Autonomous Race Car: LiDar-Powered Wall Following and Emergency Braking Algorithm

Diego Contreras, Kevin Huang, Nathaniel Morgan, Weiming Zhou

Abstract (100 words) [Nathaniel] → RANSAC, PD, safety

In this lab, we implemented a wall following and safety controller algorithm on an autonomous racecar. The wall follower uses a PD controller to maintain a desired distance from a wall, with RANSAC used to fit a line to the LIDAR data and estimate wall position. The safety controller monitors a wide LIDAR cone and halts the robot if any obstacle is detected within 0.25m. We tested both systems and found the wall follower to be stable in most environments, while the safety controller consistently overshot the target stopping distance by around 15 to 19cm due to the car's momentum not being accounted for.

Introduction and Objectives (∼250 words) [Nathaniel]

For Lab 3, our team worked on deploying two algorithms onto a physical racecar: a wall follower and a safety controller. The goal of the lab was to take the simulation work from the previous labs and get both systems running reliably in the real world.

The wall follower is designed to drive the car forward while maintaining a consistent distance from a wall on either its left or right side. To detect the wall, we use LIDAR data and RANSAC to fit a line through the relevant scan points. From that line, we calculate the perpendicular distance from the car to the wall, which is then fed into a PD controller that outputs a steering angle. The car continuously adjusts its steering to keep itself at the desired distance.

The safety controller runs alongside the wall follower and acts as a failsafe. It scans a wide angle of over 180 degrees around the front of the robot, and if any LIDAR ray detects an obstacle within 0.25m, it immediately commands the car to stop. The threshold of 0.25m was chosen because it is small enough that the car can still follow walls closely while still providing a meaningful stopping condition for real obstacles.

Together, these two systems allow the car to navigate a hallway autonomously while protecting against unexpected collisions. This lab was also our first time testing code on the real racecar hardware rather than a simulation, which introduced challenges around sensor noise, physical inertia, and hardware-specific tuning that were not present before.

Wall Follower Design (∼400 words) [Kevin]

For our wall follower, we decided to incorporate Diego's wall follower as it performed well in simulation. The wall follower has was created with a two-pronged approach.

First, we made sure that the wall follower is able to guide the car along a simple straight wall without any turns or corners. This was achieved with a classic Proportional-Derivative controller (PD Controller). Traditional PD controllers focus on correcting a system based on it's current error from its objective and the rate of change of that error. In this scenario, however, we defined two different errors for this controller.

For the P term, the error passed in was the difference between the desired distance from the wall and the robot's measured perpendicular distance to the wall. The D term got the heading error instead, which is the angle of the wall relative to the car's path. This multi-signal implementation produced desirable results. Minimal oscillations were observed in both simulation and real-world testing as the vehicle's path was being corrected by the PD controller.

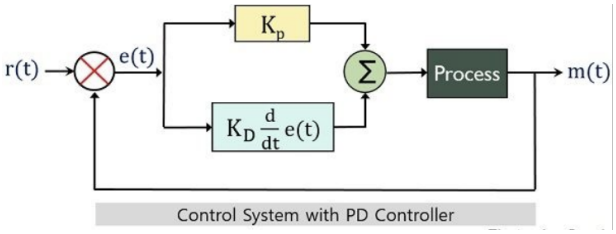

Figure 1: *PD controller response while following a wall.*

Once that was complete, we turned our attention to producing robustness in the vehicle's response as it approaches an inner corner. The vehicle detects this with a velocity-dependent lookahead mechanism that examines LiDAR points directly in front of the car. Specifically, we define a forward-facing cone of scan angles centered around the vehicle's heading and compute the minimum distance to any obstacle within that region. This forward distance provides an early indication that the car is approaching a wall directly ahead, which is representative of an inner corner.

If the closest forward obstacle falls within the lookahead distance, the controller interprets this as an approaching inner corner and computes a corner urgency value. This urgency value increases as the forward wall becomes closer, allowing the controller to respond more aggressively as the vehicle approaches the corner.

Rather than abruptly overriding the wall-following controller, the controller is able to smoothly transfer control from the PD controller to the corner steering through the urgency value. When the forward wall is still relatively far away, the urgency value remains small and PD dominates steering. However, as the vehicle gets closer to the inner corner, the urgency increases and the controller gradually shifts control toward a stronger steering signal that turns the vehicle away from the wall it is following. This blending prevents the PD controller from hindering the collision-avoiding turn while still preserving stable wall following on straight segments.

RANSAC vs. Polyfit: Handling Corner Outliers (∼300 words) [Kevin]

Traditional techniques include fitting a line through a set of data points obtained from LiDAR on the desired side of the with a convenient linear regression algorithm such as numpy.polyfit, provided by popular python libraries such as numpy. This method computes a line of best fit by minimizing the squared error between the observed LiDAR points and the estimated line. This was the initial implementation in our codebase, but simulation revealed that LiDAR measurements contained significant noise and outliers. Because least-squares regression uses all data points, even a few outliers could skew the estimated wall position and orientation.

To address this issue, we incorporated RANSAC (Random Sample Consensus) line-fitting, a robust linear regression algorithm designed to handle noise in a dataset. It does this by repeatedly sampling subsets of points and compares the resulting models to see which has the least outliers. Although we did not collect quantitative metrics to formally compare the two methods, the qualitative improvement was clear during testing. The robustness of the RANSAC algorithm allowed the wall follower to maintain a more consistent estimate of the wall's orientation and distance. The difference between the two approaches can be visually observed in the following figures, where the RANSAC fit remains stable despite the presence of outlier points, while the standard Polyfit regression is noticeably skewed by them.

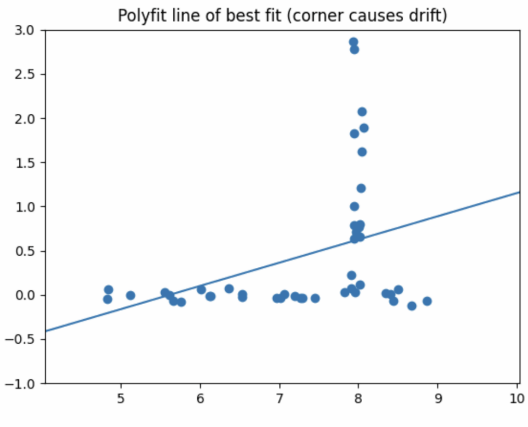

Figure 2: *Simulation of wall-fitting with Numpy Polyfit.*

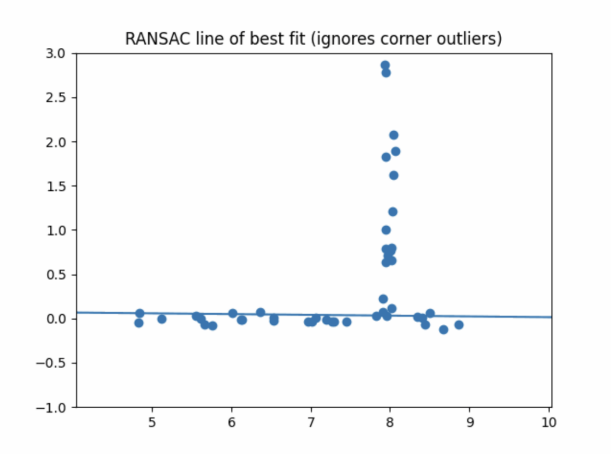

Figure 3: *Simulation of wall-fitting with RANSAC.*

Safety Controller Design [Diego]

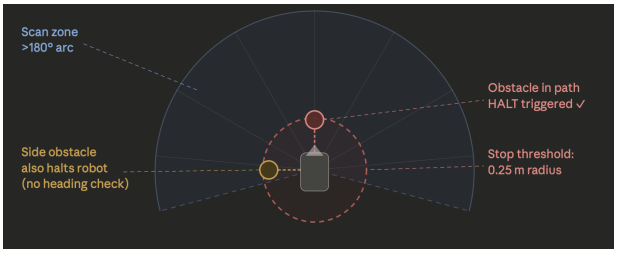

Figure 4: *Our safety controller looks in a cone (in blue), and stops when a ray is closer than a set distance (in red)*

The safety controller is what prevents the robot from accidentally ramming into people, obstacles, and walls should the wall follower fail. Our safety controller is built on the assumption that it will be used for future labs, so it was designed to be general and simple; It takes in LaserScan data from the robot in a >180 degree angle, and halts the robot from moving if any of those rays report a something lower than a set distance. For our wall follower robot, our set distance was 0.25m. This is just small enough to allow the robot to follow the wall well, while stopping should it hit an obstacle.

However, our safety controller has some weaknesses. Firstly, it does not take into account the heading of the robot. That is, for our wall follower, it scans in every direction, not just the front of the robot. This limits how far we can look ahead without interfering with the workings of the wall follower. It also results in some odd cases where the robot appears to seemingly be halted when there is no obstacle in front of it, as there can be obstacles to the side of the robot, or even its own wires.

Another major drawback of the safety controller is that the speed of the robot isn't taken into account either. As shown in the safety controller testing down below, this results in the robot not stopping at the set distance specified due to the inertia of the car when it stops.

Wall Follower Testing and Results (∼400 words) [Weiming]

We tested our wall follower at various speeds (1 m/s, 2 m/s) and in various scenarios (straight line, 90 degree inward turn). Our measurement was the robots perceived distance to the wall, and we tested for the cross track error relative to a desired distance of 0.5 meters to the wall.

Due to time constraints, our data collection was not as robust as we would have liked. We fix the robot to always follow the left wall, and do not vary the desired distance. We intended to collect hand measurements of data, but chose not to. For the 90 degree turn, we found that repeatedly stopping the robot would kill its momentum, and thus lead to a measurement unrepresentative of the robots true performance.

Below are plots for the straight line tests:

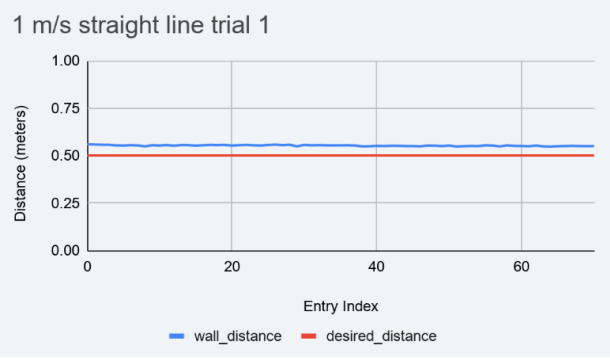

Figure 5: *Plot of desired distance (red) and perceived distance (blue). Perceived distance is distributed with mean 0.552 meters and standard error .0004 meters*

The blue line represents the robots perceived distance

Figure 6: *Plot of desired distance (red) and perceived distance (blue). Perceived distance is distributed with mean 0.711 meters and standard error .0028 meters*

to the wall, and the red line is a constant line representing the desired distance. This is measured in 40 Hz, the rate at which the laser scanner sent messages. The steady state mean distance is 0.552 meters and 0.711 from the wall at 1 m/s and 2 m/s respectively. We can observe that in both straight line tests, there is a consistent positive error in steady state which increases with velocity. We discuss this below in the next section.

Below are plots for the 90 degree turn tests:

Figure 7: *Plot of desired distance (red) and perceived distance (blue) as the robot makes an inward 90 degrees turn at 1 m/s*

The perceived distance measures the distance to the wall that RANSAC is tracking. As the robot completes the turn, this wall shifts from the left wall to the vertical wall. This happens precisely at the peak. Here are videos to the trial at 1 m/s and the trial at 2 m/s.

The robot starts noticing the wall at timestep 1. The

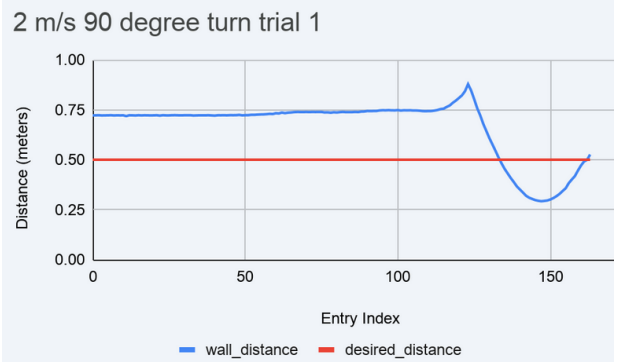

Figure 8: *Plot of desired distance (red) and perceived distance (blue) as the robot makes an inward 90 degrees turn at 2 m/s*

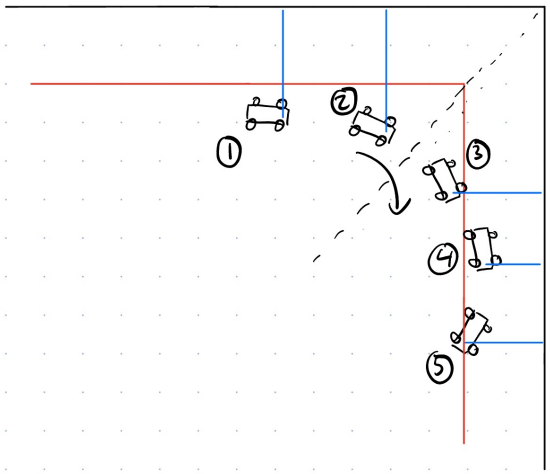

Figure 9: *The trajectory of the robot as it makes a turn with perceived distance to wall (blue) and desired distance (red)*

robot attains its maximum distance from the wall between time steps 2 and 3. It then attains its minim distance to the wall at time step 4 before entering steady state again after time step 5. In the video, the

Steady-State Error Analysis (∼250 words) [Weiming]

Currently at steady state when following a straight wall, our robot has a consistent positive error. This error amount varies proportional to velocity. We hypothesized this error was due to the wall detection algorithm incorrectly identifying a point on the wall

it was trying to follow as part of the vertical wall in front of it.

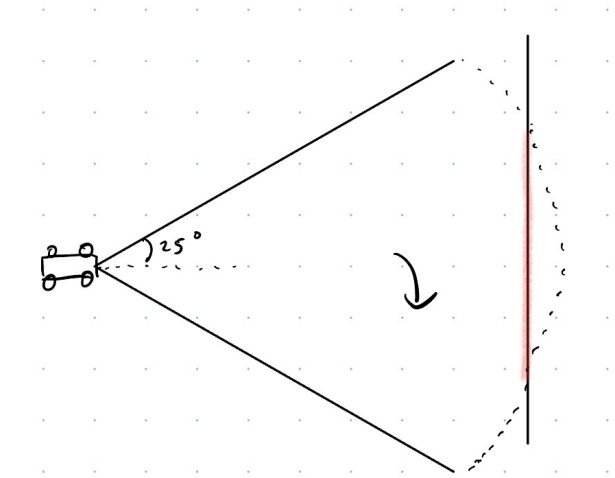

Figure 10: *Robot correctly identifying a vertical wall in front of it*

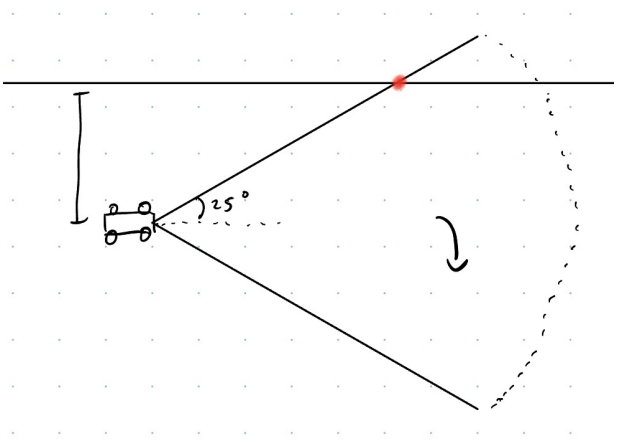

Figure 11: *Robot incorrectly identifying horizontal wall as a vertical wall in front of it*

When the robot detects a vertical wall, it turns away from the wall it is following to account for this. In the case it correctly detects a vertical wall, this turn eases into the inward turn needed to avoid collision. However, our safety controller algorithm only detects the minimum distance point within the code, so a point along the wall the robot is following will always be within this vertical wall detection cone. Thus, true steady state is only attained when the angle change of the PD algorithm counteracts the angle change of the vertical wall detection. This hypothesis is supported by the data collected; the measured steady state distance matches the calculated steady state distance needed for these two steering components to zero out.

To fix this issue, there are two approaches:

1. We can run RANSAC over the points within the vertical wall detection cone to see if a wall actually exists, instead of checking if any point is too close.
2. We can upgrade our PD controller to a PID controller. This is agnostic to the actual issue at causing the positive error, but the integrated error will counteract the consistent positive error at steady state.

Safety Controller Testing and Results (∼300 words)

To evaluate how effective our safety controller is, we decided to run an experiment where we varied stopping distance and velocity and measured the resulting error from where the robot should've stopped (stopping distance) and where it actually stopped.

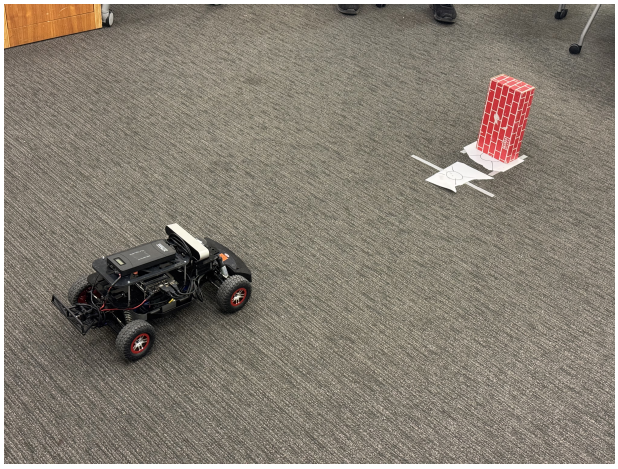

Figure 12: *Robot was placed far away from an obstacle, and set to drive at it at a certain speed.*

Here are our results:

Distance (m)	Actual - Expected (cm)
.25	19.5
.5	14.5
1	15.5

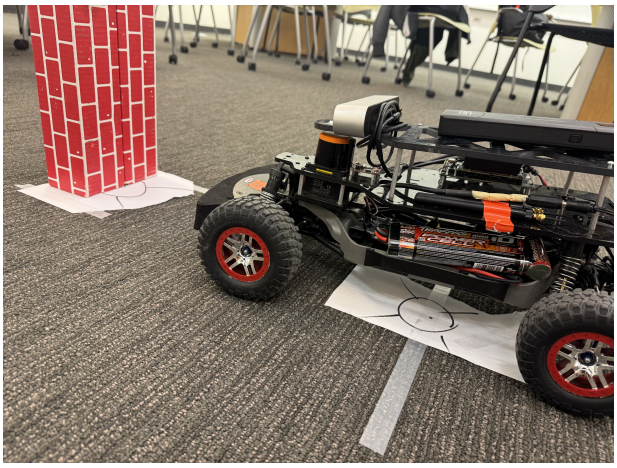

Figure 13: *We measured the distance from the center of the LiDAR to the point on the ground to see how far it overshot.*

Set Velocity (m/s)	Actual - Expected (cm)
.5	−8.5
1	15.5
1.5	39.5
2	34.5

As you can see, the error remains pretty consistent across different set distances, hovering between 14.5 cm to 19.5 cm. This data suggests that the error is mainly due to the car's momentum at the time of braking rather than how far the threshold is set. This also lines up with our safety controller implementation as currently, it simply commands the velocity of the car to 0 without keeping in mind the car's inertia.

The velocity experiment is a bit more complicated. At 0.5 m/s, the robot actually stopped 8.5cm before the set distance. We theorize that this is because the car's battery was running low and had issues with stalling at that speed. But from 1m/s onwards, it is clear that the car kept overshooting at greater errors in a trend.

Overall, the data confirms our guesses that the safety controller's lack of a velocity-aware controller would cause it to overshoot significantly. For this lab, this overshoot was acceptable as the robot was usually going at 1 m/s, so an overshoot of 15–19 cm is acceptable. However, for future labs, this safety controller would likely need to be worked on to handle higher speeds.

Future Work (∼150 words) [Nathaniel]

There are a few areas where both the wall follower and safety controller could be improved going forward.

For the wall follower, the current RANSAC implementation detects wall points but does not distinguish between a true continuous wall and scattered points that happen to form a line. Improving this to specifically identify solid wall surfaces would make the follower more reliable around corners and in cluttered environments. The code could also be better organized by moving tunable values into ROS parameter files and creating proper launch files for testing, which would make it easier for the whole team to run and modify.

For the safety controller, the biggest improvement would be incorporating the robot's current velocity into the stopping logic. Right now it simply commands a stop without accounting for how fast the car is moving, which is why we see consistent overshoot in testing. A velocity-aware braking model would give much more accurate stopping distances at higher speeds.

Conclusion and Key Takeaways (∼150 words) [Nathaniel]

Over the course of this lab, we built, validated, and learned from deploying a wall follower and safety controller on a real autonomous racecar.

We successfully built a wall follower using a PD controller with RANSAC-based line detection and paired it with a safety controller that stops the car when obstacles come within range. Both systems were validated through testing, and the results matched our expectations. The wall follower maintained consistent steady-state behavior, and the safety controller reliably halted the robot near the target stopping distance, however, it overshot by a consistent margin due to momentum.

The main thing we learned is that real hardware introduces challenges that simulation does not. Phys-

ical inertia, sensor noise, and real-world friction all have a meaningful effect on system performance. Going forward, these findings will directly inform how we design controllers for future labs, particularly around making the safety controller velocity-aware.