Developing Algorithms to Race Within Lanes and Navigate Through Stata Basement

Introduction

In this report, we detail our approach to developing both (1) an algorithm for racing the racecar on a 200m lap of the Johnson racetrack and (2) a navigation algorithm for navigating through the Stata basement while completing tasks and obeying traffic laws.

Compared to our prior work, the algorithms developed in this final challenge were much more advanced and required careful tuning. In order for the robot to accurately detect lanes, we referred to computer vision principles when applying a series of masks and transforms to pick out the white lanes. In addition, we tuned the PD controller from when we implemented wall-following to improve the robot's driving. For the second challenge, we overhauled the path planning developed in the path-planning lab for both (1) efficiency and (2) addressing the new "right side of road" constraint. We also applied color segmentation for stopping the robot at stop-signs as well as improved trajectory following.

The result from this final challenge is an autonomous system capable of racing within lanes and efficiently navigating known environments while adhering to specified constraints and traffic laws. We will focus on the modifications and improvements we made to our previous code as well as how we integrated everything in this report.

Mario Circuit

Technical Approach:

Our implementation is split into two modules: lane detection and steering controller. In the **Initial Setup**, we go over the relevant data acquired from the racecar used by the implementation. The following **Technical Approach** sections explains the theory behind the implementation and clarifies how each part contributes to the module purpose. Finally, the **ROS**

Implementation section illustrates specific design choices we made as well as their underlying motivations.

Initial Setup:

The ROS node for lane detection subscribes to live RGB image from the onboard ZED camera through the "/zed/zed_node/rgb/image_rect_color” topic. After analyzing the image and finding the endpoint for the left lane, the point is published to "/left_lane_endpoint”. The homography transformer subscribes to the left lane endpoint topic and the endpoint is put through the homography transform to be published to the “/left_lane_homography_endpoint” topic. The PD controller node for steering subscribes to that topic and after calculating the appropriate steering angle, it publishes a drive command to "/vesc/low_level/input/navigation”.

Lane Detection:

The goal is to detect lanes in the images received periodically from the camera. Figure 1 is one such image that the racecar receives—and we quickly realize directly applying line detection will not work due to the other objects present in the image.

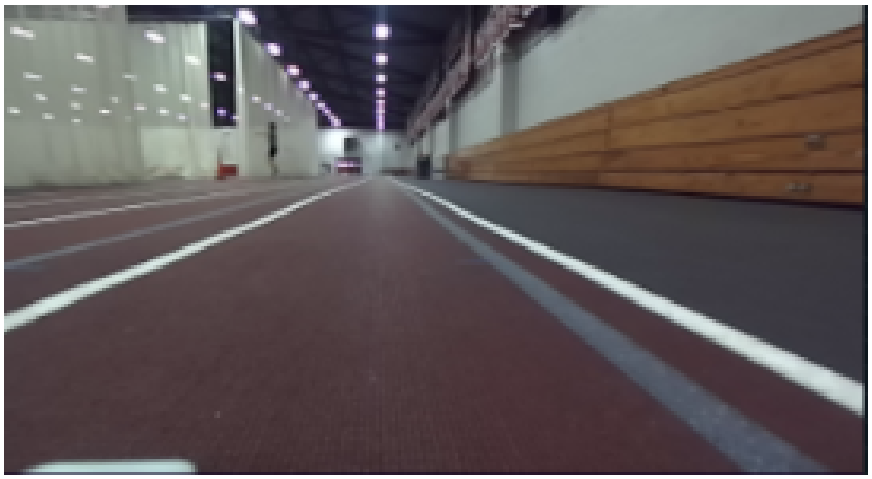


Figure 1: The image received from the ZED camera include a lot of extraneous noise, which requires additional masks to reliably segment the lane lines.

As such, we begin by cropping the top half. While a trapezoid mask was considered, the crop inadvertently covered the lanes for certain angle views such as during turning. We perform a simple crop to focus the attention on the bottom half as all the necessary lane visuals are there.

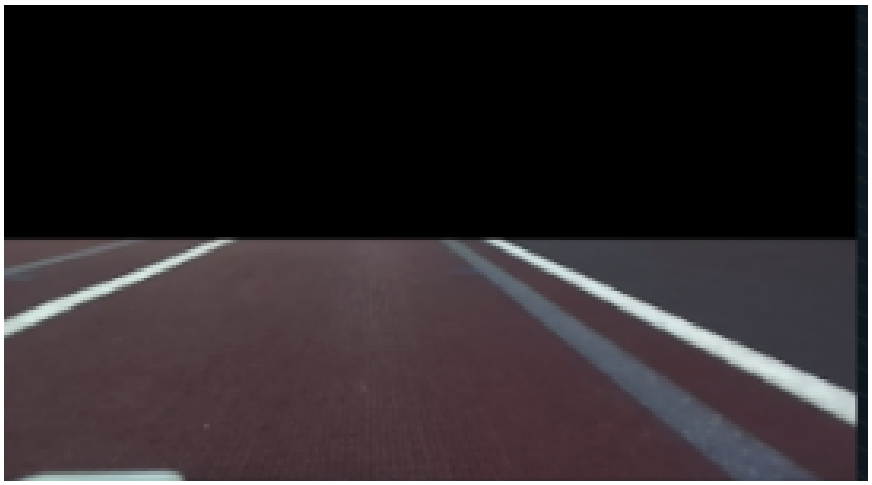


Figure 2: After cropping the top half out to remove top noise, there are much less distractors that could mess with the color segmentation performed later.

The image is then converted from the BGR to HSV (Hue, Saturation, Value) color-space. In particular, using the Hue component decreases the sensitivity to lighting variations. The lighting at various points on the track varied drastically, justifying this additional step. The resultant image after filtering out colors within the (0, 0, 150) and (180, 30, 255) thresholds for white color segmentation very clearly outlines the lanes.

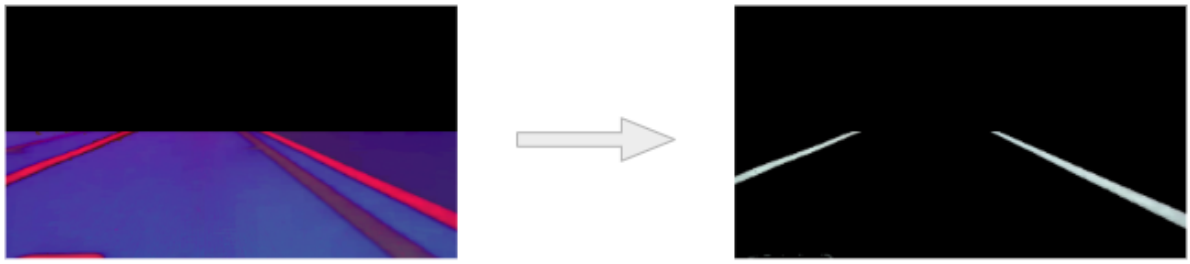


Figure 3: The converted HSV color-space image is then filtered for white. The resulting image is only of the lanes with no additional noise.

A canny edge detector is then ran on the image to extract the useful lane edge information. The algorithm identifies pixels with high intensity gradient values as edges and tracks edges with hysteresis.

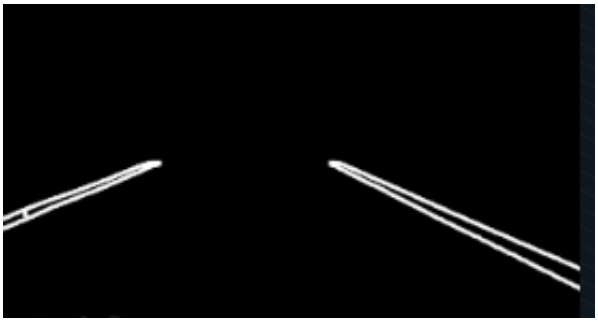


Figure 4: Edge detection provides the image pixels that outline the lanes in the image.

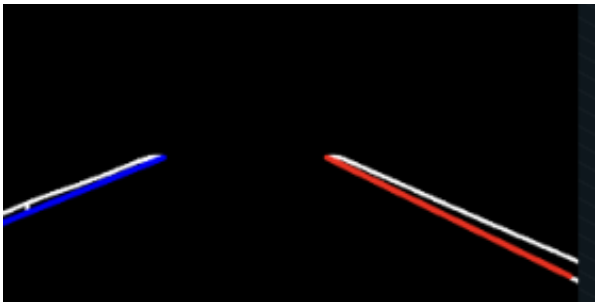


Figure 5: The Hough transform and subsequent slope filtering properly identifies the left lane, which is the reference the racecar uses to follow.

A Hough transform is applied on the resultant image to identify lines corresponding to lanes. We then filter out the lanes by slope and take the max length line, which properly identifies the left lane.

PD Controller:

We applied wall-following logic in our PD controller for steering: instead of a physical wall, the robot treats the left lane as the “wall” it needs to follow. While the robot will not crash into something physical if it touches the lane, we want it to remain a desired distance away nonetheless while it is driving so there is enough space between it and the lanes.

To begin, we apply a homography transform on the original image with the left lane reference point in order to find the actual distance from robot to the left lane.

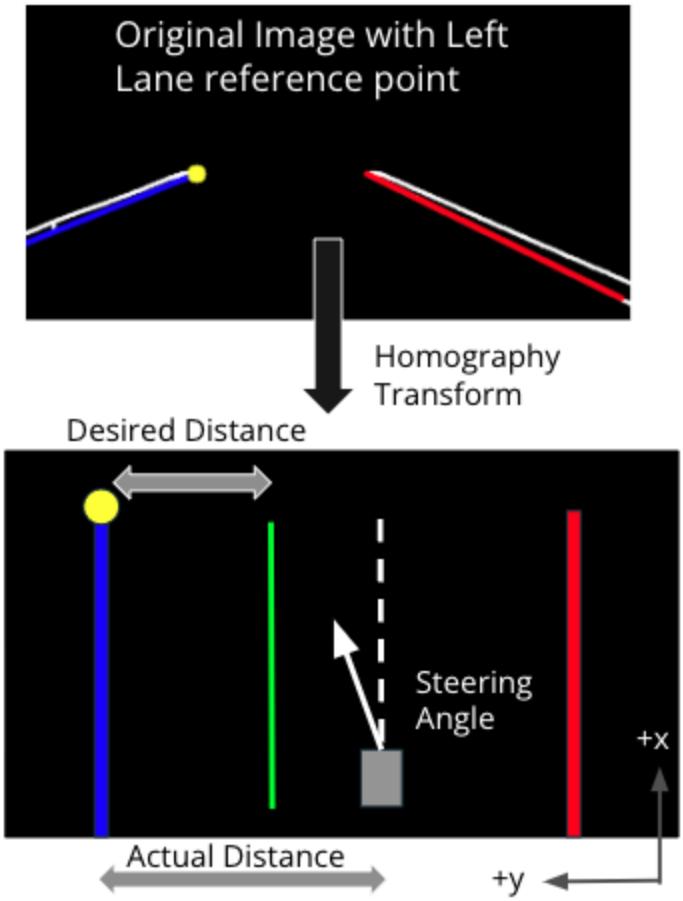


Figure 6: Applying a homography transform allows us to determine the distance of the racecar from the left lane in meters. After calculating the actual distance, the PD controller steers the robot toward the desired distance by finding the steering angle.

With the distance, our PD controller compares it to the desired distance, finds the steering angle through trigonometry as illustrated in Figure 7, then publishes the drive command.

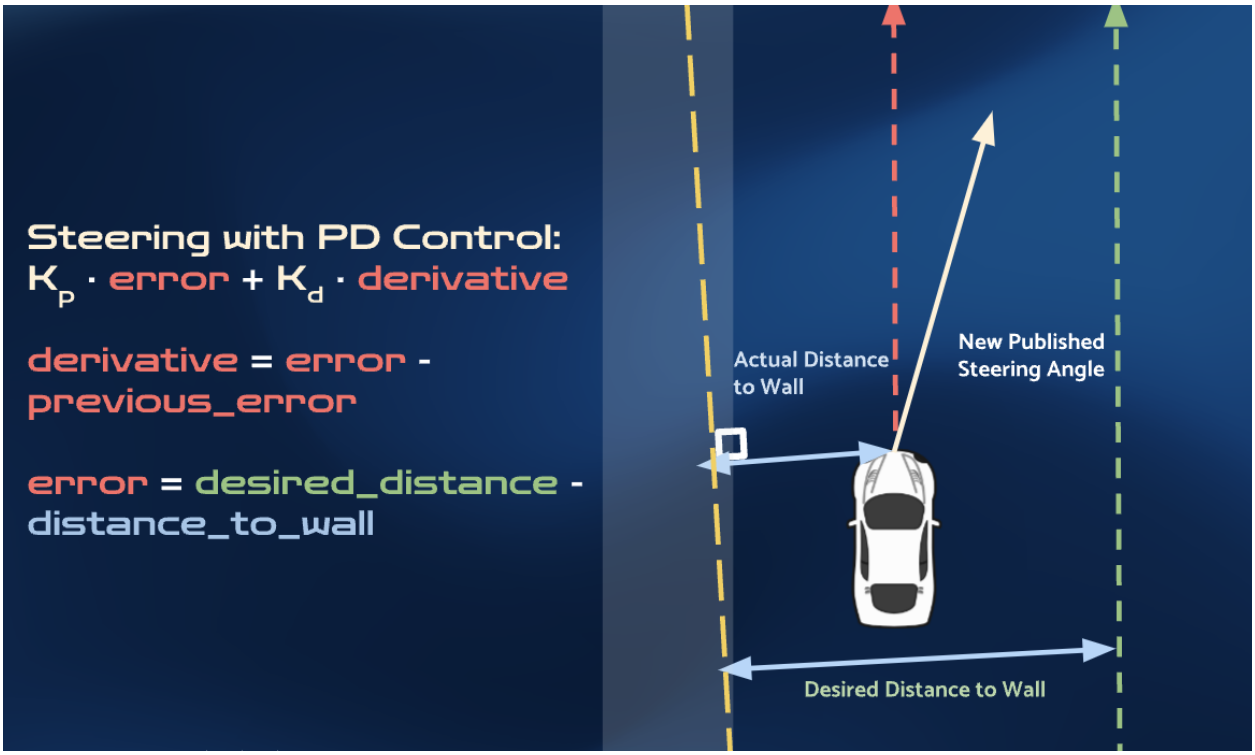


Figure 7: The PD controller for this final challenge works very similarly to our controller from Lab 3: Wall Follower. The only difference is that instead of a wall detected from LIDAR scan, we are using the lane we detected from the racecar camera.

ROS Implementation:

The Lane Detector analyzes the image from the "/zed/zed_node/rgb/image_rect_color" topic to identify the lanes the robot is within. The image is transformed to its HSV color-space and color segmentation is performed for white color ranges. After the color segmentation, a Gaussian blur is applied to reduce the amount of the noise on the image. Particularly, the large width and grainy texture of the lanes can cause the Canny edge algorithm to also draw edge pixels for the inner lane content. The edges are created after the Canny function which are then dilated in case too much information was removed due to the blur. The Hough transform receives the resulting image to create a series of lines which are filtered by a slope threshold number. This is to remove edge cases such as outer lanes, and horizontal/slanted lines. Additionally, the slope is split between negative and positive to determine the left and right lines. The lines with slopes that pass the threshold get added to a left or right lines array depending on its sign.

Originally, the left and right lines array were averaged together to obtain the left and right lane lines. However, the resulting lane lines could be inconsistent and jump around as the car moved. In order to address this issue, two lines, which had the max length of the corresponding left and

right line arrays, were picked as the lane lines. The max length check is used as an additional filter for slanted lines which pass the slope threshold and edge cases such as numbers. The two lines are then extrapolated to the height of the cropped image for consistency, and the endpoint of the left lane line is published to the "/left_lane_endpoint” topic.

The homography transformer subscribes to this topic, transforms the endpoint, and publishes the result to the "/left_lane_homography_endpoint” topic.

Our original implementation revolved around a Pure Pursuit Controller used to drive towards a lookahead point created by the average of the endpoints of the two lane lines. The Pure Pursuit encountered many failures on the racetrack, however. The likely reason was its reliance on the lookahead point which could be inconsistent due to it jumping around. Low camera frame rate, the lanes moving around, and losing track of the lane as the robot sped up could cause incorrect steering angles or slow responses from the robot.

Swapping to a PD controller which followed the left lane endpoint allowed for more margin of error. As only one lane was needed, losing track of the right lane did not affect the controller. The PD controller subscribes to the "/left_lane_homography_endpoint” topic and uses this point to calculate the steering angle. If the distance from the left lane is too far or too close, a corresponding constant steering angle is outputted to fix the issue; else, the steering angle is controlled by the PD equation and clipped by pi/120 to prevent too much oscillation from the robot.

Experimental Evaluation:

Comparing the top speed, time, and lane infractions between our Pure Pursuit and PD controller demonstrates the difference in effectiveness between the two control systems.

Controller	Top Speed (m/s)	Time (s)	Lane Infractions
Pure Pursuit	2.5	77	3
PD Control	4.0	52	0

The PD controller out performs the Pure Pursuit in every category with a 60% increase in top speed, 32% decrease in time, and no lane infractions/swaps.

Additionally, the PD controller’s error term which is the distance from the lane minus the desired distance was published during the racetrack run to measure its performance.

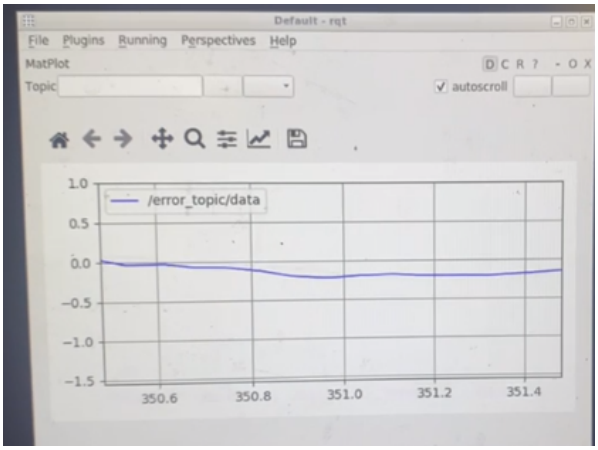


Figure 8: During straights on the racetrack the error term is relatively low and constant

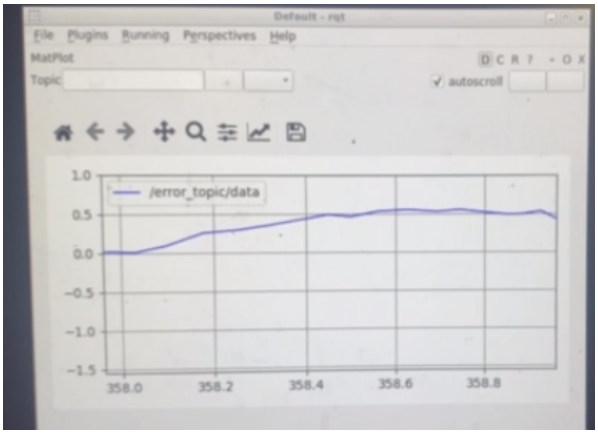


Figure 9: When the racecar is turning, the error term rises and increases

The error term average over the whole run was 0.0637 meters so overall the racecar did not deviate much from its desired distance.

Luigi's Mansion

Technical Approach:

Our implementation for this half of the challenge can be abstracted into three main modules: (1) Lane Follower, (2) Stop Sign Detector, and (3) U-Turn Handler. While our actual code is performed by two nodes corresponding to Lab 6: Path Planner, city_trajectory_planner and city_trajectory_follower, we implemented many optimizations and additions that improved performance as well as handled the new constraints in these three modules. Lane following is handled in the planner, stop sign detection is done in the follower, and u-turn handling is done in both.

In the **Initial Setup**, we go over the inputs the racecar receives for this implementation. Similarly to before, the **Technical Approach** section will explain the algorithms used. The **ROS Implementation** section will illustrate specific design choices we made.

Initial Setup:

The Trajectory Planner is initialized with city_trajectory_planner.py, and the node subscribes to the occupancy grid map, intermediate checkpoint poses, goal pose, and initial pose topics,

enabling it to receive environment information and user-specified navigation goals.

The Trajectory Follower, initialized by executing the city_trajectory_follower.py script, subscribes to the odometry data from /pf/pose/odom, LaserScan data from /scan, and the PoseArray representing the planned trajectory from /trajectory/current it receives from the planner.

Lane Following:

Although there were several different ways to handle the lane following constraint, we chose to handle this within the path planner by creating a directed edge graph representation of the map. To achieve this, we utilize a function which determines the potential movement directions for each pixel, which is how we generate our directed graph. The function calculates the possible cardinal directions based on the orientation of each cell relative to the closest lane line, allowing the possible cell directions to only be forward. By determining the cell orientation with respect to the lane line, we ensure that the graph representation reflects the constraints imposed by the lane. This allows for efficient path planning while adhering to the specified lane-following behavior.

This method also allows for a very precise path to be plotted without introducing additional potential error to the path planner or trajectory follower. While it does incur an additional cost when running the A* algorithm from the more complex neighbor checking, we believe the added precision would be well worth it.

To implement these changes, we went through a full rewrite of the path planner algorithm from our previous lab. Figure 10 has a visualization of this directionality.

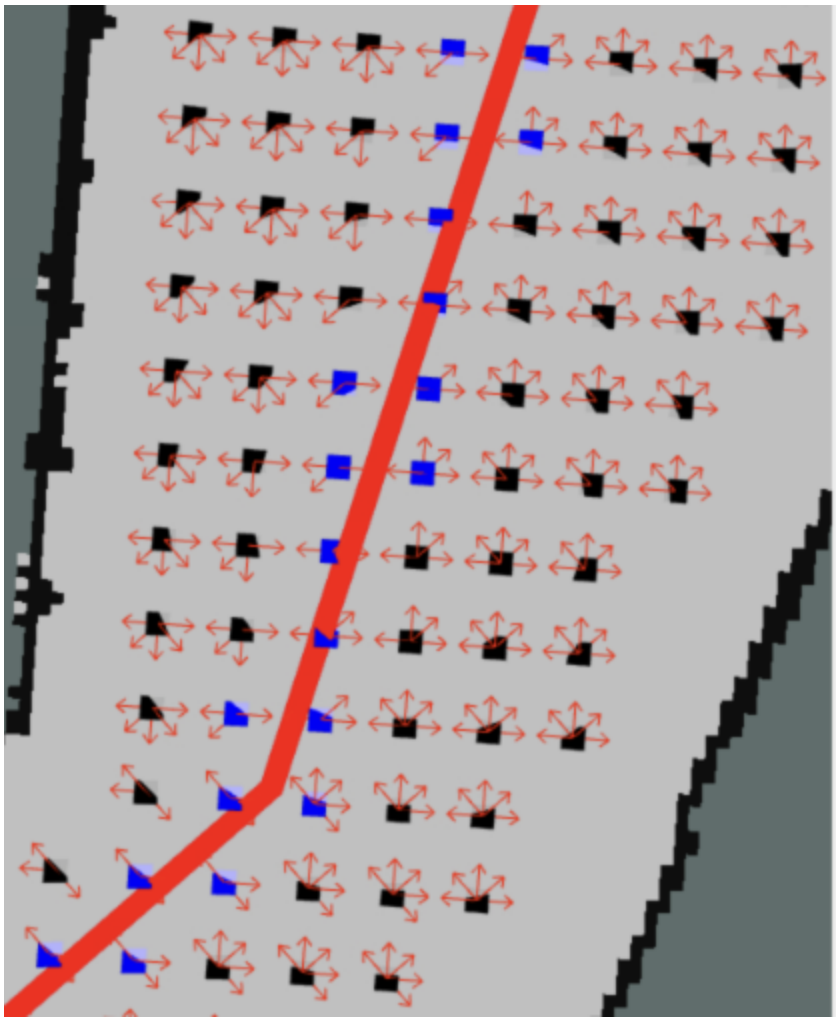


Figure 10: Each grid cell is given a direction based on its cross-product to the nearest trajectory lane segment, and each grid cell can only traverse to neighbors in front of it in the chosen direction. This ensures that A* only travels in one direction in each lane, enforcing the lane driving constraint.

In addition, we now precompute all the neighbors of each cell in the grid and keep it in a .pickle file to speed up the computation of the A*. We also lowered the resolution of the map to speed up the computation time of A* on the actual robot, as we found that the A* algorithm computes significantly slower on the physical robot.

U-turns:

The last important portion of city driving was allowing our robot to execute u-turns, or in our case, 3 point turns.

First, we had to ensure we would create a path within trajectory planner that would initiate a u-turn. We implemented a function to check whether a cell was close to the lane and allow it’s

possible directions to only be left and right—this allows the car to cross over lanes if needed while avoiding a scenario where the car is traveling on the lane line. The method used to handle u-turns is visualized in Figure 11.

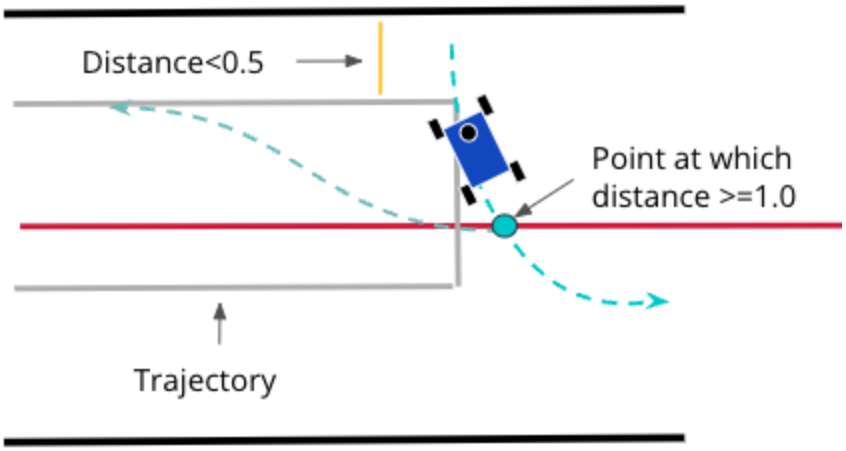


Figure 11: A visualization of how the trajectory follower handles u turns by doing 3 point turns. The robot initially reverses until hitting the position where its distance from the wall is greater than or equal to 1.0 then continuing to travel down the planned path.

With this updated trajectory planning algorithm, however, the u-turns being created in the trajectory planner were too sharp for the car to handle, causing it to drive past the wall in simulation. Thus in the follower, we added a function to initiate a k-turn or a 3 point turn if it was getting too close to the wall. Within this function, we first detect if the robot is too close to the wall using LIDAR scans: the robot is “too close to the wall” if its distance from the wall was less than 0.5. Then we would make the car reverse by setting its speed to -1.0 and its steering angle to be the max angle (0.34) and the opposite direction of whichever way it was initially turning. This sequence of actions effectively handles U-turn detection and reversing, allowing the robot to navigate through tight spaces or obstacles. It also doubled as a way for the robot to correct itself if it veered off-course in a significant way. We thus tracked the number of u-turns as an error metric in our evaluation of the trajectory follower.

Lookahead Point:

While encoding the u-turn detection and functionality, we realized that the lookahead point would often become a point not on the trajectory path whenever the robot’s position was near a corner or sharp turn in the path. To fix this we parameterized the line segment P1 to P2 by t, seen in figure 12. Any point between P1 and P2 must have a t of between 0 and 1.

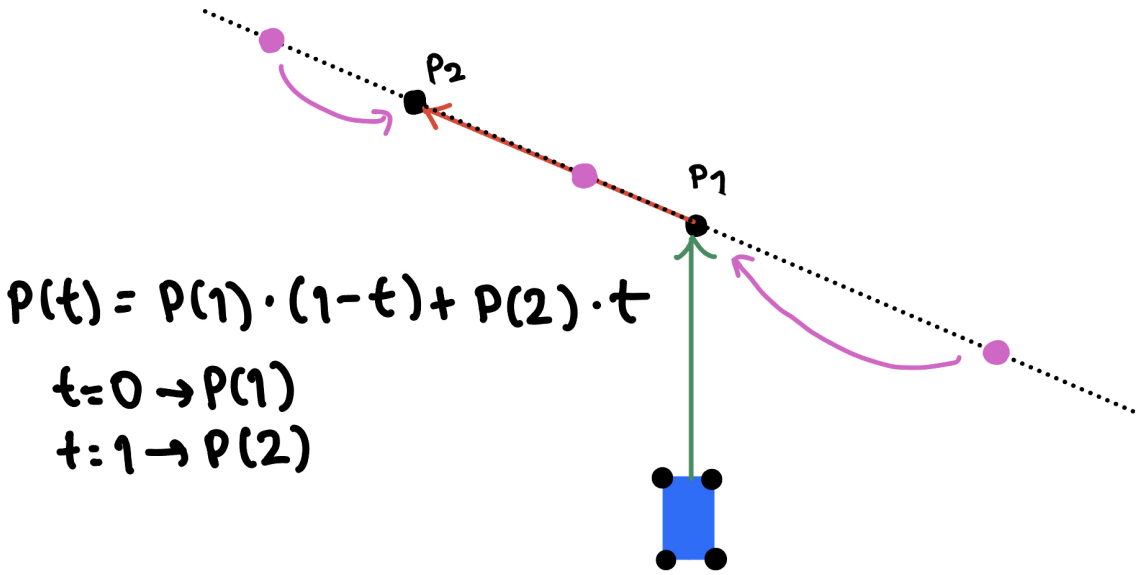


Figure 12: We clamp t to always be a number between 0 and 1, thus ensuring the projection point to always be a point between P1 and P2. This makes sure our lookahead point will always be a point on the trajectory path.

As previously stated, we run into issues when the P1 and P2 are vertically stacked corresponding to when the path is at a corner. To fix this, we clamped t: while looking at the line defined by P1 and P2, we set our lookahead point to P1 whenever our point ends up with a t less than 0, as that would indicate a lookahead point closer than P1 (not on the path). On the other hand, t greater than 1 indicates a point past P2, so we set the lookahead point to P2. Guaranteeing the lookahead point is always on the trajectory path results in better trajectory following.

Stop Sign Detector:

The stop light detector constantly watches the output from the camera, and it publishes a warning message to the trajectory_follower when a stop sign is detected. Since the detector is constantly running, these warning messages will be published constantly once a stop sign is detected. In order to prevent the robot from stalling forever when it sees a stop sign for the first time, we use a state machine to tell the trajectory follower when it can safely ignore these warnings.

This state machine has 3 states: Ready, Detected, and Ignoring. In the Ready state, the robot is actively looking for the stop sign. If it detects one within a certain distance, it will transition to the Detected state and come to a complete stop for 5 seconds. After that, the robot transitions to the Ignoring state for 5 seconds, where it will ignore stop signs even if it sees them. Following this FSM allowed the robot to stop for 5 seconds at stop signs when it saw them.

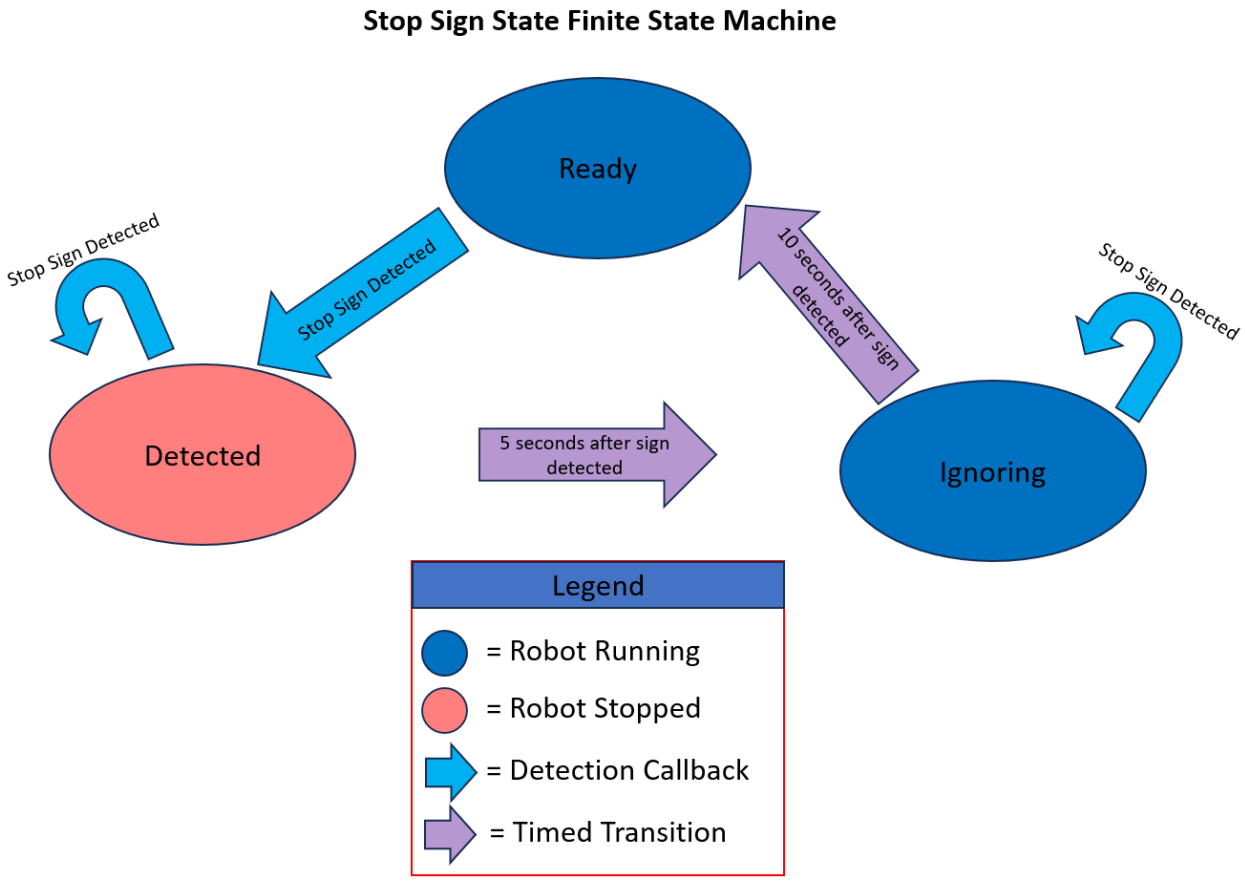
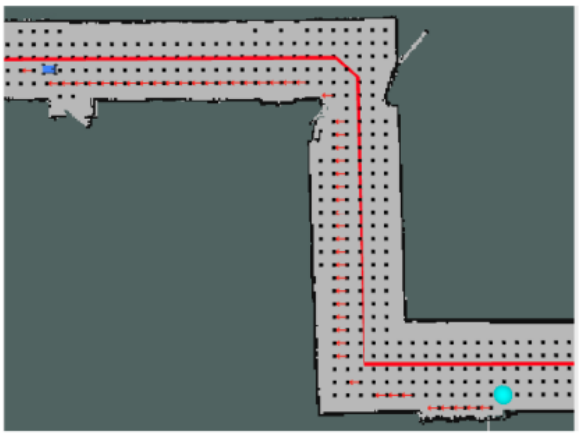


Figure 13: The trajectory follower uses this Finite State Machine (FSM) to only stop at a stop sign for 5 seconds once every 10 seconds. Since the stop sign detector will constantly be checking for stop signs and, therefore, constantly publishing detection messages when a stop sign is found, the trajectory follower needs an FSM in order to determine whether or not to ignore these messages.

Unfortunately, the robot missed the stop sign during our Part B run, as the stop sign was placed too high on the pipe for the robot's camera to see it. While we have video proof of the machine learning algorithm working, the stop sign never appeared in the robot's camera and so was never detected.

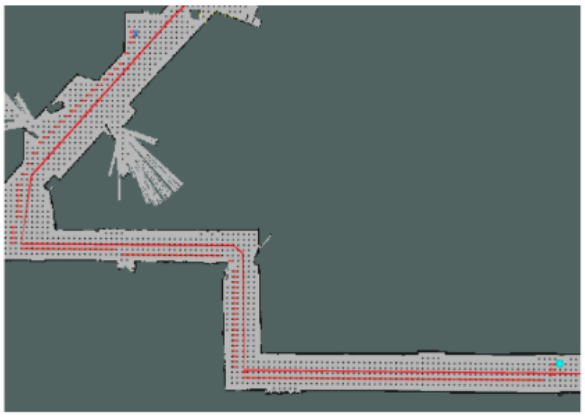
Experimental Evaluation:

In order to verify the optimality and efficiency of our new A* algorithm, we ran both our current path planning algorithm and the path planner we used in our previous report. The results demonstrated that our new A* algorithm is both more optimal and far, far faster, as can be seen below.



Algorithm	City Driving Path Planner	Original Path Planner
Time 1	0.16556	0.247325
Distance 1	19.82042	19.48280

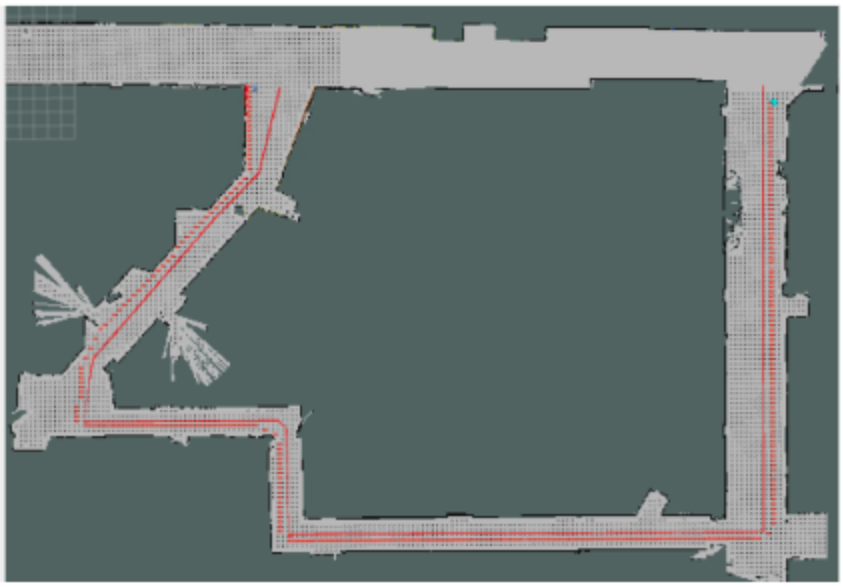
Figure 14: The time it took for the improved path planner and the original Lab 6 planner to calculate a trajectory for two points close together is not noticeably different. The same is true for the distance of the "shortest" path.



Algorithm	City Driving Path Planner	Original Path Planner
Time 2	0.11064	7.9075
Distance 2	60.92725	63.23998

Figure 15: Now slightly further apart, the difference in computation times is significantly noticeable. The quality of the path itself is also better.

Figures 14, 15, and 16 demonstrate that paths from our new path planner are shorter and calculated far more quickly compared to the path planner from our navigation report. The city driving path planner is much faster thanks to pre-computation of the neighbors of each cell, and downsampling of the graph.



Algorithm	City Driving Path Planner	Original Path Planner
Time 3	0.22337	16.04974
Distance 3	118.65269	156.80608

Figure 16: With two points very far apart, it is very clear the impact our improvements had on the algorithm. Not only is the computation time significantly faster—a 0.22 second computation allows the racecar to essentially drive off immediately while our previous version would have had a long wait— but the path it generated itself is also significantly better.

Surprisingly, it also finds paths that are shorter in length compared to the original path planner, likely due to the lanes preventing “zig-zagging” paths that plagued the original path planner.

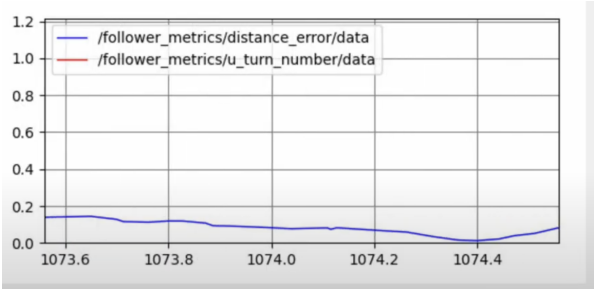


Figure 17: Error is relatively low and constant when the racecar is following a straight-line.

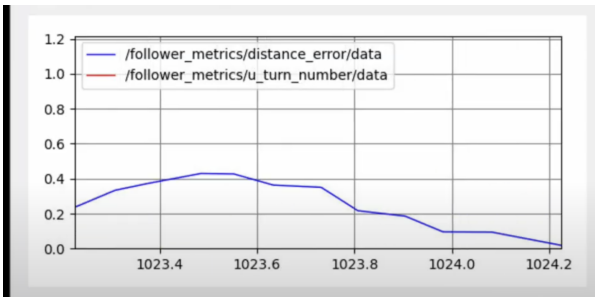


Figure 18: When the racecar needs to make a u-turn away from a wall, the backing-up is reflected in a temporary increase in error.

We also measured the distance error of the robot's location to the planned trajectory to evaluate the trajectory following. We used the robot's odometry data to estimate its current location and calculated the distance to the closest planned trajectory Pose. Overall, we noted the average was low at 0.2 when the racecar was following a straight line and the error goes up when it needs to make a u-turn to steer away from a wall if it is too close. However, as the racecar corrects itself and follows the trajectory again, the error goes back down.

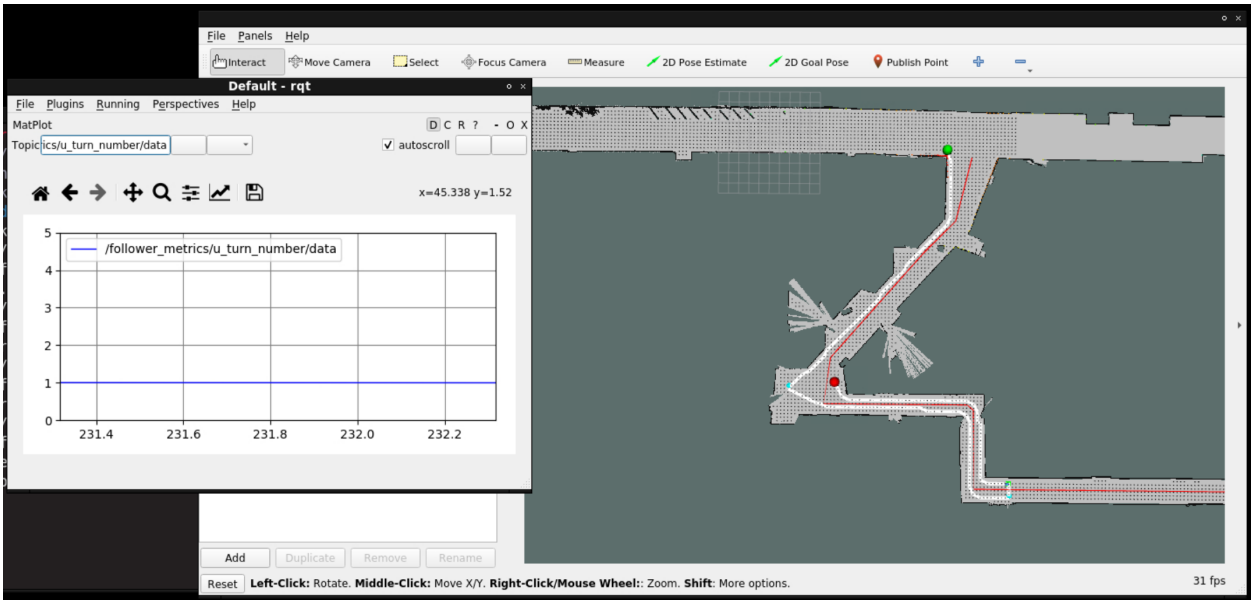


Figure 19: In simulation, the robot only performed a u-turn routine once, when switching lanes from the 2nd point to the 3rd point. On the actual robot, the robot's slower processing meant that the trajectory follower was more unstable and required many more u-turns to correct itself. The u-turn metric is thus an effective way to determine if the robot is following the path correctly by seeing if the robot has to excessively call the u-turn method.

Another metric we use to measure the effectiveness of the u-turn handling is by checking how many u-turns the robot needs to execute to continue driving along the planned trajectory. In simulation, the robot only needed one u-turn, but the physical robot had to use several more to correct itself. The physical robot frequently veered off course due to a much higher error in the trajectory follower from a combination of slower processing, more error-prone odometry messages, and mechanical error.

Conclusion

The two parts of the final challenge required revising and modifying previous algorithms from Lab 3-6. Notably, Part A: Mario Circuit relied heavily on elements from Lab 3: Wall Follower and Lab 4: Visual Servoing. Part B: Luigi Mansion, on the other hand, is primarily built upon Lab 5: Localization and Lab 6: Path Planner while using computer vision to detect stop signs. While both parts were developed separately, we followed a modularized approach in integrating the past algorithms to achieve autonomous racing and navigation.

As the final challenge is more open-ended than previous labs, there were many possible approaches that could be used to complete it. As such, we planned much more carefully and iterated more throughout the implementation process compared to before. In retrospect, this was incredibly useful as the discussions we had led us to consider more of the benefits and drawbacks.

Throughout the final challenge, we learned to address different challenges that were presented by the hardware. When testing our Mario Circuit pursuit controller on high speeds, we realized that the frame-rate on our camera necessitated a more robust algorithm. Likewise, when we first tested our trajectory planner implementation on the physical robot, it took over a minute and a half, as compared to taking roughly 5 seconds in simulation. This resulted from limited memory and CPU on the robot as compared to our docker images, and we had to resolve it with some clever optimizations. Addressing these kinds of optimization issues is crucial to advanced engineering.

Looking back, the final challenge was an extremely stressful but rewarding project. We are greatly appreciative of the TAs, as well as the instructors, for offering as much guidance as they could throughout the process. We also want to thank Mary for providing useful feedback on the briefing/report as well as working through team dynamics. For most of us, this was the first time we have worked with robots and it was a large learning curve. Over the course of the 4 labs and final challenge, we learned a lot about not only robotics but also working as a team.

Personal Reflections

Nicholas:

In reflecting on our final challenge, I'm learning how to strike a balance between trusting teammates to work autonomously while also ensuring that everyone stays on track and supports each other when needed. I acknowledge that this was something that my teammates were concerned about, and I wanted to try a looser approach in working alongside them. It worked out

fantastically well, as can be seen by the independence at which we were able to accomplish our parts of the project!

One unique aspect of our final challenge here was the challenge of organizing a much larger project and integrating all the pieces from past projects together. It required careful coordination to ensure that each node fit seamlessly into the overall algorithm. It was quite fun getting to work with our past projects together as we did!

Another obstacle we faced was finding ways around hardware limits with our robot. We had to find ways to persevere by optimizing our algorithms and exploring innovative solutions to precompute data, or even using different algorithms entirely. This experience underscored the importance of adaptability and creative problem-solving in overcoming technical challenges.

Moreover, debugging proved to be a significant aspect of our project. We often found ourselves needing to debug independently due to time constraints preventing us from developing comprehensive tests. Instead, we devised simple checks and visualizations to quickly verify things were working as intended or to use them as tools to identify issues. This ensured that our project progressed smoothly.

Overall, this class has been a very intense yet valuable learning experience. I've gotten a lot of personal experience working out extremely onerous issues in a comprehensive manner. Such is robotics! I've also learned how to trust and work through the surprisingly delicate process of gelling with a randomized team and learned that building trust and rapport is often a very separate process from simply trying to accomplish a goal. So long, and thanks for all the fish!

James:

Having a effective list and distribution of tasks was very useful in completing the final challenge in a timely and efficient manner. The racetrack portion of this final challenge taught me the process of exhausting and iterating on possible options for the solution and the challenges of hardware issues. We started the challenge with a pure pursuit controller and a image processing function that had numerous issues. Then we started iterating on thresholds and masks to create a more robust image processing function. Additionally, finding out about clipping the max steer angle helped the pure pursuit reach a lap with speed 2. However, there were still struggles with higher speeds so we tried experimenting with more different parameters with no successes. This led us to change to PD control and still failing which left us to conclude that the camera was the main limiter on our performance. Finally, overclocking the camera gave us a working racetrack controller. The final challenge was extremely stressful, but rewarding at the end.

Autumn:

In regards to this lab specifically, as I look back I wish I could have been more direct with my team members and more communicative when I thought something was wrong. In all honesty, I find it hard to strike a balance between being open and honest while not being too direct. During this lab I think I held back a lot more than previous labs and I regret that.

On a more technical level, I enjoyed this lab a lot more than previous labs because it had more creative aspects to it, as the instructions were much more vague. I also enjoyed having so much to do because it made splitting up work between team members a lot easier so it was nice to have something only I was working on. The negative side to having so many parts and all of them being of the same importance, if one person didn't do their part, another would have to pick up the slack and there would be a lot of slack to pick up.

Overall this was such a difficult but rewarding lab (I feel like I say this about every lab but in comparison to the other labs this was a monster of a lab). This class did an amazing job of showing what it's realistically like to work on a team. After taking this class I don't think other class emulate the high stress environment and complex projects that most of us will be doin in the future, so all in all I learned so much from this class.

Michael:

As I look back on this final challenge (and the class as a whole), it has become clear how beneficial and valuable this class has been. Technical expertise and active communication must exist simultaneously amongst your team, or else there would be challenges. Honestly, I think the structure of RSS did an incredible job simulating a realistic teamwork experience, which I find incredibly important. Naturally, there were various challenges amongst the team throughout the semester, but that is how it will be in the real world too. Learning how to effectively resolve these challenges while still preserving the team dynamic is such an important skill, and I think our team came a long way (both as students and as teammates) throughout the semester.

Focusing more on the final challenge specifically, James and I worked on Part A - the final race. In hindsight, I think I drastically overlooked how challenging the debugging would be on this component of the lab. When working with code on the actual robot, it can be especially difficult to pinpoint the exact cause of a bug. Specifically, we struggled to control the robot at high speeds, and it was unclear whether this was a hardware or a software issue. Many attempts to resolve the issue were unsuccessful, but in the end, we determined that it was a mixture of both. We improved the camera performance and also switched from pure pursuit to PD control to improve the robustness of our controller. After many days of debugging and finally

implementing these changes, we were able to get a perfect lap at speed 4.0 m/s. We put so much time into this final challenge, so finally getting it to work was genuinely one of the highlights of my semester. It was so rewarding to see that all of our hard work finally paid off in the end.

Lennie:

This final challenge was daunting as it required integration of everything we have implemented already in Labs 3-6, and figuring out how to even start was not easy. Isolating and identifying errors in the code was pretty difficult as so many separate components need to come together. However, it was mitigated by printing and visualizing components in each part which helped us isolate bugs. I regret not contributing more to the coding in the lead-up to the final challenge as well as not communicating my busy schedule. To that, I really appreciate my teammates for stepping up. To make up it, I tried to do more of the post-challenge day work which included the briefing slides and report and debugging of Part B.