is implemented by obtaining LiDAR measurements from the car.

To implement the aforementioned control system, the code must be laid out as in Figure 2. The Robot Operating System (ROS) uses a system of publishers and subscribers to send messages between the written code and the race car itself. Therefore, to obtain LiDAR data from the car, a subscriber was used to obtain data from the `/scan` topic which provides `LaserScan` type messages from the Hokuyo LiDAR. It contains important data about the laser scan such as the minimum scan angle value θ_{min}, the angle increment $\Delta\theta$, and the measured distance for each angle starting from minimum angle to maximum angle in steps of the angle increment. For our car, the data had a minimum scan angle of $-135°$ and a maximum scan angle of $135°$ with the angle increment being chosen such that the `ranges` list of measured distances is of length 1082.

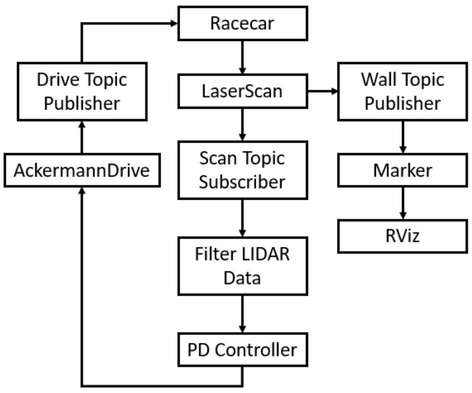


Figure 2: Code layout. This represents the software architecture on the car which represents the relationship between subscribers and publishers to do certain tasks.

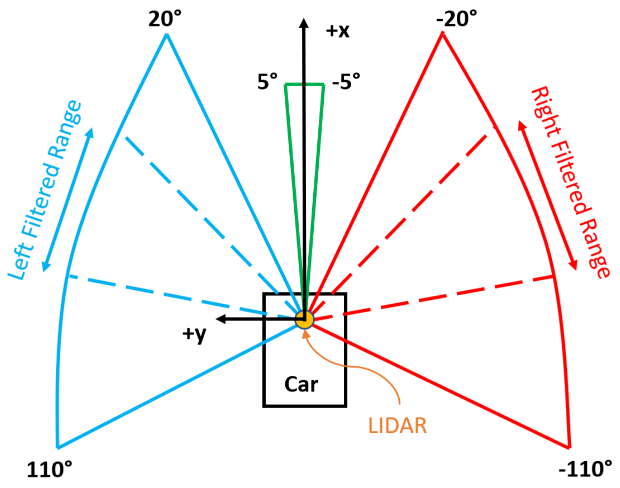


Figure 3: Laser scan lookout range selection. Since distances are measured from the LiDAR, the LiDAR is set to be the origin of the car's coordinate system. The positive x-axis points in the car's forwards direction, and the positive y-axis points to the left side of the car.

Algorithm 1 Filtering Sensor Data

```
Require: DEG2RAD, ROUND, ARRANGE, COS, SIN, MEAN, ABS, POLYFIT, STD, ENUMERATE
 1: procedure FILTER-SENSOR-DATA($scan$, $side$)
 2:     $\Delta\theta \leftarrow scan.angle\_increment$
 3:     $\theta_{min} \leftarrow scan.angle\_min$
 4:     if $side = -1$ then                                  ▷ When following the right wall
 5:         $b_{lower}, b_{upper} \leftarrow [-100, -20]$
 6:     else if $side = 1$ then                              ▷ When following the left wall
 7:         $b_{lower}, b_{upper} \leftarrow [20, 100]$
 8:     $i_{lower} \leftarrow \text{INT}((\text{DEG2RAD}(b_{lower}) - \theta_{min})/\Delta\theta)$
 9:     $i_{upper} \leftarrow \text{INT}((\text{DEG2RAD}(b_{upper}) - \theta_{min}/\Delta\theta)$
10:     $values \leftarrow scan.ranges[i_{lower} : i_{upper}]$
11:     $index_{side} \leftarrow \text{ARANGE}(i_{lower}, i_{upper}, 1)$
12:     $x \leftarrow values * \text{COS}(\theta_{min} + index_{side} * \Delta\theta)$
13:     $y \leftarrow values * \text{SIN}(\theta_{min} + index_{side} * \Delta\theta)$
14:     $sdev_x,\ sdev_y \leftarrow \text{STD}(x), \text{STD}(y)$
15:     $mean_x,\ mean_y \leftarrow \text{MEAN}(x), \text{MEAN}(y)$
16:     for $index_y, val$ in ENUMERATE($y$) do
17:         if $\text{ABS}(val - mean_y) < 1.7 * sdev_y$ and $\text{ABS}(x[index_y] - mean_x) < 1.7 * sdev_x$
    then
18:             $filtered_x \leftarrow x[index_y]$
19:             $filtered_y \leftarrow y[val]$
20:     $m, c \leftarrow \text{POLYFIT}(filtered_x, filtered_y, 1)$          ▷ fitting a line using filtered values
21:     $d_{wall} \leftarrow \text{ABS}(c/(m^2 + 1))$
22:     $i_{lower\text{-}front} \leftarrow \text{INT}((\text{DEG2RAD}(b_{lower}) - \theta_{min})/\Delta\theta)$
23:     $i_{upper\text{-}front} \leftarrow \text{INT}((\text{DEG2RAD}(b_{upper}) - \theta_{min}/\Delta\theta)$
24:     $d_{front} \leftarrow \text{MEAN}(scan.ranges[i_{lower\text{-}front} : i_{upper\text{-}front}])$
25:     return $d_{wall}$, $d_{front}$
```

2.1.1 Sensing and Filtering

This `LaserScan` data is then used to obtain a measurement of the distance from the side and front wall using Algorithm 1. Using RViz ("ROS Visualization"), we can visualize the LiDAR scan from the car as shown in Figure 4. Initially, the system looks at whether it needs to follow the left or the right wall. This determines the lookout range, and the system selects the corresponding values of the measured distances from the LiDAR data in the ranges array. Figure 3 shows a representation of the lookout ranges being used in the code. Ranges of 20° to 100° are chosen for the left wall and −20° to −100° for the right wall, allowing the vehicle to focus on objects directly to its side but also look more ahead to prepare the car for future maneuvers. This can be converted into upper and lower bound indices in the ranges list using the equations in lines 8 and 9. However, this data is in polar coordinates and in order to measure the distance from the wall, these points must be converted into cartesian coordinates with respect to the LiDAR reference frame. This can be done by a simple transformation of coordinates given in lines 12 and 13.

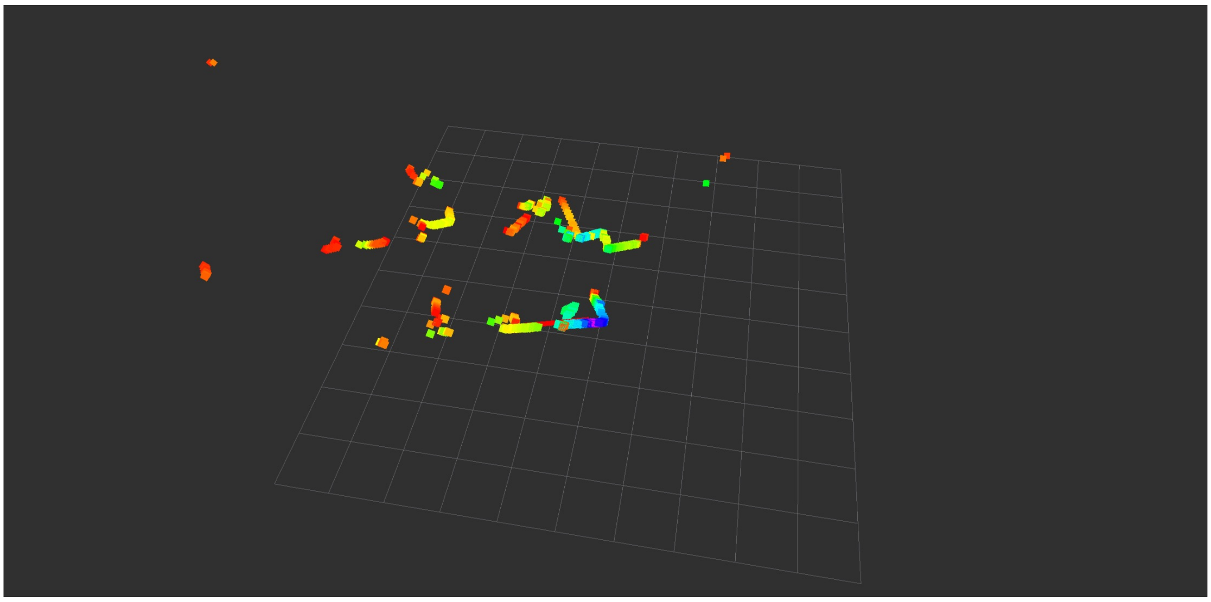


Figure 4: Screenshot of RViz visualizing the LaserScan data from the LiDAR. LiDAR data provides distances of objects in the environment relative to the LiDAR, which helps inform our autonomous vehicle system of how to drive.

Next, the standard deviation and mean of the x and y cartesian coordinate data are obtained after which these values are filtered if the difference between the x or y data point and its mean is greater than some constant times its standard deviation. This constant was found by tuning during experimentation. This allows for noisy data, that comes from the measured LiDAR distance being too large, to be removed providing better results. Initially, this extra filter was not used because it seemed redundant and the car was passing all its test cases. But when it came to turning an open corner, we were quick to realize (via RViz) that the marker being plotted produced a line that was not actually at the wall. Using this filter fixed the issue by removing data points that are obtained from far away points by only focusing on data in close range. A linear regression was then run to obtain a line of best fit through rough data as the wall being measured might be uneven, which might cause the commanded steering angle to change fast as the error would change very rapidly. Finally, the distance to the wall can be found by finding the length of the shortest line from the origin (minimum point-to-line distance). This line will be perpendicular to the line obtained by regression. Using the general result that the product of the slopes of perpendicular lines is -1, the shortest distance to a line from the origin can be calculated using the equation in line 21. To obtain the front distance, the ranges list values from a lookout range of $-5°$ to $5°$ are averaged to obtain an accurate measurement. The reason this range is used is so that there is no bias in the data if some range value produces a noisy result.

2.1.2 PID Controller

As a simple but useful algorithm, PID control uses a proportional, integral, and derivative gain to drive an error term to 0 (see Equation 1).

$$\begin{aligned}
&P = e_t \\
&I = \textstyle\sum_t e_t * \Delta t \\
&D = (e_t - e_{t-1})/\Delta t \\
&\delta = K_p * P + K_i * I + K_d * D
\end{aligned} \tag{1}$$

For the purpose of wall following, the parameters K_p, K_i, K_d of the PID controller and the computation of error term e_t are dependent on the car's distance from the side and front wall, and the output signal is the steering angle δ. The time increment Δt was implicitly chosen by taking the difference between the time a drive command was published and the time a new sensor message was received. This ensures that the PID controller always works with new sensor data, so we can then compute the desired steering angle. Finally, the car can publish a message of type `AckermannDriveStamped` which specifies a velocity and steering angle, which abstract and encapsulate the low-level control.

Our main contribution lies in:

1. Computing the error term for PID control in a situation-specific way
2. Tuning the PID parameters by combining the Ziegler-Nichols tuning method in simulation and massive experiments in the real world. See algorithm 2 for the whole process.

Situation 1: PID control for straight wall following First, this error is calculated using Equation 2, where $D_{desired}$ is the desired distance from the wall and d_{wall} is the sensed distance to the right wall.

$$e = -side * (D_{desired} - d_{wall}) \tag{2}$$

The parameters were initially tuned in the simulation environment using the Ziegler-Nichols tuning method where K_p was set to allow the car to oscillate to find the critical gain K_C and the oscillation period T_C after which the gains can be calculated using Equation 3.

$$\begin{aligned} K_p &= 0.6 \cdot K_C \\ K_i &= 2 \cdot K_P / T_C \\ K_d &= K_p \cdot T_C / 8 \end{aligned} \tag{3}$$

However, while testing with the real race car, the integral gain seemed to cause steady state oscillations when the car was following its setpoint which meant there was no need for this integral term. Thus, a PD controller instead was used.

Situation 2: PID control for corner turning Furthermore, during the simulation and in-person testing, it became evident that a naive implementation of a follower that only looks at the distance to the side wall would result in a race car that is not robust to various real-world scenarios; for instance, it could not turn around corners well. Therefore, we designed a new way of computing the error in the PID control by weighting the signal from the side wall by 25% and the front wall by 75% using Equation 4. This creates the illusion that the race car has a larger error than it does in reality while approaching the front wall in a corner, allowing the car to turn seamlessly around the corner.

$$e = -side * (0.25 * (D_{desired} - d_{wall}) + 0.75 * (D_{front} - d_{front})) \tag{4}$$

Additionally, a higher proportional gain was chosen after tuning with no derivative or integral gain to allow for a large steering angle to be commanded. Another case that was challenging in simulation and needed to be looked at was turning an open corner. This was achieved by having a large lookout range so that car could lookout behind itself in order to still try and identify a wall.

Algorithm 2 Wall Following

```
Require: FILTER-SENSOR-DATA
 1: function WALL-FOLLOWING($scan$, $side$, $D_{desired}$, $e_{t-1}$, $\Delta t$)
 2:     $\theta_{min}, \theta_{max} \leftarrow [-0.34, 0.34]$
 3:     $D_{front} \leftarrow 1.75$                                  ▷ A heuristic threshold
 4:     $d_{wall}, d_{front} \leftarrow$ FILTER-SENSOR-DATA(scan)      ▷ Filter out outliers
 5:     if $d_{front} < D_{front}$ then                             ▷ Too close to front wall
 6:         $K_p \leftarrow 0.8$
 7:         $K_i \leftarrow 0.0$
 8:         $K_d \leftarrow 0.0$
 9:         $e_t \leftarrow -side * (0.25 * (D_{desired} - d_{wall}) + 0.75 * (D_{front} - d_{front}))$
10:     else                                                        ▷ Normal controller case
11:         $K_p \leftarrow 0.6$
12:         $K_i \leftarrow 0.0$
13:         $K_d \leftarrow 0.05$
14:         $e_t \leftarrow -side * (D_{desired} - d_{wall})$
15:     $P \leftarrow e_t$                                           ▷ proportional term
16:     $I \leftarrow I + e_t * \Delta t$                              ▷ integral term
17:     $D \leftarrow (e_t - e_{t-1})/\Delta t$                        ▷ derivative term
18:     $\delta_s = max(min(K_p * P + K_i * I + K_d * D, \theta_{max}), \theta_{min})$
19:     $e_{t-1} \leftarrow e_t$
20:     return $\delta_s$
```

2.2 Safety Controller (Aaron / edited by Jess)

Ensuring a vehicle system can automatically detect impending crashes and administer appropriate safety precautions is an essential part of any autonomous robotic system. Our objective is to create a safety control system that guarantees the vehicle will not crash into any object, regardless of the scenario. However, the safety controller must be implemented such that it does not hinder the vehicle from executing safe maneuvers (i.e. will not crash into any objects). These objectives are achieved by creating a danger region via the vehicle's parameters, obtaining distances from the LiDAR, and running a basic algorithm to determine if there are objects within the danger zone, triggering the safety module. The safety module commands the vehicle's velocity to be zero, effectively stopping the car before it collides with the detected object. Figure 5 illustrates the safety controller's process.

2.2.1 Danger Zone

The vehicle looks at a region of space in front of it to detect the presence of any objects. If any object is detected inside this region, the safety module is deployed immediately to stop the vehicle before it collides with the object. We term this region as the car's "danger zone." The danger zone consists exclusively of the front of the car because, for the purposes of this lab, the car will only be moving forwards. The safety zone is constructed by calculating four coordinate points in the car's coordinate system (refer to Figure 3)

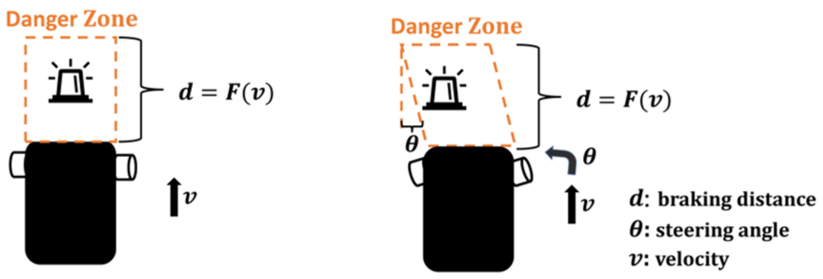


Figure 5: A comprehensive illustration of the safety controller while moving forward (left) and turning (right). The safety controller relies on constructing a danger zone parameterized by the car's width, steering angle, and braking distance which is a function of the car's velocity. When an object is detected within the danger zone, the safety module is deployed, stopping the vehicle in time before any collisions occur.

fig. 3. The general shape of the danger zone is a parallelogram, whose area is a function of the car's width, current velocity, and current turning angle.

The first two base points of the danger zone are positioned at the front-left and front-right bumper of the car. We ensure the danger region is wide enough to cover the entire front width of the car, including its tires. We obtain the measurements using a ruler starting from the LiDAR, which reads distances in meters. This is done because the distance measurements used to detect objects in the environment will come from the LiDAR; therefore, we set the LiDAR as the origin of the car's coordinate plane. The car's currently commanded velocity and steering angle is then used to calculate the remaining two depth points of the danger zone. We choose to write the parallelogram's depth as a function of the car's velocity because the braking distance of the car increases as the velocity increases. The function mapping the car's velocity to the braking distance is experimentally determined, as it is difficult of measure the deceleration of the car. In short, we obtain the associated braking distance needed at various velocities to safely stop the car before colliding with an object. We then run a regression to interpolate between the obtained data points. This data collection process is elaborated on in detail under the experimental evaluation of the safety controller in section 3.2 (see Figure 15).

As the car turns, the danger region should reflect the path the car will travel. For instance, a car turning left should look more towards the left. It should not stop for objects that are, for example, in front of the car if they can be avoided. Thus, the car's current steering angle is used to add a horizontal bias to the two depth points of our danger region, bending it in relation to the steering angle. This enables the car to execute all collision-free turns, but still stop the car appropriately if it recognizes an impending collision. Figure 6 illustrates this concept. The locations of the danger region's vertices are summarized by Equations 5 and 6 (illustrated previously in Figure 5):

$$\begin{aligned} \text{leftbase} &= (\text{LiDAR\_to\_front}, \text{car\_width}/2) \\ \text{rightbase} &= (\text{LiDAR\_to\_front}, -\text{car\_width}/2) \end{aligned} \tag{5}$$

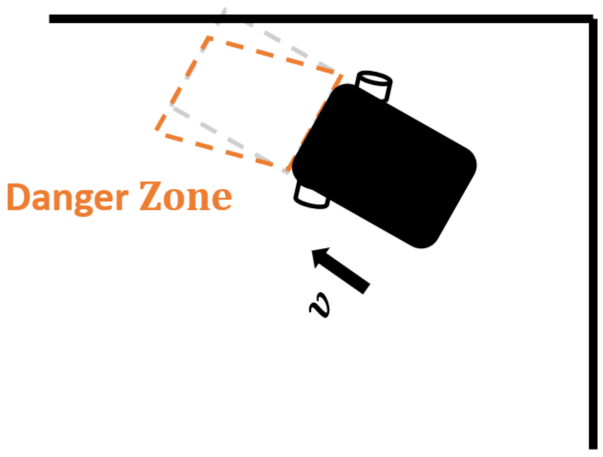


Figure 6: Illustrating the danger region's ability to shift on a safe closed turn. If the danger zone does not have the ability to shift and remains straight (gray), the danger zone intersects with the wall and mistakenly commands the car to stop. Angling the danger zone via the commanded steering angle (orange) allows the danger zone to avoid the wall. Thus, the vehicle can successfully complete the safe left turn.

$$\begin{aligned} \text{lefttop} &= (\text{LiDAR\_to\_front} + \text{braking\_dist}, \text{car\_width}/2 + \text{braking\_dist} \cdot tan(\delta_s) \\ \text{righttop} &= (\text{LiDAR\_to\_front} + \text{braking\_dist}, -\text{car\_width}/2 + \text{braking\_dist} \cdot tan(\delta_s) \end{aligned} \tag{6}$$

2.2.2 LiDAR Data

The safety controller then obtains data from the LiDAR, utilizing the perceived distances of the surrounding objects. Since our danger zone is in front of the car, the distances corresponding to angles on the positive-x side of the car's coordinate plane are obtained (corresponding to $-\pi/2$ to $\pi/2$). These distances are then converted from polar to cartesian coordinates, which will then determine if an object is present in the danger zone.

2.2.3 Object Detection Algorithm

With the danger zone constructed and the LiDAR distances obtained in cartesian form, an algorithm detects if an object is present in the danger zone and therefore executes the safety module. We do this by iterating through the LiDAR points to determine if each point is located inside the danger zone. We utilize the ray casting algorithm to determine if a test point is located inside the safety region by counting the number of times a ray emanating from the test point crosses the edges of the defined polygon. When a LiDAR point has been located inside the danger region, the algorithm begins counting. If the algorithm has detected three successive points in the safety region, the safety module is immediately activated, publishing a velocity of zero to the car before it collides with the perceived object. A threshold of three points is used to help account for potential noise that may be presented through the LiDAR. It would not be ideal for the vehicle to simply stop because it sees, for instance, one noise point inside the danger region that may not even correspond to an object. We also count successive points rather than simply allowing any three points in the region to trigger the safety module, as successive

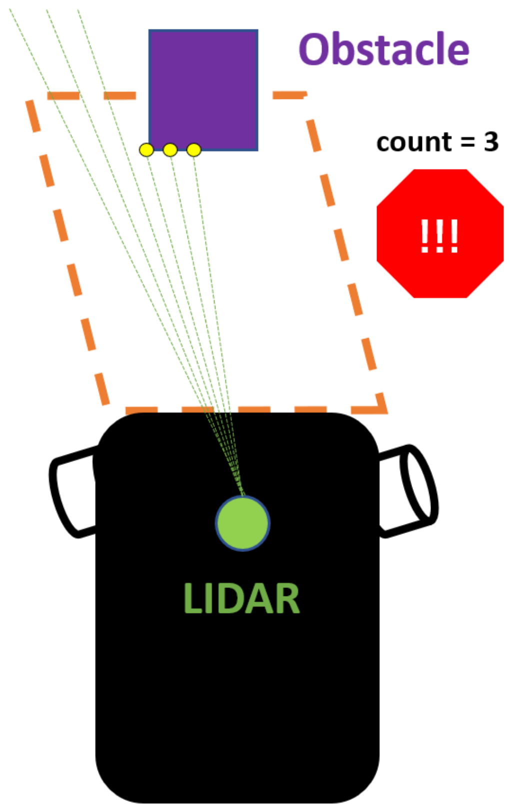


Figure 7: A visualization of the Object Detection Algorithm. The algorithm counts the number of successive points from the LiDAR data that are within the danger zone. If three successive points are found inside the region, the safety module is immediately deployed.

points more plausibly describes if a surface is in the danger zone. Having a threshold of three points (spanning approximately 0.75°) is more robust in guaranteeing that an actual object surface was perceived yet won't overlook objects that may be thin in width. Last but not least, the algorithm's ability to quickly break out of the iterations as soon as the threshold is surpassed allows the safety module to execute more instantaneously. Figure 7 illustrates this algorithm.

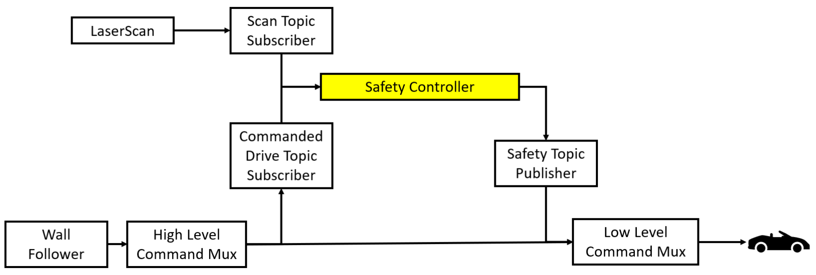


Figure 8: A graphical representation of the safety controller's ROS implementation. Via the different priority command muxes, the safety controller is able to intercept the commanded driving instructions from the wall follower and override them if the safety module is executed before publishing the commands to the race car.

2.2.4 ROS Implementation

To successfully obtain the necessary data and publish the safety module to the car, the safety controller intercepts the current driving commands via the topic `/vesc/high_level/ackermann_cmd_mux/output` and publishes any safety executions to `/vesc/low_level/ackermann_cmd_mux/input/safety`. The safety controller is able to work due to the race car's command mux which has different levels of priority. The safety topic is a higher priority topic than the navigation topic (where the wall follower commands are published to). This enables any published safety command to override any command the wall follower attempts to publish, stopping the car in time to avert any foreseen collisions. Figure 8 illustrates the ROS implementation of the safety controller. Algorithm 3 summarizes the safety controller.

Algorithm 3 Safety Controller

```
 1: procedure SAFETY-CONTROL($scan$, $ack$, $v$)
 2:     $\delta \leftarrow ack.drive.steering\_angle$
 3:     $D_{LiDAR} \leftarrow 0.1524$    ▷ The distance between LiDAR and the front edge of the car (m)
 4:     $W_{car} \leftarrow 0.33$    ▷ The width of the car (m)
 5:     $\theta_{view} \leftarrow \pi$    ▷ The whole front view taken into consideration.
 6:     $N_{point} \leftarrow 3$    ▷ Stopping threshold when counting obstacle points in a streak.
 7:     $n_{point} \leftarrow 0$
 8:     $p_0, p_1, p_2, p_3 \leftarrow$ CONSTRUCT-DANGER-ZONE($D_{LiDAR}, W_{car}, \delta$)
 9:     $x, y \leftarrow$ CARTESIAN-CONVERTER($scan, \theta_{view}$)
10:     for x, y do
11:         if POINT-INSIDE-DANGER-ZONE(x, y, $p_0, p_1, p_2, p_3$) then
12:             $n_{point} \leftarrow n_{point} + 1$
13:             if $n_{point} \geq N_{point}$ then
14:                 break
15:         else
16:             $n_{point} \leftarrow 0$
17:     if $n_{point} \geq N_{point}$ then
18:         $new\_ack \leftarrow$ CREATE-ACKERMANNDRIVE-MESSAGE($\delta$)
```

3 Experimental Evaluation

To ensure an effective wall follower and safety controller, a series of tests were designed to verify the functionality of the design. The tests were centered around creating controlled, reliable, and safe mobility for frequent and necessary autonomous vehicle maneuvers. Each test was repeated three times to guarantee consistency of the results and error was recorded to evaluate the car's success. All relevant videos/images related to our experimental testing can be found here.

3.1 Wall Follower (Jessica, Shara, Chuyue / edited by Aaron)

To guarantee the car's safety, create uniformity among the tests, and adapt to our spatially-limited testing environments, we chose a constant velocity of $0.5ms^{-1}$ for the majority of the wall follower test cases.

Table 1: Wall Follower Test Cases and Results

Priority Score	**Test Type**	**Test Description**	**Measurement of Success**
1	Straight Wall	Align the car parallel to wall, let it run for 5 seconds	The abs(error) should be <0.05m.
1		Align the car parallel to wall, offset by -50% and 50% from desired distance.	Time it takes for robot to converge to 0 error (+/- 0.05m) should not exceed 0.5 seconds. Maximum error should not exceed original offset.
2		Align the car head direction at -45° and 45° compared with the wall, and from twice the desired distance.	
1	Corners	Align the robot parallel to the wall facing a 90° closed corner. Run until the robot has turned the corner and no longer oscillates.	Time it takes for robot to converge to 0 error (+/- 0.05m) after peak error should not exceed 1 second.
1		Align the robot parallel to the wall 3m away from a doorway. Stop after the robot has travelled through the doorway.	
3	Cluttered Environment	Run the robot along an unsmooth wall in the lab classroom. Run until failure.	Robot should successfully navigate most obstacles in cluttered environment. Record limitations.

3.1.1 Default Parameters for Tests:

Throughout the tests, we set our target value to be 0 for the error. We define an epsilon of $\epsilon = +5\%$ to be an acceptable range of our error values. PID values used are $k_p = 0.6, k_i = 0.0, k_d = 0.05$ unless otherwise stated.

3.1.2 Straight Wall Tests

Straight wall test 1: starting at the desired distance with PID tuning For our first experiment, we decided to test whether our car could follow a straight wall starting at the desired distance from the wall. Moreover, we studied how different values for our control impacted our computed error.

1. Straight wall integral: $k_p = 0.6, k_i = 0.001, k_d = 0.05$
2. Straight wall integral derivative: $k_p = 0.6, k_i = 0.005, k_d = 0.1$

In Figure 9, we observe that the error converges to a large value and this is due to the lack of control from our integral term. We introduced the integral value into our testing after observing this behavior, so that we could have better control over our steering angle and how much we overshoot. Increasing the integral term and our derivative term helped us control how much we converged.

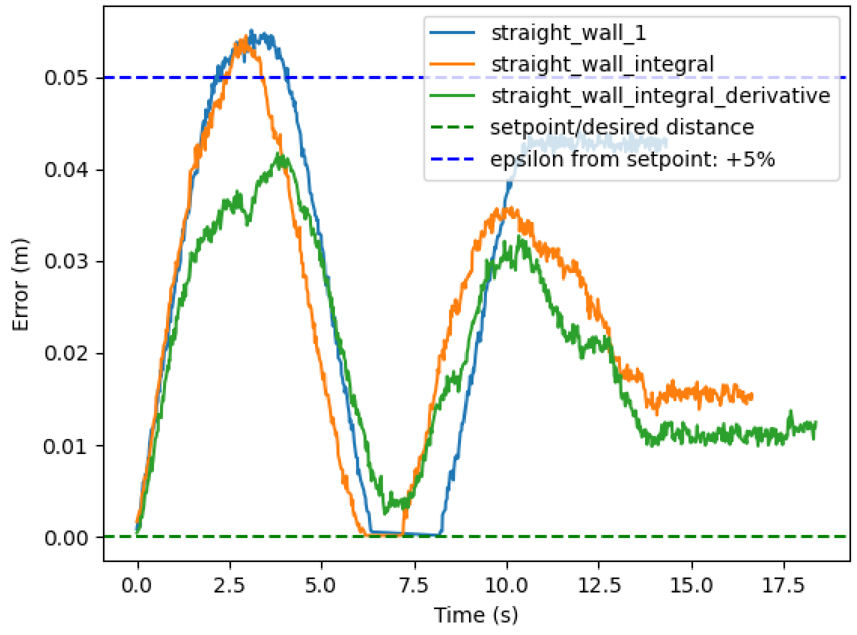
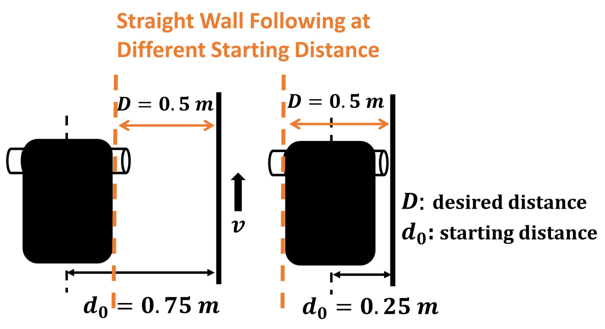
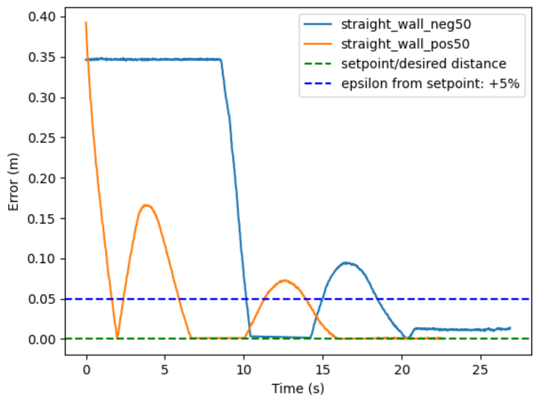


Figure 9: Straight Wall Test #1: starting at the desired distance with different PID parameters. As we observe through the blue graph, we see that the lack of an integral term allowed our error to fluctuate the most. Adding and increasing the integral term in the other graphs helped decrease and stabilize our error.

Straight wall test 2: starting at different distances from the wall We define the starting distance from the side wall $d_0 = \{25\text{cm}(-50\%), 75\text{cm}(50\%)\}$. Given that d_0 from our wall is greater than our desired distance, the behavior in the graph in Figure 10 exhibits readjustments to our robot's position and a convergence towards a smaller error term once we reposition ourselves to be closer to our target set-point.



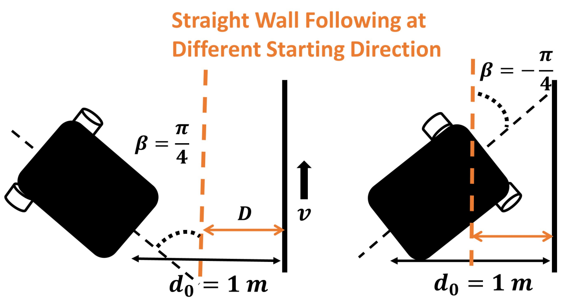
(a)



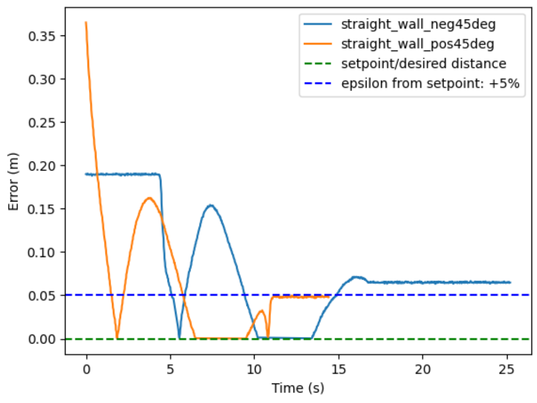
(b)

Figure 10: Straight Wall Test #2: starting at different distances from the wall. We begin a distance of d_0 away from the wall for each graph in the plot, as illustrated in the diagram. In the plot, we observe a higher error that quickly drops. This is due to the readjustment of the robot to our $D = 0.5m$ (our desired distance)

Straight wall test 3: starting at different direction against the wall In this experiment, we set the angle of our robot at $\pm 45°$ in relation to our wall. We can observe from Figure 11 that we can still minimize the error and stay close to our desired distance.



(a)



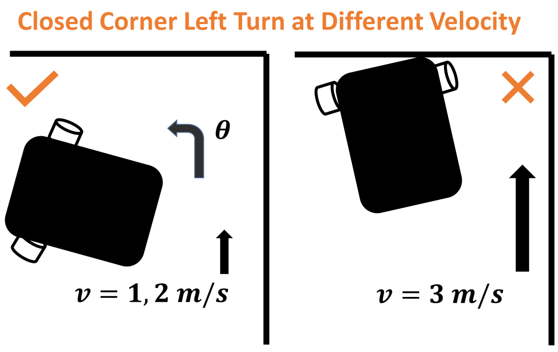
(b)

Figure 11: Straight Wall Test 3: varying starting angles relative to the wall. Our robot starts 1m away from the wall in both cases but is positioned at different angles. We can observe that the error of the robot that begins at an angle of $-45°$ has an error larger than our allowed epsilon and stabilizes slower compared to that of the robot with the starting angle of $+45°$

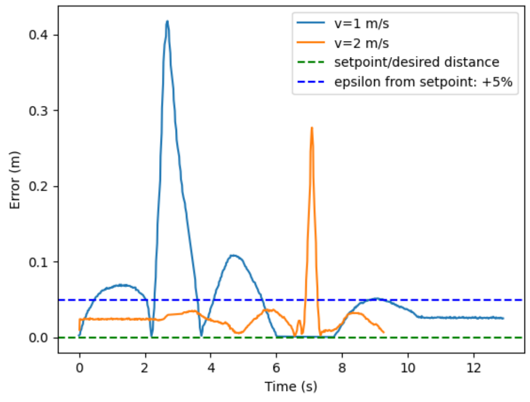
To conclude our first section of tests, we have observed that we are able to successfully drive straight along a wall without crashing.

3.1.3 Corner Tests

Corner test 1: closed corner turning at different velocity In our code, if we found our front distance was less than the reacting distance ($D_{front} = 1.75m$), we would start to take the front distance error term into account, which hopefully drives the car to turn. Currently, our reacting distance is a constant. However, as we increased the speed of our robot, we found it crashing into the wall when we increased our velocity to 3 m/s. Higher speeds result in wider turns and larger turning radii, so we must account for this with a higher reacting distance by making our reacting distance be a function of velocity to allow handling of turning at higher speeds.



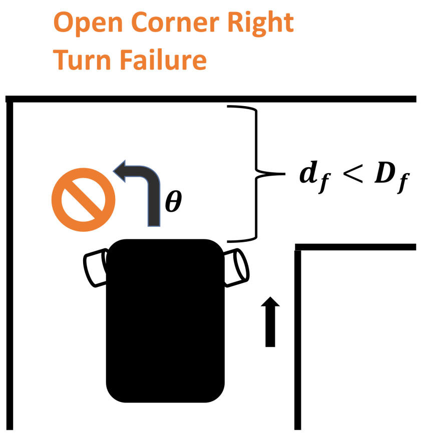
(a)



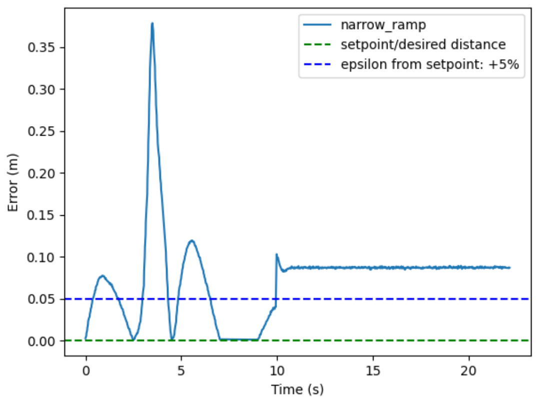
(b)

Figure 12: Corner test 1: Turns at 90° closed corners at different velocities. In the plot the error for $v = 3m/s$ is not included because we were not able to properly turn and this test contained a lot of noise. In the diagram, we illustrate how at higher speeds we get closer to the wall while turning when our reacting distance D_{front} is a constant number.

Corner test 2: open corner turning failed when front wall is near and detected In our current code, we fail to turn in the proper direction after following the right wall because the detection of our front wall makes us begin to turn left instead of considering the open corner that we have on the right side. Our code assumes that whenever we detect a front wall, we have a closed corner. Our wall follower cases are oversimplified and makes strong assumptions about our environment. In the appendix of this report (see Appendix 11 11), we have included pseudo code that uses a state machine to handle different corner layouts and cases where we are not near any walls (which we would've implemented given more time).



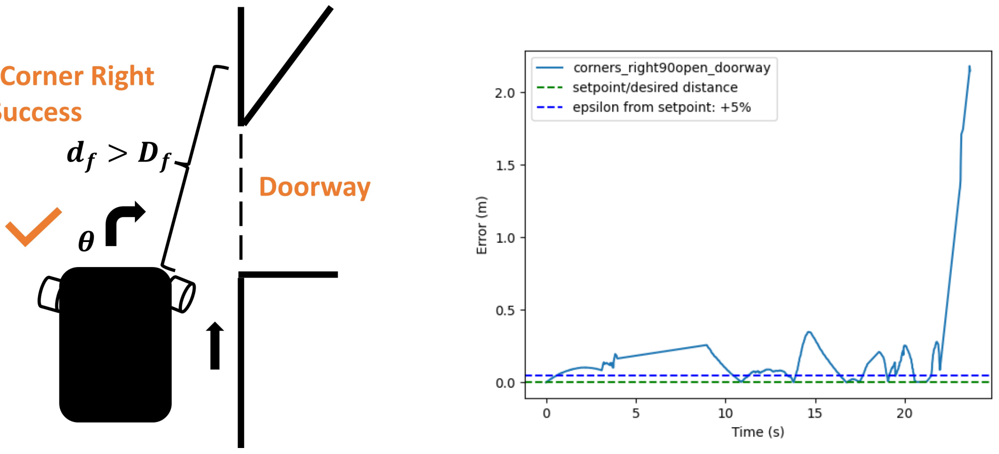
(a)



(b)

Figure 13: Corner test 2: Align robot parallel to a side wall that introduces an open corner along the wall we are following and make the robot face a front wall. Note that the robot turns into the wrong corner and error is above our acceptable margins.

Corner test 3: open corner turning succeeded into the doorway Contrary to the previous test, when we do not detect a front wall in the presence of an open corner, we are able to successfully turn into the doorway because it happens that $d_{front} > D_{front}$ and $D_{desired} < d_{sensed}$, which results in turning right.



(a) (b)

Figure 14: Corner test 3: Exposure to an open corner along the wall we are following without the presence of a wall in the front. The error we observe in this scenario is greater than other errors we have seen previously because the last five seconds displayed shows the error of the robot as it is passing through the door way since the (closet) wall in the hallway is farther from our robot.

3.1.4 Cluttered Environment Tests

The robot was able to successfully follow the wall without crashing into obstacles that were positioned against the wall to create an uneven wall path. The robot successfully maintained its desired distance from the wall and obstacles. A link to an example video of this test can be found here.

3.2 Safety Controller (Jessica / edited by Aaron)

Our initial testing for the safety controller focused on creating a velocity-dependent function to model the car's braking distance. This is vital to implementing a robust safety controller that is effective at various velocities. Assuming there is no lag time from data processing and autonomous control, the minimum breaking distance (D_{min}) is given by Equation 7 where V is the car's velocity and d is the car's deceleration.

$$D_{min} = \frac{V^2}{2d} \tag{7}$$

While we determined it was not realistic to measure the deceleration, we can extrapolate the quadratic relationship between the braking distance and the velocity. We measured $\frac{1}{2d}$ experimentally by determining the ideal D_{min} at different velocities. This experiment was conducted by placing the autonomous car 0.5m from a wall. The autonomous code was then run at a given velocity, with the braking distance decreased by 0.01m each test until a stopping tolerance of 0.05m from the wall was achieved. This experiment was repeated for velocities in the range [0.5, 3] at increments of 0.5m/s as seen in Figure 15. After determining a quadratic regression with low variance ($R^2 = 0.9985$), we implemented the equation into our code to ensure the safety controller operated effectively at multiple velocities.

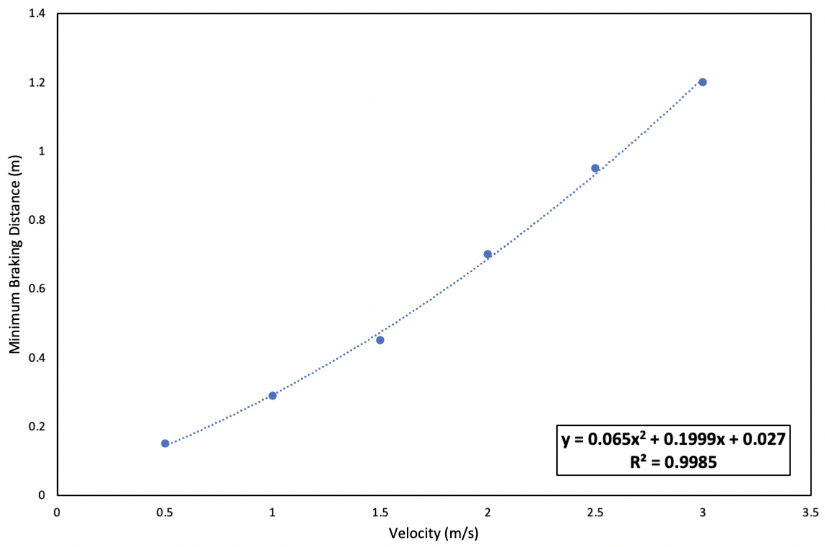


Figure 15: Braking distance at different velocities. An R^2 term of 0.9985 can be interpreted as an indication of a strong fit to our experimentally determined data points. Literature agrees that braking distance should be proportional to V^2, suggesting our regression equation perhaps has some validity and can be temporarily used to estimate the minimum braking distance needed to calculate the depth of the safety controller's danger zone.

Our safety controller test cases were designed to ensure the safety of the car and its surroundings, while not impeding upon the vehicle's mobility. To guarantee maneuverability, we first successfully ran all wall follower test cases with the safety controller implemented. The remainder of test cases focused on vehicle safety for realistic scenarios, including different velocities, different obstacles, and different driving scenarios. See Test Cases and Results in Table 2.

The safety controller passed all of our test cases in Table 2, giving us high confidence in the reliability of our safety controller in preventing potential crashes. The main limitation of the safety controller however is its danger zone. While turning, it is meant to mimic the car's trajectory and turning radius, which we simplified to a parallelogram. While we determined that this was an adequate approximation, there are certain edge cases where the car stops despite being capable of clearing an object, demonstrated in Figure 16. Additionally, we experimentally discovered that above a velocity of $2ms^{-1}$, the safety controller cannot consistently react to obstacles while turning corners.

4 Conclusion (Shara)

In this design phase, we successfully designed and iterated over algorithms to navigate in parallel to nearby objects while implementing safety precautions to avoid collisions. Transitioning from the wall-following code we wrote for a simulation to wall-following code for our physical car was challenging due to limitations in the time we had to tune values and test our robot in different scenarios. It was also challenging because we had

Table 2: Safety Controller Test Cases and Results

Priority Score	**Test Type**	**Test Description**	**Measurement of Success**	**Result**
1	Non-Impedance of Wall Following	Run all Wall Follower test with the safety controller implemented	None of the tests below $2ms^{-1}$ should fail	Pass
2	Dynamic	For all brick and chair tests run, the brick or chair should be moved out of the way of the robot after it successfully stops. (all at velocity=1.0)	After the brick is removed the robot should resume its path.	Pass
1	Different Object Types	The car should safely stop on a variety of different object types of different sizes, such as a brick, a human, a wall, and a cone.	100% stop rate	Pass
1	Different Car Velocities	The Safety Controller should be properly executed when the car is running at different velocities in range $[0.5, 2]ms^{-1}$ in increments of $0.5ms^{-1}$.		Pass
2	Turning	The safety controller is robust to objects that may be found while turning.	100% stop rate if object will impede (at higher velocities won't turn)	Pass

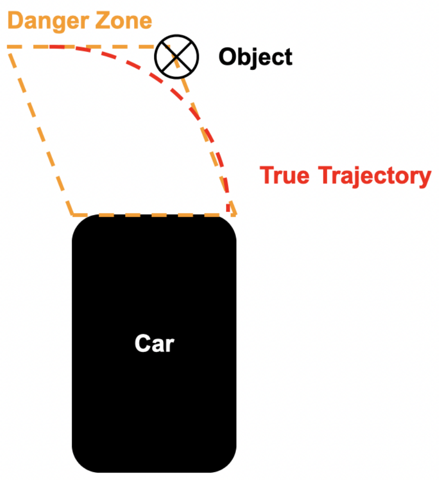


Figure 16: Limitation of Safety Controller at Edge Case. For simpler calculations, we estimated the car's danger zone to be a parallelogram; however, in practice the car's danger region should have curved sides that more closely mirrors the car's true trajectory. Utilizing the parallelogram causes the safety controller to inadvertently be employed in edge-case situations as illustrated where the vehicle could indeed safely maneuver around an obstacle, which we revealed through our experimentation.

to consider real-life implications that may have been negligible in our simulation. We needed to be able to successfully maneuver and stop at the correct time and places so that we could ensure the safety of our robot and the environment.

In order to ensure the correctness of our algorithms, we developed a set of comprehensive test cases. We evaluated how well we were able to navigate along walls with different control parameters by determining the error of how far we were from our desired distance throughout our trajectory. We further evaluated how well we turned corners at different speeds, and how our system handled different types of corners that we could approach. We also found that we could successfully navigate through a cluttered environment by keeping a safe distance from obstacles and stopping when needed. For testing our safety controller, we wanted to ensure that our wall follower was not impeded by our safety controller yet still being able to stop and move when objects were placed and removed from the path of our robot. We also wanted to ensure that we could stop in the presence of objects with different geometries and be able to stop when moving at different velocities. We also wanted to handle the case where we run into obstacles immediately after turning. We were able to satisfy all test cases for our safety controller for velocities under $2m/s$. We were also able to satisfy most of our test cases for the wall follower, but some behaviors for turning are undefined and we fail due to our algorithm missing the consideration of some important environment features in the detection of a front wall. In order to address the robustness of our algorithms, we have taken steps to begin pseudo coding a wall follower that considers our environment as a state machine and then making movement decisions. In the case of our safety controller, we can accommodate a larger range of velocities by experimenting with higher velocities because it is important for our race car to be able to travel faster and stop appropriately.

Overall, our wall follower and safety controller are able to function well according to our defined tests – but they can be improved. In developing our current system, we took unique approaches to create a working product, such as calculating our error as a weighted average of front and side distances to be able to turn better and filtering our data by using standard deviation so that all distances we consider are within a certain threshold. This elementary system design of an autonomous vehicle provides an excellent framework as we begin to add various new modules in future labs to increase the capabilities of our race car.

5 Lessons Learned

Aaron: On the technical side, I learned about how large the disparity truly is between getting something to work in simulation versus getting something to work in the real world. We ran into a tremendous amount of hardware issues with our robot at the beginning of the lab that we hadn't accounted for, which put our group behind schedule. Looking ahead, it is essential not to overlook these problems and be flexible enough to deal with such challenges. I think I also learned how we could be more efficient as a group in collaboration. Ideally it would be nice to have everyone working on the same thing at the same time, but it is impractical as individuals have different schedules. It perhaps would be more efficient to have individuals be in charge of different modules of the lab, especially given the time constraint of the lab deadlines. This allows time and effort to be allocated more evenly across all important aspects of the project.

Jess: I learned the importance of planning ahead. Since we already had the program for the Wall Follower written, we jumped right into working with the robot and developing the remainder of the code. By the time we realized we needed to have collected data for the write up and briefing, we didn't record some initial tests and had to quickly develop code for data collection. This will easily be implemented in the future, where we can sit down, create an organized plan, and list out all of our objectives before we begin the lab.

For communication I think we could have done a better job communicating changes and commenting on the code. When working in a team it's important that tasks are delegated to reduce working time, but exchanging issues and communicating plays a large role in saving time while problem solving.

Chuyue: One technical lesson I learned from this lab is the difference between simulation and real-world for a robotics algorithm. Also, from my teammates' implementation and our final choice, I learned the trade-offs between simplicity and robustness. At first, when the car didn't run well enough, Sarah and I tried to improve the robustness by taking more situations into account through another algorithm (in the appendix). While we were developing the algorithm, other teammates' working on tuning the original algorithm and it turned out to work pretty well!!! As for the communication part, at first I felt a little hard to speak out what I was thinking of or doing at hand (as a visiting student and not part of the MIT community). But as time went by, I found everyone in the group was really helpful and kind. And I realized that timely communication could indeed help everyone live better.

Nihal: The most important lesson I have learned completing this lab on the technical side of things is to have a plan of action before starting the experimentation for the lab. There is a lot of data that we could have collected including videos of working and failed tests which we couldn't because we didn't think about our testing procedure thoroughly. Also, having made a meeting notes document with all the important information there was a good decision. In terms of communication, we can split the times that we work on the car better by understanding when people are free a little better. Otherwise, I personally feel like the team has put in all their effort to make meetings and to get the lab completed.

Shara: I felt like this lab was a good introduction into the team and how we will plan our work throughout the semester. Since we were able to use code from our previous lab and modify it based off of our team members' code, it was a good learning experience to communicate technical ideas and decisions. It also gives us an introduction to everyone's communication styles. I also think that this lab allowed us to figure out how we want to organize our weeks and workload throughout the semester because we learned that things can take much longer than expected and that we may face technical difficulties. We also started a meetings document where we take notes and plan things out, for the next labs we should come up with test cases and go over all requirements and plan a schedule before starting. I think in the future we should have designated leads and teams for parts of the lab so that we can better use our time and resources. I was also surprised by how helpful RVIZ was in testing when we did not have possession of the robot, it helped us get a better idea of how things would run on our physical race car and tune values more easily.

Algorithm 4 State Transition

```
Require: DETECT-WALLS
 1: function STATE-TRANSITION($scan, d_{front}, d_{sensed}, D_{front}, D_{desired}$)
 2:     if $d_{front} > D_{front}$ & $d_{side} > D_{desired}$ then
 3:         $s \leftarrow$ DETECT-WALLS($scan$)
 4:     else if $d_{front} < D_{front}$ & $d_{side} > D_{desired}$ then
 5:         $s \leftarrow$ FRONT-WALL
 6:     else if $d_{front} > D_{front}$ & $d_{side} < D_{desired}$ then
 7:         $s \leftarrow$ SIDE-WALL
 8:     else if $d_{front} < D_{front}$ & $d_{side} < D_{desired}$ then
 9:         $s \leftarrow$ CLOSED-CORNER
10:     else
11:         $s \leftarrow$ ERROR
        return s
```

Algorithm 5 Action Execution

```
Require: STATE-TRANSITION, FILTER-SENSOR-DATA
Require: PID-CONTROL, GO-STRAIGHT, TURN-LEFT, TURN-RIGHT
  procedure ACTION EXECUTION($scan, D_{desired}, D_{front}$)
      $d_{side}, d_{front} \leftarrow$ FILTER-SENSOR-DATA($scan$)
      $s \leftarrow$ STATE-TRANSITION($d_{side}, d_{front}, D_{desired}, D_{front}$)
      if $s =$ RIGHT-WALL then
          PID-CONTROL($d_{side}, D_{desired}$)
      else if $s =$ NO-WALL then
          GO-STRAIGHT()
      else if $s =$ CLOSED-CORNER then
          TURN-LEFT
      else if $s =$ FRONT-WALL then
          TURN-RIGHT
```