# Synthesis of Speed and Adaptability: Autonomous Racecar Performance in Track Racing and City Driving Challenges

Final Challenge

Team #11: Stattatata

Timber Carey
Marcos Espitia-Alvarez
Subin Kim
Amy Shi
Christian Teshome

6.420 Robotics: Science & Systems

May 13, 2024

## 1 Introduction

***Subin Kim***

The Final Challenge for the RSS represents a critical culmination of a semester's dedication to mastering the complexities of robotics. In this final lab, students are tasked with integrating a broad spectrum of skills and knowledge acquired through the course to address two distinct yet equally challenging tasks: Mario Circuit and Luigi's Mansion.

Mario Circuit demands precision and control as students navigate a high-speed race along a defined path on the Johnson Track. The primary goal is to maneuver a car at optimum speeds without straying from the track, requiring a delicate balance of speed and stability. Our team's approach includes the use of sophisticated controls systems like the Pure Pursuit Controller. We have also experimented with various image processing techniques such as Birdseye view transformations for lane detection and DBSCAN clustering for line segmentation. These techniques are effective in applications of real-life scenarios, especially in the context of competitive racing where it is crucial to manage

high-speed dynamics while adhering to racing lines.

Luigi's Mansion simulates an urban environment where the vehicle must navigate complex city-like infrastructure while obeying traffic laws. This challenge emphasizes advanced path planning as well as adherence to traffic regulations, incorporating path planning and localization to adjust the vehicle's trajectory based on detected road markings and traffic regulations. Our application of machine learning for detecting stop signs and employing color segmentation to recognize traffic lights are also implemented for practical applications of urban autonomous navigation.

The significance of this challenge extends beyond the parameters of real-life scenarios for autonomous vehicles. This not only tests students' technical abilities in developing and tuning parameters for speed and optimization, but also their capacity to troubleshoot and refine these solutions.

Testing and troubleshooting are fundamental components of this challenge, which pushed our team to iteratively refine our approach based on real-time feedback and performance data. This iterative process underscores the need for flexible adaptation, including the ability to revert to previous software states, fine-tune the controls parameters, and integrate comprehensive functionalities of all previous labs together that represents the dynamic and complex nature of the final-challenge applications.

# 2 Technical Approach

***Subin Kim***

For the team's final challenge, we put together all the techniques we learned throughout the semester in RSS to tackle the demanding tasks of Mario Circuit and Luigi's Mansion. For Mario Circuit, our approach centered on lane detection and precise high-speed navigation. Luigi's Mansion integrated path planning algorithms and traffic constraints, in addition to localizing of the Stata basement.

## 2.1 Mario Circuit

### 2.1.1 Lane Detector

***Amy Shi***

The left and right lines of a given lane were detected by analyzing the Zed image in its given frame and utilizing Hough transforms. To start, the image was gray scaled and then masked to isolate the bottom half of the screen. The mask left a section in the shape of a trapezoid because this reduced the likelihood that the algorithm will detect the other lanes.

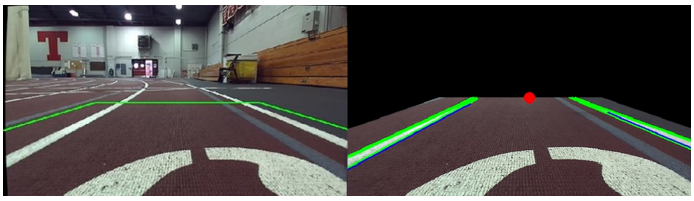


Figure 1: The original camera view (left) is masked (right) to ease filtering of the lines.

After masking the image, an adaptive threshold to detect white lines in various lighting was used as part of a bitwise-and operation on the image to turn it into a binary image. This made it easier to isolate the lane lines since the processed image only consisted of black and white pixels.

However, before we were able to apply a Hough transform on the image to detect the lane lines, we had to isolate the edges in the image. At first, we used Canny from OpenCV to perform edge detection. However, there were many missing segments on the lane lines and produced unreliable results, so we switched to using morphology functions from OpenCV. Specifically, we used MORPH_CLOSE and MORPH_GRADIENT. The MORPH_CLOSE function was able to close the gaps between lines in the image, which allowed the HoughLinesP function to better detect the lane lines. The MORPH_GRADIENT function allowed us to isolate the edges of the lines.

After all the image processing, the HoughLinesP function from OpenCV is applied on the new image to return the detected line segments within the image.

Figure 2: This is an example of a processed image that is passed into the HoughLinesP function. Only the lane edges are not filtered out of the image.

To sort the lines, we first identified the lane lines by their slopes. If the absolute value of the slope of a line segment is greater than a certain threshold, then it is considered a lane line. This allowed us to filter out the horizontal lines that crossed the lanes at certain points.

Next, to identify whether the line is on the left or right side of the lane, we calculated the average x-position of the current line and compared it to the previous left/right line position. If it was within a certain distance of the previous lines, then it is considered to be a lane line. Without keeping track of the previous state, our algorithm would not have been able to account for all the edge cases where the robot might have driven into a different lane.

Finally, to determine a single left and right line for the lane, the lines with the maximum slope magnitudes were picked. Their x-values were averaged together to determine a point along the horizontal axis for the car to follow. If there was only one side of the lane detected, we translated the x-position of the line segment to determine the following point along the horizontal axis. The y-value of the point to follow was a constant value that we determined as the lookahead distance for the Pure Pursuit controller.

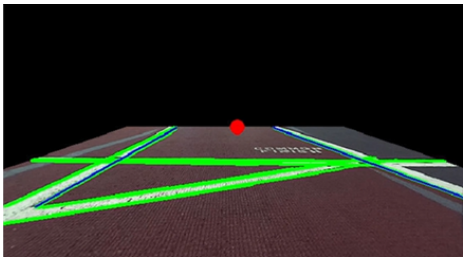


Figure 3: The horizontal lines were able to be filtered out by the sorting algorithm, and the program correctly identified the left and right lanes (shown in blue) as well as the point to follow for the racecar.

#### 2.1.2 Driving Controller

***Marcos Espitia-Alvarez***

We started with a Stanley Controller, where the steering angle is given by the equation

$$\delta = K_h \psi + tan^{-1}\left(\frac{Ke}{K_s + v}\right) \tag{1}$$

However, we ran into a consistent issue of the car being either too responsive on straightaways or needing to be more responsive on turns. This was caused by the fact that the Stanley Controller requires us to approximate the center line which may bounce around the center of the screen. Tuning gains to be sensitive to CTE (like on a turn) would cause the car to be unstable when dealing with small centerline errors on straightaways. This is also why a PD controller failed; The error calculation was susceptible to sudden perturbations of the centerline. Finally, we decided to use a Pure Pursuit controller because it only required a

look-ahead point which would not bounce around as much as a centerline.

$$\delta = tan^{-1}\left(\frac{2Lsin(\alpha)}{l_d}\right) \tag{2}$$

It was much more intuitive to tune, only needing to adjust the look-ahead distance. While the Pure Pursuit did oscillate, it did so less than the other two controllers when properly tuned. Eventually, it corrected the car to a smooth drive down the straightaways of the track lanes in spite of sudden lateral errors at the look-ahead point and was responsive to turns.

## 2.2 Luigi's Mansion

### 2.2.1 Path Planning and Following

***Timber Carey***

For the path planning and trajectory following sections of the Luigi's Mansion challenge, we used a modified version of our code from the previous path planning lab. The path planning had to be modified to account for three goals instead of one, and to plan a path that stayed in the right lane. The trajectory follower algorithm did not need to be much different, but it did have to be tuned to perform better for the final challenge.

The new path planning algorithm used cross products to determine which direction would be forward on the right side of the lane divider. Our original path planning algorithm used breadth-first search (BFS), so we only had to add some conditions on which neighboring points could be searched in each next step. For each search point $P$ in the BFS process, the point $C$ on the lane trajectory closest to $P$ was calculated by finding the closest point on the nearest segment of the lane trajectory. Then, for each neighboring point $N$ adjacent to $P$, it was determined whether $N$ lied in the correct direction to go forward on the right side by using the cross product $\overrightarrow{CP} \times \overrightarrow{CN}$.

The algorithm for determining the correct driving direction is illustrated in Figure 4 below. The line represents the nearest segment of the lane divider, point $P$ is the current point of interest in the BFS algorithm, point $C$ is the point on the lane divider with the shortest distance to $P$, and each of the points $N$ are the neighbors to potentially be added to the search path. To search only the neighboring points going forward on the right side of the lane divider (in the direction of the green arrow), we added to the search path only the neighbors that met the condition $\overrightarrow{CP} \times \overrightarrow{CN} > 0$. This ensured that the path could still cross the lane divider to make U-turns, but once it did, then it would only search in the correct direction for that side.

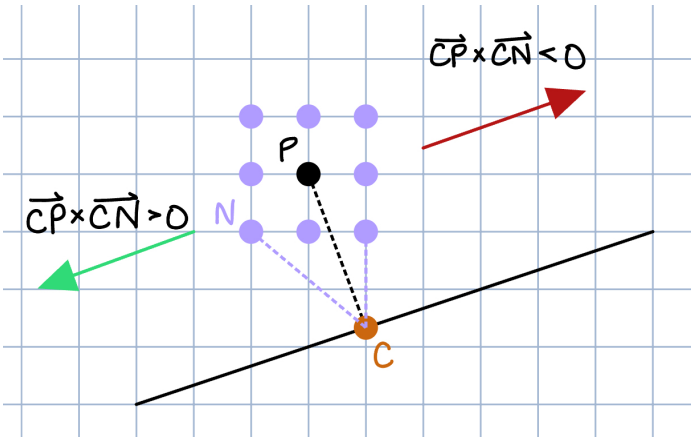
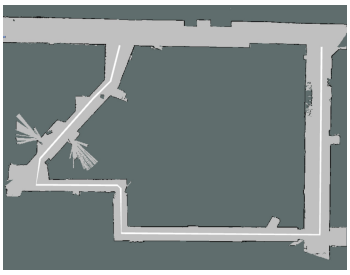
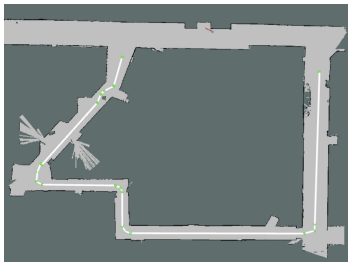



Figure 4: An illustration of the algorithm for determining which direction is forward on the right side of the lane divider. The line represents the nearest segment of the lane divider, point $P$ is the current point of interest in the BFS algorithm, point $C$ is the point on the lane divider with the shortest distance to $P$, and each of the points $N$ are the neighbors to potentially be added to the search path. Only the neighboring points $N$ where $\overrightarrow{CP} \times \overrightarrow{CN} > 0$ are added to the search path.

In order to navigate through some of the more narrow parts of the map (i.e., the spot with a pillar in the hallway), the path planner had to be adjusted to be able to ignore certain segments of the lane divider. These segments had dotted lines on the divider in real life, so they were okay to cross over in any direction. To pick out specific segments of the trajectory, we needed a lane divider trajectory that was somewhat more accurate than the trajectory that was provided to us. Using the trajectory builder from a previous lab and the real robot localization, we drove the robot to each point on the real life lane divider line and clicked on its location in rviz to get a very accurate lane divider trajectory. The provided lane trajectory and the more accurately built lane trajectory are shown in Figure 5.



(a) The provided lane divider trajectory.



(b) More accurate lane divider trajectory.

Figure 5: Visualizations of (a) the provided lane divider trajectory, with sharp corners and lines that didn't quite match the real life line, and (b) our more accurate trajectory, built using localization and the trajectory builder from the path planning lab.

Now that the path could be planned properly, the trajectory follower needed to be adjusted. The previous wall follower only accounted for a single path output from the planner, but in this challenge, the path planner would output three paths immediately after each one was finished planning. In order to save time in the final run, we modified the original trajectory follower so that the robot would start following each path as soon as it had reached its previous goal, even if the next path was not published yet. This way, subsequent paths could be planned concurrently.

The trajectory follower was not perfect in that it sometimes steered into walls and that making U-turns was difficult in the small hallways. To remedy this, we implemented a wall detector. This new node was built off of our previous safety controller, which sent a command to stop driving if anything was detected directly in front of the robot. The safety controller code was modified so that instead of stopping, the robot would back up slightly to the right. This allowed the robot to correct itself and continue on its trajectory if it ever ran into a wall on the right side.

#### 2.2.2 Stop Signs, Traffic Lights, and Pedestrian Crossings

***Christian Teshome***

In addition to simply reaching the three shells, our car also had to obey all the traffic laws in the city. In this scenario, there are stop signs, traffic lights, and pedestrian crossings. Our car was required to come to a complete stop at stop signs, stop at red lights, and never hit pedestrians.

The first step to ensuring our car follows the traffic laws is to be able to detect objects that would incur traffic infractions, such as stop signs, traffic lights and crosswalks, so that we can safely stop in time. Luckily, the wall detector discussed previously is able to handle the pedestrian crossing concerns, since

when the car sees an object in front of it, it will back up and turn to avoid hitting it. We considered using our safety controller or using a machine learning model that could detect pedestrians, however we ultimately decided that a wall detector would be sufficient and would also help us correct any slight errors with our trajectory following on the fly.

We used a pre-trained machine learning model to detect stop signs. The model takes in an image, and outputs whether there is a stop sign in the image, along with coordinates representing the bounding box of the stop sign (in pixels).We don't know where the stop signs are in advance, so the stop sign detector needs to be running at all times

In order to detect traffic lights, we used our color segmentation algorithm from the visual servoing lab and adjusted it so it would look for the color red. Specifically, it would filter based on a lower and upper bound of HSV (Hue, Saturation, Value) values. Since the locations of the traffic lights are known in advance, the detector is only run when the car is near those locations.

Once we had both detectors working, we then had to consider how to combine them and how they would interact with our trajectory follower. We ultimately decided that each of the two detectors would publish a message to its own topic, and this message would include a boolean value indicating whether the object was detected, as well as coordinates representing the bounding box of the object. We then created a separate node that would subscribe to these two topics. If it determined that the car needs to stop, it would publish a stop command to a topic of higher priority than the trajectory follower so that it could override it.

In order for the node to determine if the car should stop, it uses the following logic. First, it checks if the stop sign detector returned true. If it did, then it checks the size of the stop sign's bounding box. The size of the box is used as an approximation of how close the car is to the stop sign. If the size is above a certain threshold, it will publish a stop command for 0.5 seconds to ensure the car comes to a complete stop. If no stop sign was detected, then it will check if a red light was detected. If it was, and the bounding box was large enough, then it publishes the stop command. We decided to only look at the traffic light detector if there is no stop sign to ensure that the traffic light detector wouldn't get confused by the red from the stop sign.

# 3 Experimental Evaluation

***Marcos***

For our experimental evaluation, there were many ways to measure the performance of our various systems. However, this final assignment boils down to how well our systems performed on Race Day competitions. This makes evaluation simple, we can see how well our robot performed given the objective function

for each task in addition to traditional benchmarks.

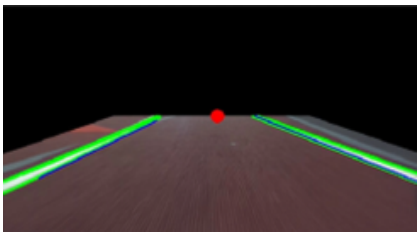
## 3.1 Mario Circuit

### 3.1.1 Lane Detector

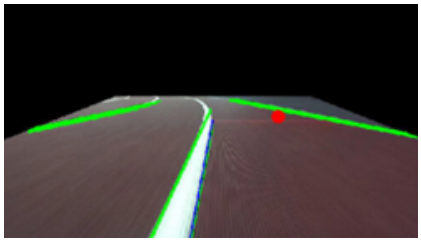
***Amy Shi***

We evaluated the accuracy of the lane detection algorithm by slowly driving the car around and analyzing whether its final calculated x-point is what we expected.

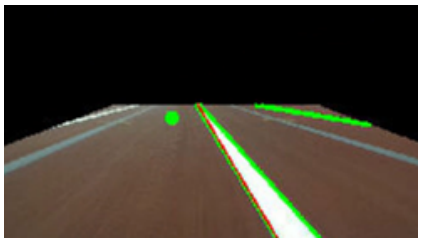
Some of the cases we considered and tested:



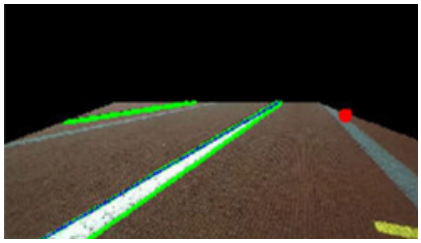
(a) Identify mid-point between two lines.



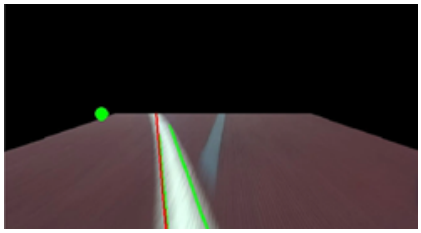
(b) Identify left line.



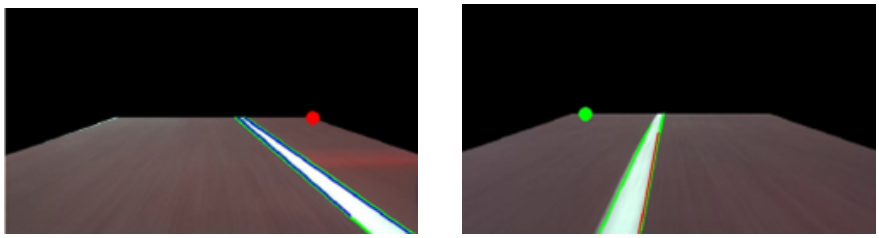
(c) Identify right line.



(d) Identify left lines when the robot is on the edge of the lane.



(e) Identify right lines when the robot is on the edge of the lane.



(a) Identify the correct left line when the robot drives into a different lane.

(b) Identify the correct right line when the robot drives into a different lane.

Figure 7: The images above show the algorithm correctly identifying the lines as either the left or right side of the lane. The x-point is also correctly identified for each of the scenarios.

The results from applying the program on the various cases matched our expected behavior of the algorithm. Thus, we deemed it as satisfactory for usage during the challenge.

#### 3.1.2 Driving Controller

***Marcos Espitia-Alvarez***

We evaluated the Lane Controller based on the rules of the Mario Circuit component of the Final Challenge: the speed at which it can complete an entire run around the track and the number of infractions from the run. When fully integrated with the Lane Detector, we completed a full run around the track at 58 seconds with zero infractions.

### 3.2 Luigi's Mansion

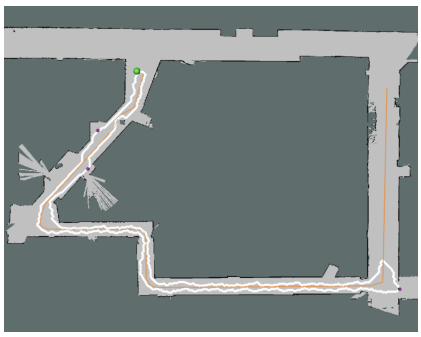
#### 3.2.1 Path Planning and Following

***Timber Carey***

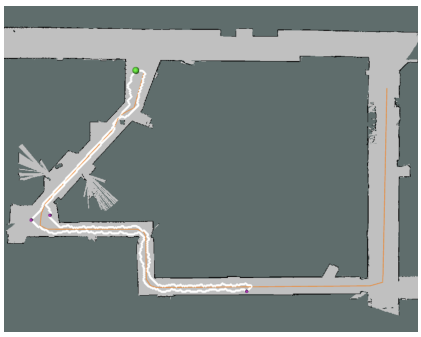
The path planner was evaluated both qualitatively and by the time it took to plan paths. Throughout the process of changing and adding to the path planning algorithm, we would visualize the path in rviz and print to the terminal how long it took to plan each leg. Some qualitative aspects of a planned trajectory included whether it was too close to a wall or to the lane divider, and whether it followed a logical path between the points (i.e. did it make a U-turn as soon as possible or continue a ways down the path unnecessarily first?).

The final path planning algorithm was evaluated on three different configurations of goal points (Figure 8). All of the trajectories met the qualitative goals of not hugging the walls or lane line and following a logical path. The time that it took to plan each leg of the paths in all three configurations is listed in Table 1. Because the robot can begin driving to the first goal immediately after the first leg is planned and continue planning the subsequent legs while driving,

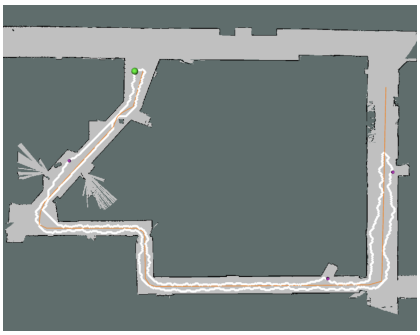
the time required to plan the first leg is most important. While planning the first leg of the path, the robot must wait at the start, and this planning time is counted towards our total run time.



(a) Goal configuration 1.



(b) Goal configuration 2.



(c) Goal configuration 3.

Figure 8: Visualizations of planned paths in three different goal configurations. Start point is a green dot and goal points are purple dots. The orange line is the lane divider and white lines are planned trajectories. All trajectories meet desired criteria of not hugging the walls or lane divider too closely and following a logical path.

As we can see from Table 1, the time required to plan the paths is not great enough to be an issue for the two minute time limit in this challenge. The robot only needs to wait at the start until the first leg is planned. The planning time for the first leg ranged from 1.8 seconds to 4.2 seconds, which is a very short amount of time to wait before driving. The planning time for all of the subsequent legs is also short enough that the robot would realistically not have reached the previous goal yet, so overall the planning time is efficient enough to only negligibly affect the total run time.

Table 1: Planning time for each leg of the path in three different goal configurations (Configurations shown in Fig. 8). Planning times are short enough that the total run time will only be affected by at most a few seconds.

| **Leg of Path** | **Goal Config. 1** | **Goal Config. 2** | **Goal Config. 3** |
|---|---|---|---|
| 1: Start → Goal 1 | 1.84 s | 4.16 s | 2.51 s |
| 2: Goal 1 → Goal 2 | 10.93 s | 7.22 s | 12.57 s |
| 3: Goal 2 → Goal 3 | 12.75 s | 8.09 s | 6.39 s |
| 4: Goal 3 → Goal Start | 4.23 s | 5.34 s | 14.69 s |

#### 3.2.2 Stop Signs, Traffic Lights, and Pedestrian Crossings

***Christian Teshome***

We first evaluated each component in isolation. The stop sign detector was tested to ensure that the amount of false positives and false negatives are as low as possible, so that our car wouldn't stop unnecessarily and wouldn't drive past stop signs. Additional testing was done to pick a good threshold for the bounding box size. First, we logged the area of the bounding box generated to get a sense of scale. While we didn't get to testing this on the actual robot, we next planned to tune the stop-time of 0.5 seconds and the bounding box size threshold by doing multiple runthroughs of the car with a stop sign, and seeing what values made the car come to a complete stop at the right distance while not stopping for unnecessarily long periods of time.

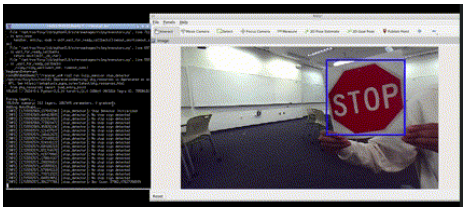



Figure 9: The stop sign detector working reliably while its moving relative to the car, drawing a blue bounding box around it.

Next, we tested the traffic light detector. The main things we adjusted were the lower and upper bounds for the HSV color values. Our initial implementation would have a lot of false positives, due to objects such as the door in the classroom, random orange/brown things on the floor of the hallway, the red board on the wall in the hallway, etc. We also noticed that when the red light is on, the color of it in the image would be partially white due to the brightness of the light. We tightened our color bounds to account for this, and made it so that it prefers objects with a smaller bounding box and therefore wouldn't get confused by the red board.

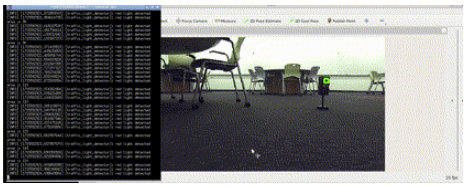


Figure 10: The traffic light detector working reliably while the car is moving, drawing a green bounding box around it.

As for testing how our car handles pedestrian crossings, this was mainly done by tuning the wall detector. We noticed that the wall detector detected too many false positives, and would unnecessarily back up during normal operation. We tried strictening the criteria for what counts as an obstacle, and also tried reducing the back up period so that it spends less time overall backing up. We weren't able to get this to fully work, so in our final run we had the wall detector turned off, however, if we had more time we would have adjusted the wall detector's parameters to ensure it could stop for pedestrians but also not stop too often.

# 4 Conclusion

***Subin Kim***

The final challenge for this course has been a comprehensive and enlightening experience that not only tested our technical abilities but also deepened our understanding of ROS. We tackled the Mario Circuit and Luigi's Mansion challenges, applying them into real-life inspired scenarios.

Our experience during the Mario Circuit emphasized the critical role of precision in control systems as well as lane detection. This part required a robust combination of algorithms to ensure that the vehicle stayed on track while maintaining high speeds. Conversely, Luigi's Mansion tested our capabilities in urban scenarios, where path planning, localization, and traffic laws dynamically came together.

The iterative process of troubleshooting and testing provided valuable insights that helped refine our knowledge and strategies. The real-time feedback from how the race car performed with given conditions provided effective evaluations for immediate adjustments, which ensured continuous improvement in our race car's performance. As we divided the tasks among different components of the challenge, we recognized the invaluable lessons we learned not just in technical aspects but also in managing project complexities and team dynamics. Debugging process allowed our team to dive deep into the intricacies of robotics, and we demonstrated our ability to integrate and apply complex robotics concepts in a real-world context, showcasing our preparedness to tackle future challenges in the field of robotics. This has been an invaluable experience,

providing us with both the skills and confidence to pursue further advancements in the field of robotics, as well as in other academic and professional endeavors.

# 5 Lessons Learned

## 5.1 Timber's Lessons Learned

I feel like I learned so much in this lab, especially because I think I contributed more to this lab than I have on any other labs in terms of just planning and writing code. The Luigi's Mansion challenge consisted of many parts that were being worked on concurrently by different people, which helped me learn that it is very important to keep code organized and modular. That way we could test each of the parts individually without other parts of the code getting in the way. I was able to really learn about the benefits of using the tools provided by ROS to organize different nodes with different functions, instead of just trying to put functions for different aspects of the challenge into one file. After this lab, I feel like I really mastered the organization of ROS nodes and topics with connecting publishers and subscribers. I looked back at some code that I had written in earlier labs this semester, and it was incredible to realize how much I've learned about using ROS since then.

I also learned that it is really important to make time for tuning and debugging, because code rarely works perfectly the first time and debugging can easily take way more time than writing the code in the first place. In the same vein, I learned about the importance of saving working versions of the code while tuning and debugging, even if it's not working perfectly. There were times in this lab where we would have some part working okay, but not perfectly, and then after further tuning it would not perform as well, but it was difficult to get it back to the previously better-working version.

Throughout the semester, but especially through the challenges of this final lab, I learned about the importance of having a good relationship with my team, and supporting each other both technically and personally. Our team ran into a lot of issues with our final implementation and worked through the night several times, but we always made an effort to all be there and support each other, and to try and never leave one or two people working on the lab alone. I think this team support really helped get through the lab's many difficulties.

## 5.2 Marcos's Lessons Learned

I had not worked too much on the CV portions of previous lab assignments. Therefore, working on the lane detection of the Mario Circuit was an incredibly fun and engaging learning experience. Now I feel confident using openCV to solve simple robotics problems requiring vision. Furthermore, I learned about some clustering methods and their use cases.

I also learned about the importance of more documented and stringent version control. It is easy to forget to save to Git when making small changes, such as tuning gains for a controller. Sometimes a set of gains would work at a certain speed, but the version would not be saved to Git as we continued iterating on a better controller. However, because of time constraints, our older versions were better because the current iteration would be buggy. Nonetheless, I had a great time working on the lab and I am thankful for being Amy's partner on Mario Circuit as she kept me in check whenever I got too excited and worked diligently to deliver a working final version. I am also grateful for being surrounded by a supportive team, we all worked hard and had each other's backs through the stresses of RSS.

## 5.3 Subin's Lessons Learned

This final challenge proved to be as demanding as it was enlightening, which required diligence and patience from all team members. Developing our approach to the challenge was one aspect; debugging and correcting the issues was another. The process was somewhat arduous, but the experience was ultimately rewarding, mainly because we tackled the challenge collectively as a team.

I am particularly grateful for my teammates who supported one another throughout the process, helping each other overcome obstacles and keeping ourselves accountable. Through numerous trials and errors and sleepless nights, our group not only gained a deeper understanding of ROS, but also some valuable lessons about collaborative effort for a common goal.

## 5.4 Amy's Lessons Learned

Through this final challenge, I truly learned the importance of troubleshooting in an organized manner. Due to the nature of the challenges, there was often a lot of code to keep up with. Any slight change with tuning the parameters of the controller or the algorithms can produce dramatically different results. It became very overwhelming very quickly, and I had to learn how to keep track of all my changes as well as thinking over carefully which parts can be going wrong before making any edits.

After experimenting with so many algorithms, I've learned a lot about the different types of driving controllers and ways to detect lanes. I learned a lot more about what to watch out for when tuning the racecar to produce a steady drive, and I've learned that there's a lot of different ways to complete the task. There may not always be a best solution, so it's important to test your algorithm and then stick with it instead of trying too many methods.

I've also learned the importance of communicating little things between teammates. Since the final challenge was split into two areas, it was hard for us

to help out with parts that we weren't as involved in. There were also a lot of Git conflicts, so it was a good learning experience for me.

### 5.5 Christian's Lessons Learned

This final challenge was a great opportunity to bring together all of the concepts I learned in RSS. It was also a good reminder, that if things should work "in theory", it's still very possible that it will not work in practice, and it's necessary to allocate time and effort to ensuring that it will actually work. Specifically, even though our localization, path planning, and path following were all demonstrated to work well in the path planning lab, there was a very large amount of testing / debugging required just to get it to work to the same level as before in the final challenge. I also learned the importance of planning tasks such that the most critical features are fully functional first.

In terms of teamwork, this challenge was interesting in that at the start we were partitioned into a group of 2 for mario circuit, and a group of 3 for luigi mansion. I learned the importance of over-communicating rather than under-communicating to ensure we are operating more in unison. I also learned that even if we are working on separate tasks, its still beneficial to ask each other questions to get other ideas on how to approach problems / help each other debug.

## 6 Acknowledgements

Thank you so much to all of the wonderful RSS staff this semester! It's been a great class! ♡

Formatting by Timber Carey. Editing by Everyone.