RSS Lab 3 Report: Wall Follower and Safety Controller
Team 9

Susan Ni
Kevin Carlson
Talia Pelts
Jonathan Samayoa
Vlad Seremet

OVERVIEW AND MOTIVATIONS - Jonathan Samayoa

This lab was the first lab in which we met our teams and received our race cars. Thus, the purpose of this lab was an introduction - an introduction to the robot and an introduction to the team. We had to understand how to communicate with the robot with a joystick and through our code, how to make the robot do what we wanted it to do, how to keep the robot safe, and how to work together as a team.

The first task of the lab was communication with the robot with a joystick. This was essential going forward, as we always want to be the first in command when controlling the robot. We want to be able to stop the robot when necessary. For all autonomous robots, there is always a human who is first in command.

The second task of the lab was communicating with the robot with our code. We had already written a wall follower algorithm in a previous lab for a simulated robot. In the simulation, our autonomous robot could follow a wall at a desired distance. However, it was now our turn to put this code to work in a real robot. This was not straightforward, as we had to augment our code to fit the frame and mechanics of the robot.

Once we figured out how to communicate with the robot, we had a third task of keeping the robot safe. The robot is expensive and we did not want faulty implementation or an edge case to cause the robot to hit a wall and break. We implemented a safety controller that would override drive commands from the wall following code to stop the robot whenever it got too close to walls and other objects.

Since one of our final tasks is to race our robots autonomously, this lab is the foundation for the rest of our future works in this class. We will not be able to move forward with other components of our autonomous race if we are not able to accomplish the simple task of following walls. However, this lab is not only applicable to this class, but it is also the basis of many impactful real-world applications such as self-driving cars. For example, self-driving cars have to perform similar wall-following tasks to stay within road lanes and have safety controllers to prevent crashes when objects unexpectedly come too close to the car.

As we worked on solving all of these tasks, we began to learn about what dynamics work best for our team. We started to notice what we had to work on and what we did well together. Hopefully, by the end of this semester, we will all improve in our team working abilities. This lab was just the start.

PROPOSED APPROACH — Susan Ni

Ros Implementation

Our race car uses the Robot Operating System (ROS) to manage sensed data and push drive commands to the race car. In this lab, we are using the 3D Velodyne LiDAR on the our car (Figure 1). The scanner creates 3D laser scans that can be assessed by the robot to understand its distance from objects in its environment. In this lab, laser scan data is assessed by our wall-following algorithm and is eventually translated into a drive command.

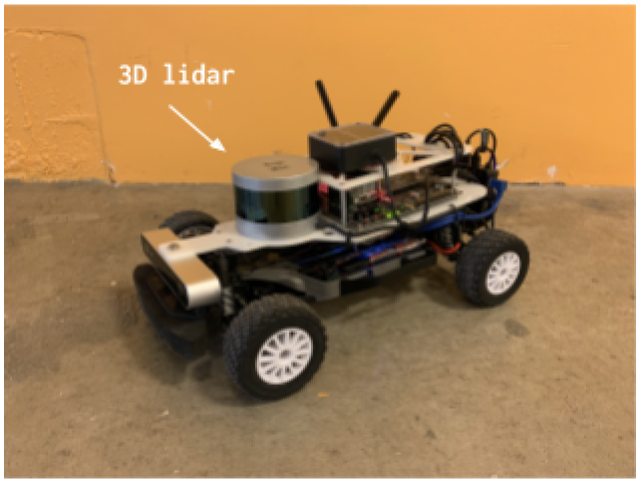

Figure 1. An image of our race car, displaying the 3D lidar that is used to obtain laser scan data of the car's surroundings.

To manage the data, ROS uses pub/sub messaging in which different nodes can publish data onto topics and other nodes can subscribe to those topics to retrieve the data. Our LiDAR publishes the laser scan data onto a topic we called "/scan", which our wall-follower node on the robot subscribes to, so every time new data is published, the robot can receive the data, process it, and then publish driving commands, such as setting a velocity and a steering angle, to a topic we called "/vesc/ackermann_cmd_mux/input/navigation".

However, since our wall-follower algorithm can possibly fail in some edge cases, we also have a node for a safety controller. This node intercepts the drive commands published by the wall-follower node to check if these drive commands will cause the robot to crash. If these driving commands are safe, the safety node publishes the original commands. If an incoming crash is detected, the safety the node overrides the previous drive command with one that prevents the robot from crashing. On top of both of these nodes, we have teleop running, which means that our inputs on the joystick will trump any command published to drive topics (Figure 2).

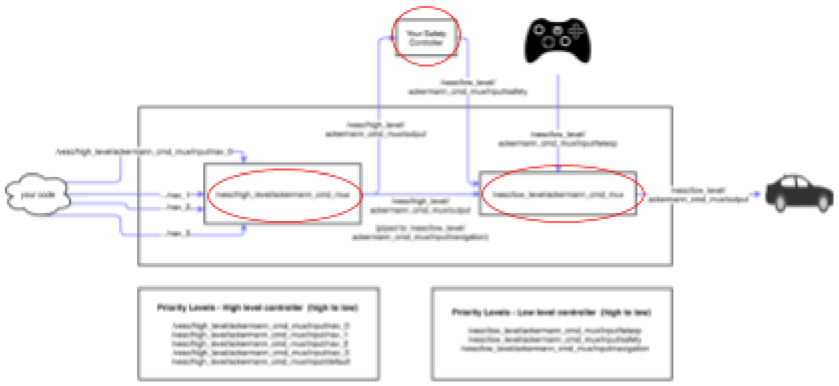
Your Safety Controller
your code
Priority Levels - High level controller (high to low)
Priority Levels - Low level controller (high to low)

Fig 2. Our racecar listens to the following chain of command. The code publishes to the topic on the left. Our safety controller can interfere with what is published. Our joystick (right) has the final say before the race car drives.

Technical Approach to the Wall Follower in the Robot

Our main task for this lab was to integrate the wall-following algorithm code from our simulated race car lab onto the real robot. The robot uses a set of parameters including a desired distance and side to follow the wall (right or left) to adjust its steering angle to ultimately follow walls. The algorithm parses the laser scan data and uses a proportion-derivative controller based on its estimated distance from the wall and angle from the wall to calculate a steering angle. The other task in this lab was to create a safety controller that ensured our robot would not crash into walls.

I. Slicing the Correct Frame of Data

Since the robot is set to follow walls on only one of its sides at a time, it makes logical sense to only consider the laser scan data from that specific side of the robot. Therefore, the range data has to be sliced based on the appropriate field of view, i.e. 0 to π/3 for following the left side and -π/3 to 0 for the right side. However, the x-axis (0 degrees) of the 3D LiDAR is not aligned with the x-axis of the robot. The 0 degree of the LiDAR is opposite of the charging cable, so it is clear that the LiDAR's axis is rotated by π/3 with respect to the robot's x-axis (Figure 3). As a consequence, the ranges data can no longer be sliced directly based on the angles that should be used for the respective sides. Now, an angle θ in the robot's frame is θ + π/3 in the LiDAR's frame (Figure 4). By the robot's frame, we want π/3 to 2π/3 to represent the left side and 0 to π/3 to represent the right side.

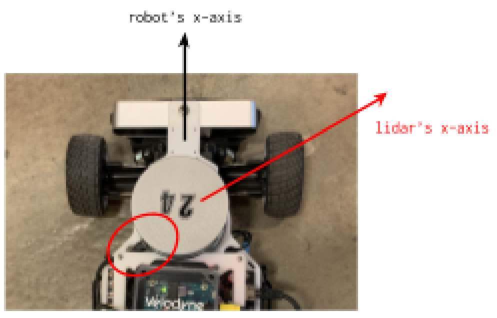

Figure 3. The lidar's x-axis is in-line with its charging cable. With the way the robot is set up, this means that the lidar's x-axis is rotated by π/3 with respect to the robot's x-axis.

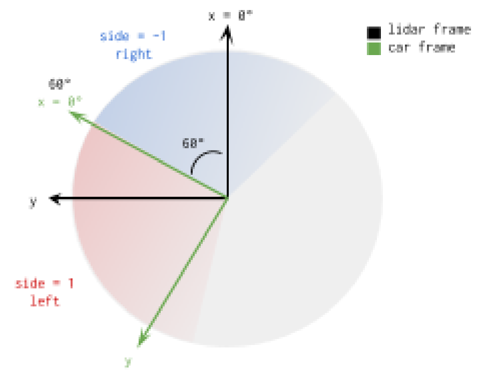

Figure 4. The black axes indicate the x and y frame of the lidar. This is compared to the x and y axes of the car frame (where x = 0 is the front). Notice the 60 degree shift.

This angle offset caused a lot of confusion at first. It was always known that we had to shift the angles we were slicing by π/3, but we thought we had to rotate them back to the robot's frame. After slicing the range data and transforming the range and angle data to cartesian points, we multiplied all of these points by a rotation matrix. However, this led to over-compensation for the offset. When we were trying to sense the right wall, we ended up sensing the left wall. To fix this issue, we dropped the usage of the rotation matrix. We concluded that adding a factor of π/3 to all of the angles we wanted in our slice was sufficient.

II. Linear Regression

After slicing the correct selection of the range data and transforming the points into cartesian points, we used those points to calculate a linear regression to model the wall the robot was following. A linear regression is not only useful because it simplifies the model of the wall and smooths out irregularities, but it also provides a convenient method to calculate the estimated distance between the robot and the wall. The origin of the coordinate frame of the linear regression is the car, so the distance, d, between the wall and the robot follows the point to a line distance formula,

$$d = \frac{|ax + by + c|}{\sqrt{a^2 + b^2}}$$

where (x,y) is the origin and a, b, and c can be determined by the equation of the linear regression in the form $ax + by + c = 0$.

The standard linear regression from the Python numpy package gives a good approximation of the wall in most cases, but it is not robust enough. The line is heavily skewed by outliers in the laser scan data, causing the robot to behave undesirably. For example, there was a test case in the wall follower simulator that tested to see if the robot's controller was robust to holes in the wall (Figure 5). With just the simple linear regression, the robot would constantly escape through the hole and not follow the desired wall. As a solution, we add weights to each point in the linear regression so the regression algorithm would favor points closer to the car. Therefore the points far away from a hole in the wall have a minimal impact on the linear regression. Naturally, the inverse of the distance of the points to the car was used as a first attempt at weighting. The robot was still escaping from the hole, so a stronger weight of the inverse square of the distance $\frac{1}{x^2 + y^2}$ is used in the final linear regression.

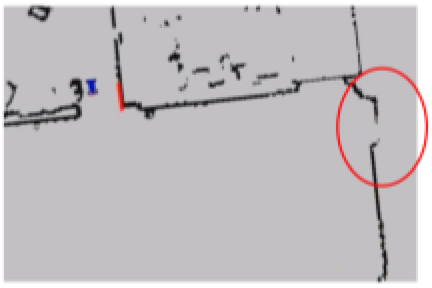
Figure 5. The simulator wall has hole in it (circled). Before weighing the distance data in the linear regression, the robot would exit out the hole.

III. Proportional-Derivative Controller

Based on the linear regression model of the wall, the robot uses a proportional-derivative controller to determine its steering angle when following a wall at a desired distance. We first start with just a proportional controller to allow the robot to adjust itself to the desired distance from a straight wall. The proportion term, P, is calculated using the formula,

$$P = K_p \cdot (distance_{estimated} - distance_{desired}) \cdot side$$

The K_p term is the gain factor that can be used to scale the proportion term to suit the the conditions of the robot. The difference between estimated and desired distance allows the robot's steering angle to be proportional to its current distance error. The estimated distance can be calculated using the distance from the linear regression as previously mentioned. The

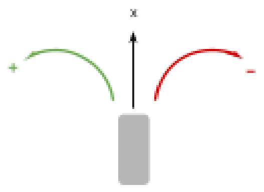

Figure 6. Angle sign conventions for the robot frame.

$side$ term represents the side that the robot is using to follow walls, where -1 represents the right side and +1 represents the left side. This term is necessary because the direction in which the robot has to turn depends on which side the robot is trying to follow (Figure 6). When the robot is following the right wall, the calculated error term has the opposite sign of the angle in which the robot has to turn, so multiplying by $side = -1$ corrects the difference in sign. When the robot is following the left wall, the calculated error term has the same sign as the angle in which the robot has to turn, so multiplying by $side = 1$ preserves the sign. See Figure 7 on the following page for a comprehensive look at how the robot uses its error to update it's position.

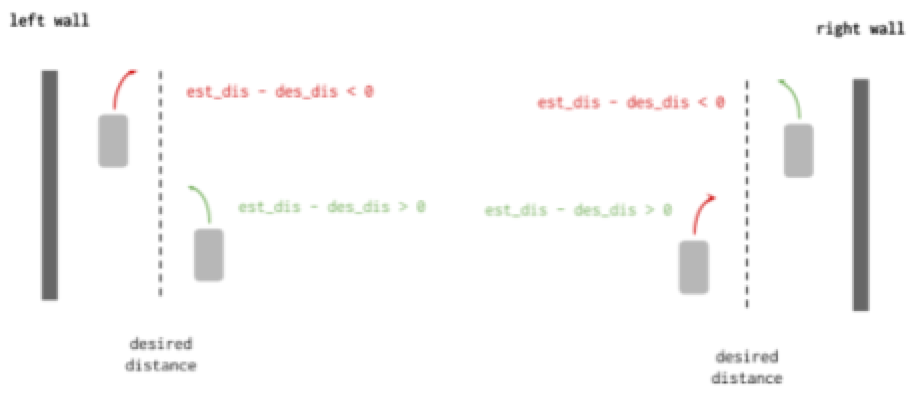

Fig 7. When the robot is too far from the wall, it adjusts its angle to move closer to the wall. When the robot is too close to the wall, it adjusts its angle to move farther from the wall.

With only the proportional term, the robot is able to adjust its distance from the wall properly, however, it has a difficult time rounding corners. So far, the steering angle is only a function of the robot's distance from the wall, so the derivative term was chosen to be a function of the robot's angle from the wall. This is important because there were cases in which the robot would start driving perpendicular to the wall. Without accounting for the robot's angle from the wall, the robot would set a large steering angle to compensate for the fact that it was far away from the wall and overshoot the turn.

The robot's angle from the wall can be calculated by finding the angle of the linear regression, which follows the formula $tan^{-1}(slope)$. The derivative term our robot uses, D, is

$$D = K_d \cdot \frac{slope}{1+|slope|}$$

which is a simplified version of the inverse tangent function. The final steering angle is simply the proportional term added to the derivative term.

The gains for the proportional and derivative terms were determined empirically. With the two gains set to be equal, the robot did not round corners efficiently. Since the derivative term is what controls the robot's ability to turn well, the derivative gain was increased until the robot was able to smoothly round corners.

IV. Safety Controller

An implementation of a safety controller is necessary in order to stop the robot from crashing if a bad command is given to it. Our first priority for the safety controller is to ensure the robots safety. Our first version uses a 2-D scan from our LiDAR to map the robots environment. From this scan data, we find the minimum value of the scan (which translates to an object's distance from the car). If value is smaller than our set “safety distance” of 0.55 m, our safety controller takes over the driving commands and set the robot’s velocity to 0 m/s, bringing it to a stop (Figure 8).

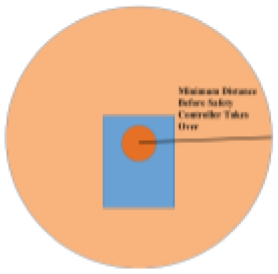

Fig 8. If an object enters this radius of the robot, the safety controller takes over.

Our second implementation of the safety controller resolves some issues created by the 3D scanner. The first issue is that our LiDAR is only capable of sensing objects farther than 0.5 m from the robot. Whenever an object gets closer than 0.5 m, the laser scanner returns an infinity value. With our initial implementation of the safety controller, this would cause the robot to stop and then stutter and continue forward if the object got too close. We fix this issue by checking for infinite points in our scan. We interpret this data as an object that is closer than 0.5 m away and let the safety controller take over. Due to the environment that we are using our robot in and the downward tilt of the 3D scanner, it is a safe assumption that the car will not encounter an object that is outside the LiDARs maximum range. Thus, our approach should not cause the robot to stop in safe conditions.

Our third and final implementation of the safety controller accounts for noise and allows the robot to recover from unsafe scenarios. Instead of sensing every point, we look ahead and to the sides of our robots in 10 degree scans (Fig 9). We then check to see if 20% of the collected distances are either less than 0.55 m or equal to infinity. If this is true, the safety controller takes over. By creating a threshold of 20%, we disregard noisy data. The response of the safety controller is also augmented. When a robot senses an object too close on its right side, the robot halves its current velocity and turns completely left until it is operating under safe distance conditions. The robot performs the same on the left side, now turning right. If an object is too close in front of the robot, the robot reverses and corrects itself in a manner so that it can continue to follow a wall.

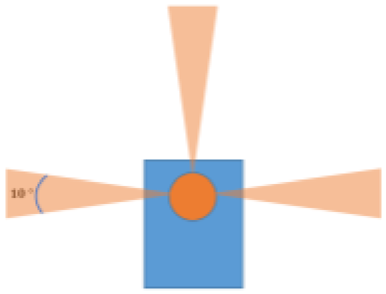

Fig 9. The robot takes 10 degree scans of the its front and sides to observe for objects that may be too close.

Looking forward, there are a few key aspects that we would like to work on. We want to use more than three scans, to give our safety controller a more robust view of the robots surroundings (in the case where a small object appears in the blind spot of the three current ranges). Secondly, while our current algorithm works well for our current speed, it begins to struggle as the robot’s speed increases. We will be focusing on modulating our “safety distance” as a function of the robots speed, as well as modulating the speed of the robot itself as it approaches walls and objects. This, in combination with our current recovery algorithm, will allow for a robust safety controller that will work at any speed and with any environment our robot will encounter.

EXPERIMENTAL EVALUATION – Kevin Carlson

The team conducted a series of qualitative tests to determine if the proposed approach translated to expected real-world performance. While we planned to perform quantitative tests to determine average robot error, we were unable to execute these tests due to the lengthy troubleshooting required to ensure our LiDAR was working properly. Our qualitative analysis ultimately determined that our proposed approach successfully met the lab requirements.

I. Wall Follower Evaluation

To evaluate the wall follower implementation, we ran the wall follower in 5 rooms with distinct wall patterns for each side of the robot. In determining whether the wall follower was executing correctly, we looked at 3 primary qualitative measurements. The first qualitative measurement we focused on was the distance at which the robot followed the wall. This metric provided us with instantaneous feedback on the performance of the wall follower by simply watching the robot perform the task. The second metric we evaluated was the steering. This offered insight to the performance of the robot through the consistency of the robot steering angle. If the steering angle was shifting quickly across positive and negative angles, we would know that the robot was overshooting. The final metric that confirmed the consistency of the wall follower was based on how it reacted to unexpected obstacles or abnormal walls and corners. With these tests, the robot performed well by correcting its trajectory fast enough to avoid the obstacle while maintaining enough control to return to following at the desired distance from the wall.

II. Safety Controller

The safety controller was tested and improved in a sequence of steps that gradually led to a more robust safety controller.

The first step in testing our safety controller was to ensure that the robot would stop once the minimum distance of 0.55 meters was detected. This did not initially work well because the robot requires approximately 0.25 seconds to stop when traveling at approximately 1 m/s. The real-world stop delay caused the robot to go within 0.5m from the wall. Since the Velodyne LiDAR cannot detect less than 0.5m, the LiDAR returned “inf” values and the robot continued to drive forward. This testing led us to decide to set all “inf” values to 0 in the safety controller.

With the front slice of data successfully stopping the robot before hitting the wall, we implemented the left and right slices to ensure we would stop if we came to close to an obstacle on either side. Testing with the three slices was conducted using log statements to display which slice would trigger the safety controller. By running the wall follower and placing obstacles randomly in its path, we were able to conclude that this simple safety controller was reliable in stopping the robot at a minimum distance from the wall.

The final step in the test process of the safety controller was to ensure that the implementation of the corrective actions worked as expected. These tests were performed using a node that drove the robot straight at a constant speed. Once the robot reached the wall, the safety controller would stop and correct the robot while not coming in contact with any obstacles. We ran these tests in four rooms that all had different wall patterns and obstacle types. Because the robot never hit a wall or obstacle, we concluded that the safety controller worked as designed in our proposed approach.

LESSONS LEARNED - Talia Pelts

This lab taught us technical, problem solving, and team working skills.

I. Technical Lessons Learned

The first technical lesson learned in this lab was how to manipulate the scan data from the Velodyne 3D sensor. As mentioned above, the sensor was shifted 60 degrees clockwise. In order to account for this shift, we wrote additional functions in our Wall Follower code that would rotate the scanned data 60 degrees counter clockwise. It was essential that we figure out how to do this in the first lab, as we will need the scanner in future labs. In the following week, we hope to write a separate python file that implements the conversions, so that we can easily adjust laser scan data for all future python files.

The second technical lesson learned in this lab involved learning how to transition from simulation to real life. In this lab, we quickly learned that success in a simulation does not equal success in the real world. Although the Wall Follower code we used performed with a 98% accuracy in the simulator, when we implemented it in our robot, our robot drove into the wall. A large portion of the errors resulted from the shift in the 3D LiDAR. However, even when that was fixed, we also had to update some of the parameters in our linear regression and PD controller to get our robot moving more steadily. Sometimes, abandoning approaches that work well in theory for new approaches that work well in the real world can be frustrating. However, the team was very thoughtful in deciding what portions of the original Wall Follower to keep and what portions to augment.

Lastly, we also learned how to create a safety controller for our robot. This was essential to ensuring that our robot would not accidentally crash in any future lab. We deliberated what sides of the robot to focus our safety controller on and at what distance to slow down. We also deliberated on what to do after the robot stopped (move back or stay put).

II. Problem Solving Lessons Learned

Problem Solving in this lab was essential. Our robot came with errors that were not common among the other teams (mainly due to the 3D scanner). We had to find these problems and then fix them efficiently. One of the best moments of problem solving in this lab was how we combined two problems to create a solution. In addition to being shifted, our LiDAR could not detect objects that were closer than 0.5 m from the robot. The LiDAR was also heavy which weighed the robot down, causing it to see the floor. Immediately, we realized that the range issue could affect our safety controller, as if we were too close to a wall, the robot scanner would return infinity values, indicating that it could not sense the obstruction. We then saw an opposite issue with the front weighing. We thought the floor would cause the robot to stop prematurely. However, after brainstorming, we learned that the 0.5 m range actually prevented the robot from stopping due to fsensing the floor or itself. Additionally, since the scanner looked down, it never gave us infinity values for objects in range because the floor would always come into vision. Therefore, we realized that we could infinity values to indicate to the robot that we were too close to a wall. This solution worked well for the safety controller portion of the lab. We may only need to make some adjustments in the future when the robot has to move down an incline and the floor becomes further away.

III. Teamwork Lessons Learned

In this lab the team learned the importance of delegating work. Since we were relying heavily on Susan's wall follower code, it was difficult for anyone to help her implement the 3D LiDAR shift and later debug her code. We also had to implement a safety controller, but that could only be done when Susan's code was fixed. Therefore, we never had a long period of overlapping work that we could do together. We hope that future labs will be different and will allow us to split up the work more evenly. Since a lot of bugs were fixed this week, hopefully next week can be more efficient and we can split up the work immediately.

We also learned the importance of team communication this week. We found it difficult to find a common time to meet, so we would often meet with a team member missing. This caused some difficulties when catching up the missing team member. In the future we will have a schedule where we keep track of all of the work that gets done during every meeting so teammates always know what is going on. However, we also greatly improved our communication skills when working together. At first, we were not being a cohesive team. Susan had her bug that she needed to fix and the rest of us were trying to help her, but we weren't really listening to each other. We would often come to team meetings and look at Susan's code and repeat possible solutions that were mentioned before or sit back. However, as the week continued, we began to plan together and deliberate decisions as a team. This kept lab members engaged even when there was no work to delegate and most importantly, it allowed us to reach a solution faster. For example, we discussed the wall follower in detail before Jonathan implemented it. This is what allowed us to realize that we could use our problem of the front weighing as a solution to the 0.5 m range. Communication was vital to our problem solving. Even though there is room for improvement, we are getting there.

POTENTIAL IMPROVEMENTS - Vlad

Although our robot performed nominally, there are some things that can be improved. First, the code responsible for processing the lidar data (rotating 60 degrees and performing adjustments) should be decoupled from the code responsible for the PD controller. ROS best practices dictate that concerns should be separated into different nodes. This would provide an extra layer of abstraction and would make it easier to modify and tweak the code without breaking too many things.

Second, the safety controller must be adjusted to be more robust and handle various situation smoothly. Currently, our safety controller modulates its speed based on the distance from the wall. We think that it is best to control both speed and acceleration based on both the distance from the obstacle and the velocity with which the robot is approaching the obstacle. This will allow our robot to avoid collisions at higher velocities.

Finally, we noticed that using a PD controller for a robot with Ackermann steering sometimes leads to abrupt turns and oscillations, which in turn lead to worse performance. To address that, we will implement the pure pursuit controller, which we will likely use in the final competition as well.

Edited by Talia Pelts