Lab #3 Report: Wall Follower

Team #12

Yasin Hamed
Artem Laptiev
Evan Bell
Cruz Soto

Robotics Science and Systems

March 11, 2023

1 Introduction
by Cruz Soto

The goal of this wall follower lab was to adapt the simulation developed in the previous lab to properly follow a wall in this lab. In particular, the wall follower was set to maintain a certain distance from a given wall and appropriately respond to environmental changes such as inner and outer turns, variations in the wall geometry, and initializing movement at different angles in respect to the wall. This is the first step in developing a robot that can detect and respond to its environment by way of an active extoreceptive sensor.

Using a Proportional Integral Derivative (PID) controller, a mathematical system through which error can be mapped to a successful output (where success is minimizing the error), the team was able to individually build their simulations where the car could respond to all of these parameters.

Adapting this to a real robot had its difficulties, however. Most prominently, the wall follower code depended on LaserScan data, a datatype built from a 2D Light Detection and Ranging (LiDaR) system, but now required the use of a 3D Velodyne Lidar® puck, which outputted 2D data in a different format for a different range of angles, meaning that the data needed to be re-patched, rotated, and re-adapted for practical use.

This lab also lays the groundwork for safety and operation of the car as it introduces the team to the workflow of setting up a router, checking power to the vehicle, secure shell (ssh) connecting to the car, and the code sharing between members of the team and the car itself. On the safety end, one of the core requirements here is to construct a safety controller. The cars are already in very few supply, and the difficulties of managing this with many technical issues on the vehicle itself will be discussed later in the report. As such, to prevent any significant damage or need to decommission the car, a node which can detect and stop the car from moving to avoid a collision, overriding any inputs to do otherwise, was implemented. This will be useful in stopping any autonomous code tested on the car in future from leading to significant damage while operating at high speeds in future labs.

2 Technical Approach

2.1 Wall Follower Considerations

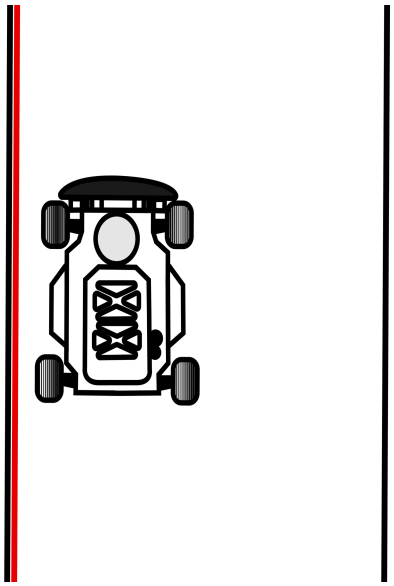
This is the math, least squares regression distance and math explanation. @Evan ?

2.2 Wall Follower Implementation by Yasin Hamed

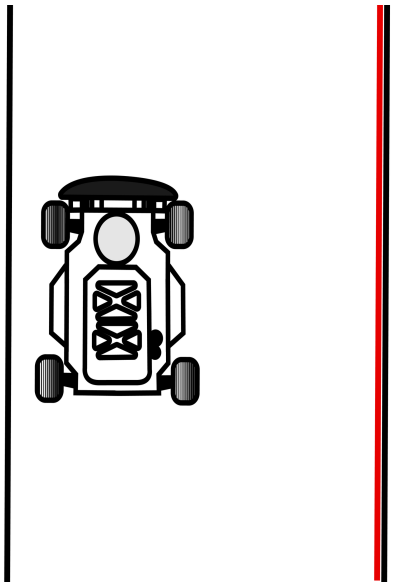
The first thing to note about this implementation of the wall follower is that the wall selection for the robot can be made static or dynamic which significantly affects the performance of the robot in different situations.

In the dynamic wall selection mode, the robot takes the LIDAR data, splitting it up into left and right sections. It takes an average of the ranges of the points on each side and decides to follow the side which it is closest to.

In the static wall selection mode, the car will always use the data on the prespecified side of the robot to follow the wall – that is to say that it will always try to follow either the right or left wall but will never switch from one to the other.



Here, the car has dynamic wall selection enabled, and is thus following the wall closest to it – the left wall.



Here, the car has static wall selection enabled, with the code set to follow the right wall. It is following the right wall despite that wall being further from the car than the left wall.

Once a line representing the wall relative to the robot has been calculated – as described in the previous section – the error signal is taken to be the difference between the desired distance (setpoint) and the actual distance of the car from the wall (shortest distance from the car to the generated line). This signal is then passed into the PD controller to control the steering angle of the car, driving it towards the setpoint. This method works very well for following generally straight walls without corners.

For corners, the wall following algorithm distinguishes between 2 cases: inside corners and outside corners. During the approach of an inside corner, the robot will be approaching a wall directly in front of it. Once the robot detects the wall in front of is within a critical distance, the generated line is shifted to be directly on top of the robot, so the robot will think that it is way too close to the wall it is following. This will result in the car turning away from the wall it is following, which is the desired behavior in anticipation of the inside corner that is approaching. The generated line will continue to be "on top of" the robot until the robot no longer detects a wall in front of it, i.e. when it has completed the turn.

The car deals with outside turns the same way it deals with following a straight wall. The points in the car's data set will simply become further and further away as the car approaches the outside turn, shifting the generated line to be further and further away, increasing the magnitude of the error and causing the

car to start gradually driving towards the wall during its approach to an outside corner, which is the desired behavior.

In the case of outside turns, the distinction between static and dynamic wall selection. Our algorithm works for outside corners only works for static wall selection in all cases. If dynamic wall selection is enabled, and the robot were trying to follow a wall into an outside corner at a T-intersection, it will begin to drive past the corner, turning slightly as the generated line adjusts to be further and further away. This works well until the average of the points on the opposite wall becomes smaller, and the robot then switches targets and begins following the opposite wall, which is not the desired behavior for the car. This problem is solved by enabling static wall selection on the robot since it will follow the desired wall regardless of how far it is from the car (given that it is still within the range of the LIDAR, of course).

As a side note, dynamic wall selection exists as an option in our code to allow for the robot to drive down the exact middle of a hallway, regardless of the width of the hallway.

2.3 Reformatting Velodyne Laserscans by Yasin Hamed

The simulation that we were provided with to test our wall follower code assumed a format for the laser data messages which provided an array of ranges which was symmetric about the front of the car. The total range of the LIDAR sensor, which spanned from $-120°$ to $120°$ was provided in a single laser message in the simulation.

Unfortunately, our team had to deal with the Velodyne LIDAR sensors which used the laser messages in ROS to represent their data very differently from the simulation. The Velodyne sensors published not one, but *two* laser messages for a single $360°$ sweep. Additionally, it used the full length of an array representing the $360°$ in each of these 2 messages, filling half of each message with "Inf" values. It is possible the Velodyne does this to publish data at a higher frequency, albeit incomplete data. Additionally, the ranges array always had a constant section of "Inf" values representing the back of the LIDAR where the actual car was. To make things even more confusing, the ranges data was not even symmetric about the front of the car. All of these novelties in the data representation led our team to write a separate node which subscribed to this data, reformatted the angles and ranges arrays to be exactly the same format as the laser messages in the simulation, and publish a new, reformatted laser message to a different topic. This new topic is where the wall follower node gets its LIDAR data from so that changes did not have to be made to the code which already worked in simulation.

2.4 Writing a Data Recorder

The code implentation of the data recorder and a walkthrough of the approach @Evan

2.5 Implementing a Safety Controller by Cruz Soto

Now that the robot has a configurable wall follower and is collecting data to properly document the functionality of said wall follower, it is necessary to implement a safety controller to prevent any damage coming to the robot in the case of a malfunction. The controller must satisfy a few requirements. The controller must:

1. Recognize the distance of the wall in front of it.
2. Intercept the drive commands outputted by other nodes.
3. Publish a movement-killing drive command to override any existing AckermannDriveStamped data.

To begin this implementation, the team wrote up an alogorithm to sort through only the portion of laserscan data that applied to a range

$$\theta$$

.

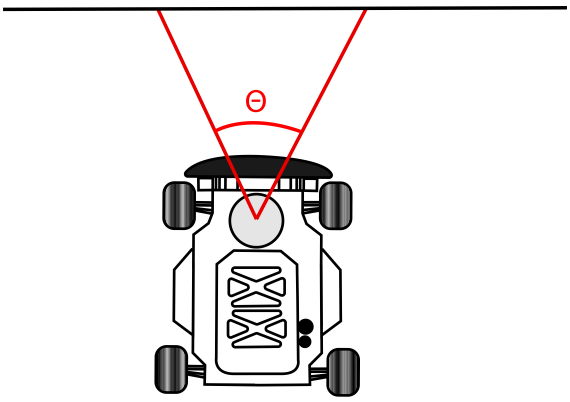


Figure 1: The robot detecting a wall with sweep angle θ

Getting the data would require an implementation that collects data as follows, sifting through an existing laser scan.

$i \leftarrow length(LaserScan)$
$L \leftarrow 0.3 * i$
$R \leftarrow 0.6 * i$
$LaserScan[L : R]$
for $k \in LaserScan$ **do**
  if $k = \infty$ **then**
    $k = d_{max}$
  end if
end for

The code which takes a laserscan of the front 90 degrees (30%) of the data Following this, the average distance to the wall can be calculated by taking the mean of these values. This official distance can then be printed to the console as well as plugged into a loop to prevent the robot from breaching a minimum distance to the wall. However, the scan itself has an issue with the velodyne LIDAR, as the Velodyne's minimum distance buffers around hte 0.5-0.4 meter mark, making it difficult to tell if the device is any clsoer as this will incur generation of the "inf" values seen in the psuedocode. To alleviate this, if the total quantity of "infs" in a scan exceeds a certain amount, the net distance to the wall will be assumed zero and the car will stopped.

while *alive* **do**
  if $num_{infs}/size_{front} > 0.4$ **then**
    distance_in_front = 0;
  end
end

Algorithm 1: A loop that sets he distance equal to zero if the number of "infs" is more than 40% of the total array size

In order to intercept the drive topic and publish a new command, the node needs to subscribe to drive topic at the highest level. This new command will now be spit out at the safety topic level, creating the following workflow.

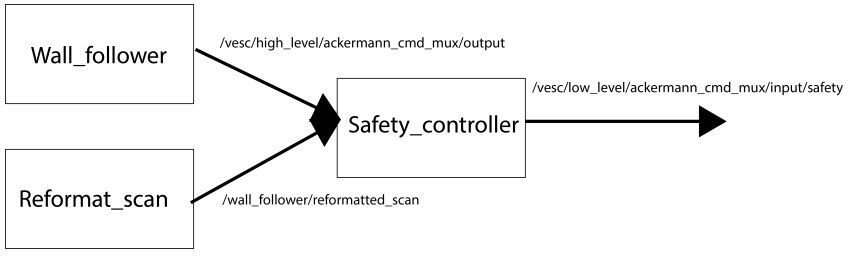


Figure 2: A block diagram dictating the publishing workflow for each node.

These were chosen as the safety topic is below the joystick topic, which we like to use to override commands in case the robot is difficult to collect, but above all other active nodes. This controller proved good at stopping with distances further from the wall at great accuracy, to be discussed in the experiment section.

2.6

3 Experimental Evaluation

3.1 Testing of the Safety Controller by Evan Bell

3.2 Testing of the Safety Controller by Cruz Soto

The safety controller setup was tested at a set of multiple distances with a single speed running directly at a wall. This car was always started at a distance of ten meters and the distance decreased until the robot hit the wall (as the distance, or base_link is not at the perfect distance zero). Distances were also checked by the Velodyne LIDAR as well as physically to it with a tape measure. This yielded the following results:

Stopping Distance (m)	**Actual stopped distance (m)**
3	1.4478
2.5	0.7874
2	1.3208
1.75	1.0414
1.5	0.889
1.25	0.6096
1	0.381

Which had a surprising degree of error and showed that the controller was malfunctioning in some way.

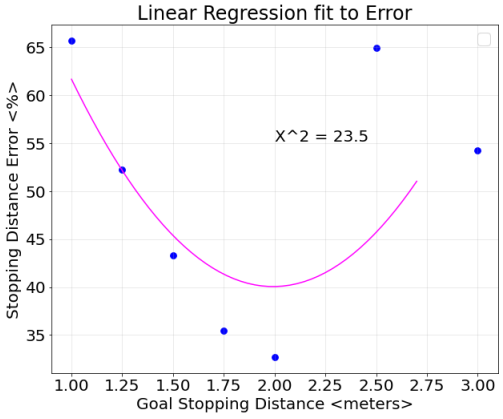


Figure 3: A block diagram dictating the publishing workflow for each node.

After inspecting the LIDAR and looking at data generation in RViz, it was

found that the robot was actually running slightly skew to the wall by 30°, and so the angle adjustment in the reformatted scan node was adjusted to reflect this. Following this adjustment, as well as increasing the sample collection size for each distance from two to three samples and averaging them, the error greatly decreased and demonstrated a clear inverse relationship between stopping distance and the error in said distance. The χ^2 value for the exponential fit also greatly decreased, indicated a low probability of better fit parameters being generated. This gives a high probability that there is a direct causation between stopping distance increasing and proportional stopping distance error decreasing. In the future, the safety controller will be tested with wider viewing

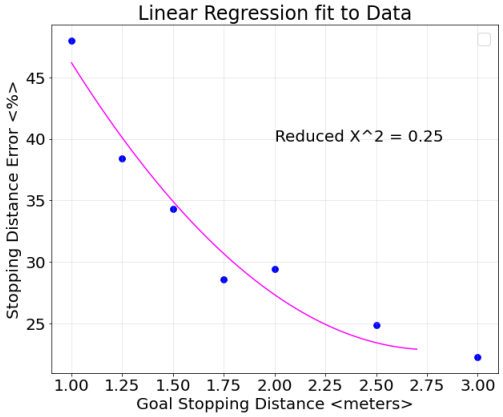


Figure 4: A block diagram dictating the publishing workflow for each node.

angles for the LIDAR sensor (to sense walls nearly perpendicular to the robot) as well as test at speeds above 1 meters per second (to make sure fast testing doesn't lead to incurred damage). It should also be run at the wall with a greater spread of angles to ensure that damage is prevented even when the car is skewed towards the obstruction. The controller should also be tested with the fast introduction of obstructions (such as team members jumping in front of it) to make sure nobody within or without the team is injured while code is running.
Discuss the distances we picked, difference between actual stopping distance and distance anticipated in

4 Conclusion

Summarizes what you have achieved in this design phase, and notes any work that has yet to be done to complete this phase successfully, before moving on to the next. May make a nod to the next design phase.

(no more than 500 words) @Artem

5 Lessons Learned

5.1 Cruz Lessons Learned

Managing time with the robot is difficult, and since the schedule of the team during the day was hectic, the best choice was to meet in the evenings and at the night, leading to a lot of diffuculty if and when the robot refused to work. Troubleshooting the controller, batteries, and, msot importantly, the Velodyne LIDAR, took a significant toll on our team.

Converting the Velo scans to a readable format took three days, significantly depleting our team's ability to implement and test the wall follower. Even with the wall follower implemented, tuning the controller was hard at close distances owing to the maximum range of the device itself capping out within 0.5 meters.

In order to better collaborate with peers, starting on the technical report even when the project isn't done (parallel performance) will be crucial in future labs for proofreading and reorganizing both the briefing and report. Presents individually authored self-reflections on technical, communication, and collaboration lessons you have learned in the course of this lab.

5.2 Yasin Lessons Learned

The biggest takeaway from this lab was that real life implementations of ideas will never work as they did in simulation – not just due to inaccuracies in the physical models used and idealizations made – but also because data will not always come into the system in the same way that it was assumed it would be received in the simulation.

This goes hand in hand with another lesson learned – time management. I realized that we as a team should plan to get the project done days in advance of the actual deadline since several factors (including the sharing system which severely limits our time with the car) are certain to inhibit the progress that we plan to make and postpone our termination of the project.

6 Formatting examples for figures, pseudocode, etc

6.1 Tables

Many other table packages and options exist but here is one example:

item 11	item 12	item 13
item 21	item 22	item 23

6.2 Images

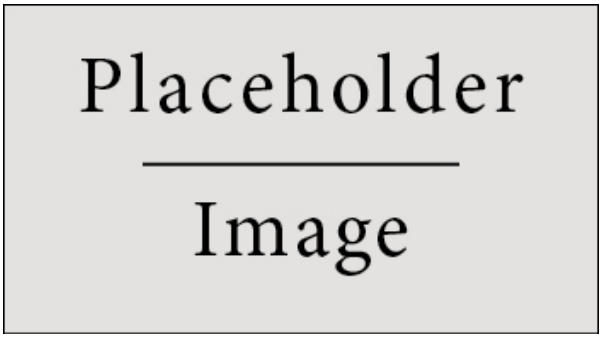


Figure 5: Figure caption.

6.3 Code Blocks and Algorithm Pseudocode

```
json
{
  "6.141": "normal",
  "16.405": "woke",
  "no_sleep": "spoke"
}
```

```
def do_something_productive():
  if not_productive:
    do_work()
  else:
    cry()
```

```
while alive do
    if sleepy then
        sleep;
    else
        eat;
    end
end
```

Algorithm 2: caption

$i \leftarrow 10$
if $i \geq 5$ **then**
  $i \leftarrow i - 1$
else
  if $i \leq 3$ **then**

$i \leftarrow i + 2$
end if
end ifcaption