Final Challenge Report: Mario Circuit & Luigi's Mansion

Team 18

Eric Delgado, Arianna Ilvonen, Jesus Diaz, Charles Ge, Dora Hu

---

1: Introduction [Jesus]

In this report, we present an overview of our approach to the Mario Circuit and Luigi's Mansion challenges. The objective of Mario Circuit was to race around the indoor track at the Johnson Athletic Center as fast as possible without veering off of the track. Meanwhile, in Luigi's Mansion, we were tasked with navigating through the Stata Center basement to collect shells, taking care to avoid traffic infractions when encountering stop signs, traffic lights, and pedestrians along the way. Together, these challenges required us to draw on our experience from previous labs with controllers, computer vision, path planning, and localization to build a robust and successful robotic system. We begin by describing the setup and our approach to both challenges in sections 2 and 3, followed by an evaluation of our results in section 4, and conclude with our key takeaways from this challenge in section 5.

2: Technical Approach: Mario Circuit [Charles, Dora, Eric, Arianna]

The objective of the Mario Circuit was to run one lap around the track at a high velocity. We approached this by taking images from our ZED camera, then using computer vision to determine a point to head towards. After finding that point in the image, we used homography transformations to calculate that point's position relative to the car. We then used that point to

calculate a drive command. By continually using the camera feed to publish commands for the car, we aimed to make our car keep going forward in its lane on the track.

2.1: Vision [Eric]

To implement the Mario Circuit around Johnson Track, we required a method to determine a goal position using the ZED camera and to track this position along the circuit. This was achieved through a sequence of OpenCV operations on the incoming RGB images, as depicted in Figure 1a. Initially, the RGB image was transformed into a binary grayscale image to isolate the white components. We then applied a MorphOPEN operation to eliminate noise and small features, followed by a dilation operation to enhance the white features, ensuring accurate readings. Subsequently, a specified bounding box was applied to the image, and by averaging the columns of white values within the image, a goal pixel was determined at the designated lookahead height and average column, which was then published.

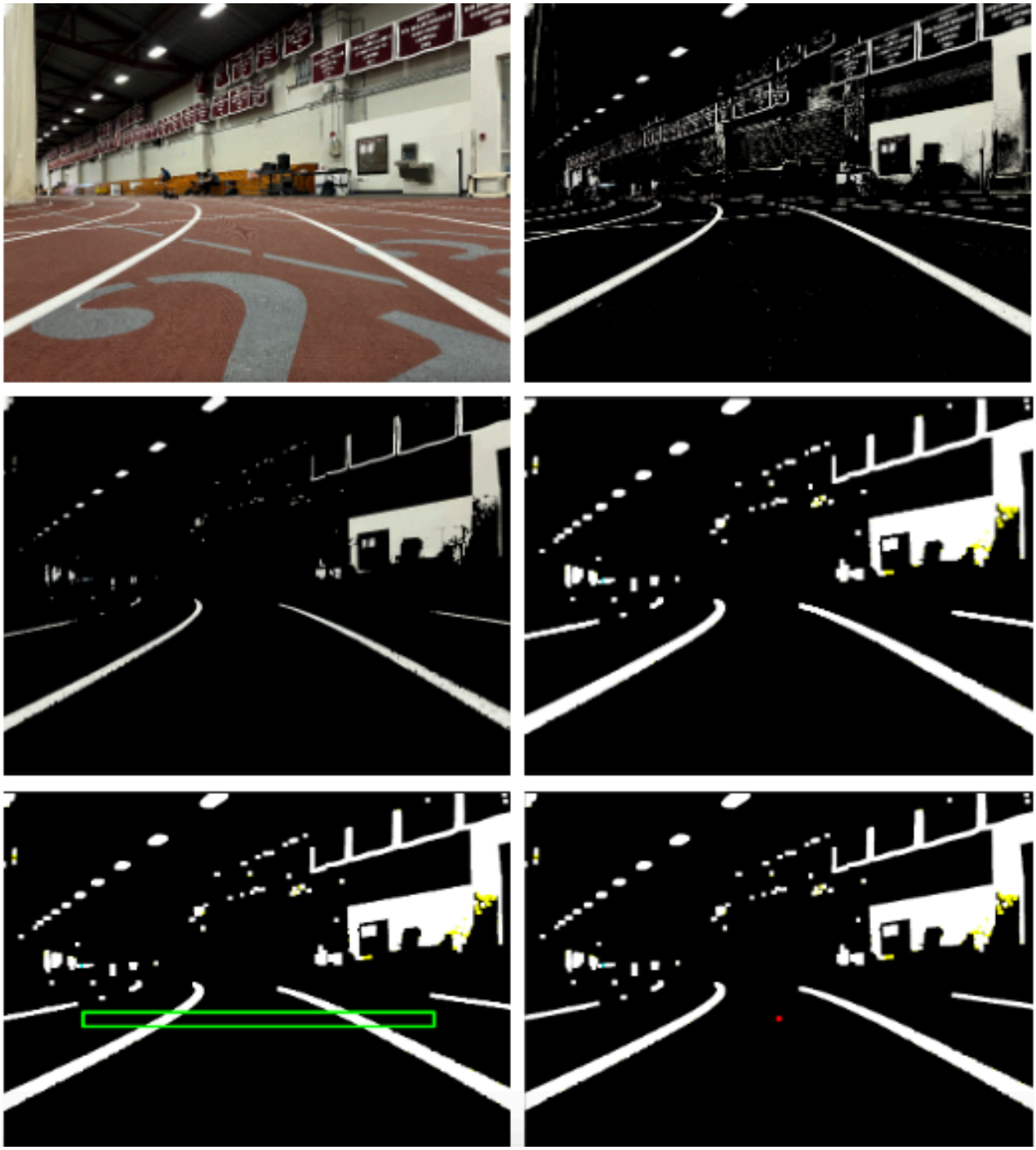

Figure 1: Visualization of extraction of goal point. Operations proceed left to right, top to bottom. Green box indicates bounding box. Red dot indicates the goal pixel.

2.2: Goal Following [Charles]

After determining a goal pixel in the image frame of the robot, our approach applied a homography transformation to derive the goal point in the robot's frame. This algorithm converted the (u, v) coordinates of the image goal point, in pixels, to (x, y) coordinates in meters by multiplying by the homography transformation matrix H and adjusting for a scaling factor s, given by Equation (1) below:

$$s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{1}$$

The values in H were manually determined and calibrated in previous labs for the robot's ZED camera setup.

From the goal point location (defined as positive x forwards and positive y left), we computed the error as the angle θ between the goal position vector and the current robot heading (θ defined as positive counterclockwise), as in Equation (2) below:

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) \tag{2}$$

We then computed the control action using a controller with θ as the error input. Before computing the control, we utilized several manual corrective measures to help keep the car within its lane and make our solution more robust to variation within the lane. If we determined that the magnitude of the horizontal error $|y| > 0.5$, we ignored the spike in error and instead kept the car driving on a straight line command. Also, if our image detection found more than two white lines, thus rendering the averaged goal pixel inaccurate, the algorithm returned the previous drive command. Finally, if the image detection algorithm only found one white line (and incorrectly set a goal pixel on the line), we applied corrective steering action by inverting

the goal position along the robot's axis of travel to steer the robot back towards the center of the lane.

2.3: Controller Action [Charles]

Finally, to keep the car within the lane, we used a modified proportional-derivative (PD) controller on the error input θ. Alongside tuning K_p and K_d values for the controller, we also added a constant steering angle offset δ_0 when running the robot in the counterclockwise direction along the track to correct for the robot's natural rightwards drift. Lastly, our controller clips the steering angle control action to the range $\left(-\frac{\pi}{100}, \frac{\pi}{100}\right)$. This clipping helps add robustness to the Mario circuit algorithm by reducing the impact a single callback can have on the trajectory of the car and smoothing the output steering commands. Figure 2 below shows the block diagram of the full controller used to steer the car.

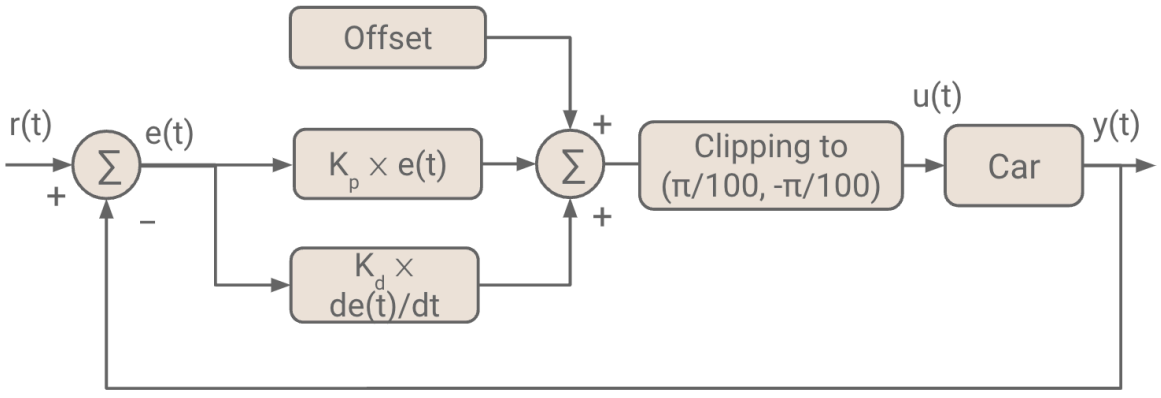

Figure 2: Block diagram of the controller used to steer the robot. The error signal $e(t)$ is the computed θ from the goal following algorithm in Section 2.2.

3: Technical Approach: Luigi's Mansion [Dora]

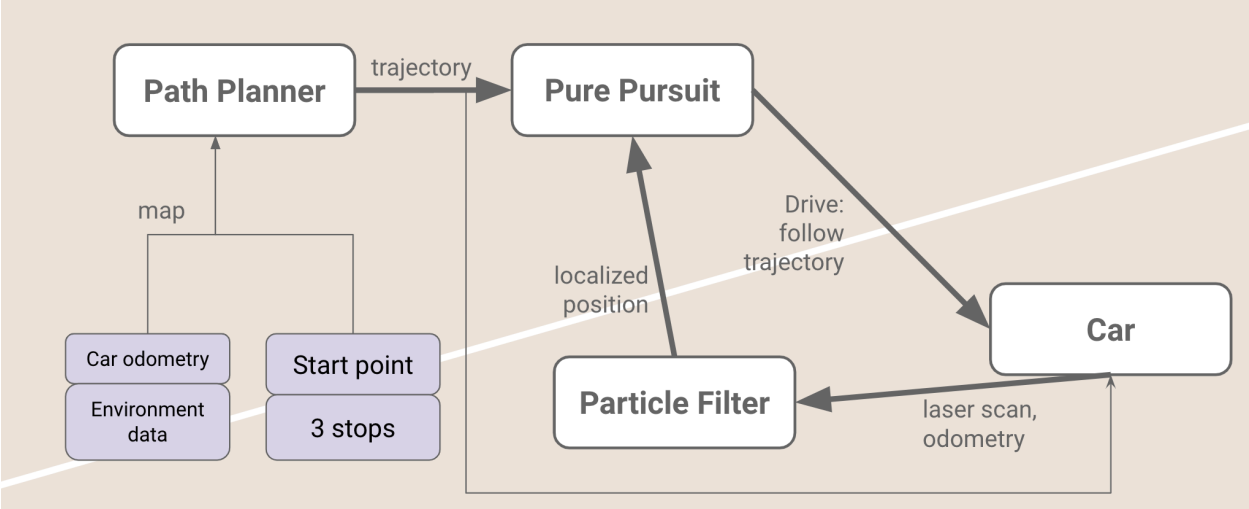

Figure 3: Block diagram of modules used to implement Luigi's Mansion challenge..

Luigi's Mansion had several challenges involved. The primary challenge was to be able to randomly select three points to travel to and briefly stop at, in order. Furthermore, we were challenged to obey traffic laws while traversing the path determined by selecting the three points, as well as return to the starting point of the robot car. Traffic laws included staying within the lanes (of which there were two going in opposite directions), stopping at red traffic lights, stopping at stop signs, and yielding to pedestrians. Our system was centered around a path planning system that would calculate a trajectory for this challenge, given our map of Stata basement, and use pure pursuit to follow it while running other modules in parallel to follow traffic rules.

3.1: Path Planning [Jesus]

One of the most significant changes from the path planning lab to the Luigi's Mansion final challenge was the introduction of lanes, which required our car to adhere to the right-hand

traffic standard to reach shell pick-up points and only cross lanes to change driving directions with a 180 degree U-turn. To achieve this, we built on our approach from the path planning lab, where we had devised a method of discretizing areas in the given map of the Stata basement into a traversable graph.

Lanes required us to modify the way we chose to connect nodes in the graph. Previously, we only had to ensure that we were connecting unobstructed areas together. Now, these unobstructed areas also had to be connected in such a way that our robot would find paths that adhered to the unidirectional nature of lanes, while also allowing for the possibility of a complete 180-degree U-turn. Our solution to this was simple: define a set of rules for connecting nodes according to their distance from the lane. For nodes close to the lane, we connect them to nodes on the same lane at a 45-degree angle, in the correct driving direction, and for nodes in different lanes, we connect them in such a way that the edges between nodes are directly perpendicular to the lane. Nodes farther away from the lane would be connected to nodes within 45-degrees of the line directly parallel to the lane, in the correct driving direction.

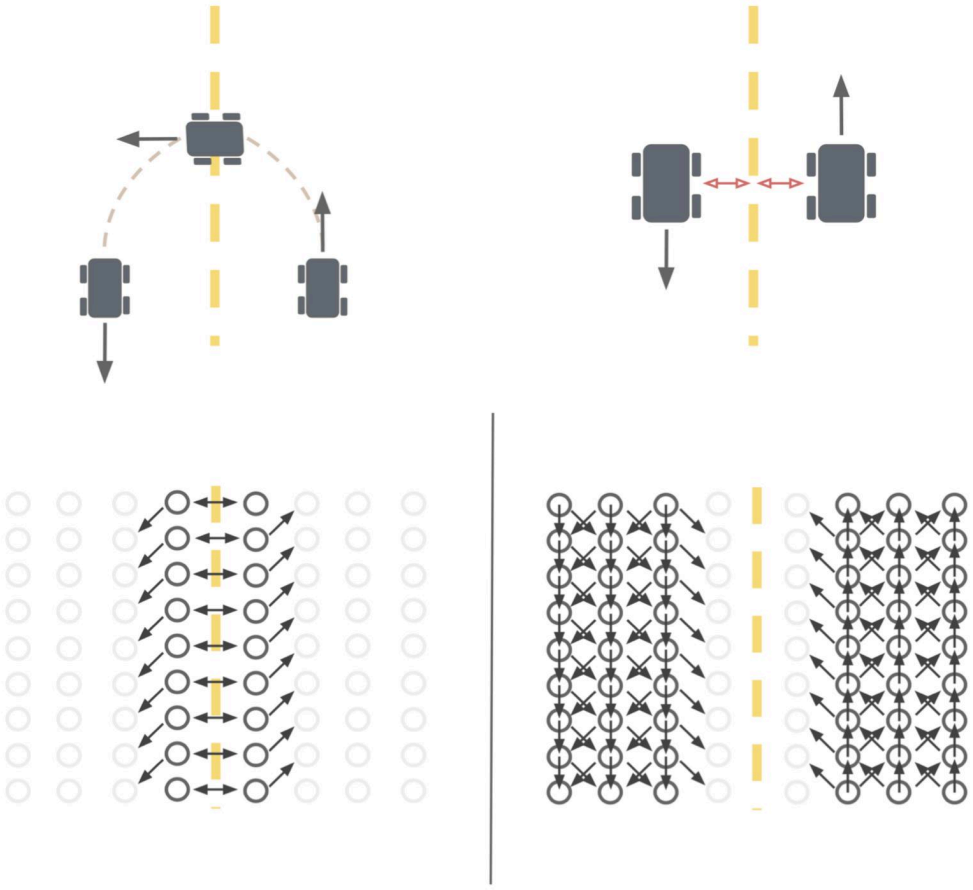

Figure 4: Desired driving behavior and visualization of the way that nodes are connected in the graph according to short and long distances from the lane.

Connecting the nodes in this way restricted our path-planning algorithms in a few ways. Firstly, the connections from nodes that were close to the lane gave us the ability to create paths with U-turns. Once a path crossed into another lane, it would be “carried away” so to speak by the nodes in that section of the graph in the correct driving direction. Nodes close to the lane were also only connected to adjacent nodes (that is, nodes on the same side of the lane) at 45-degree angles, and as such required more traversals to travel the same amount of distance when compared nodes farther from the lane, which could go directly in the correct driving direction. Because of this, the shortest paths calculated rarely went near the lane except to conduct U-turns, which allowed us to keep a safe distance from the lane at all times.

3.2: Acknowledging Stopping Points [Dora]

To earn points at all for Luigi's Mansion, we had to have our car stop at all three user-selected stops for at least five seconds. We did this by storing the points in order and after all three were chosen and calculate between each pair of consecutive stops. The way we represent trajectories is based on a list of points for the robot car to travel to, so we concatenated the smaller trajectories into a single larger trajectory that visited the stops in our desired order.

However, we wanted the car to pause when it reached a stop, but continue in its trajectory after the allotted time had passed. This would require us to suspend the drive commands published by our pure pursuit algorithm—when necessary—to enable these stops. We did this by publishing the pure pursuit's drive commands to an intermediary instead of directly to the car. When the car's location reached a stop, the intermediary would temporarily publish zero-velocity drive commands to the car to make it pause; otherwise it let the pure pursuit's drive commands go to the car

3.3: Stop Light Detection [Arianna]

Although we were not able to fully integrate stop light detection into our Luigi Mansion runs, we did independently create a stop light detector capable of outputting when the car should stop. To handle the stop lights, we decided to detect if there was a red light when approaching the designated intersections, and if there was, set our car to stop until there was no longer any. In order to do this, we created a node that takes in data from the camera, then runs it through a series of OpenCV transforms and thresholding to determine if there is a red light present.

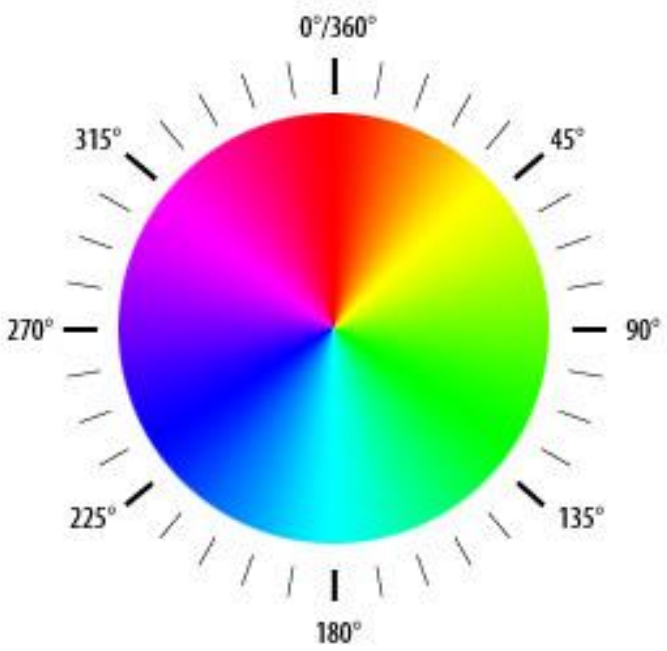

Figure 5: Visualization of the HSV hue spectrum. This image displays how thresholding for the color red can be difficult, as it is both near 0 degrees as well as 360 degrees.

Figure 6: Image showing an original test image vs a test image that has been processed down to only the relevant sections by setting the top, bottom, and right side to black. This reduces accidental detection of red environmental objects. Adapted from [1].

We started by converting the image from the ROS Image message type to the OpenCV image type by using ROS's cv_bridge. In order to threshold for a specific color, we converted to HSV, but since red is at the very end and the very beginning of the HSV hue spectrum, it was difficult to threshold for, as illustrated in Fig. 2. To combat this, we inverted the image, so

instead of thresholding for red, we were thresholding for cyan, a color that makes up the very middle of the hue spectrum. After converting to the HSV color space, we set all the areas of the image where the stop light should not be to black to prevent false positives from other red objects. This clipping is illustrated in Fig. 3.

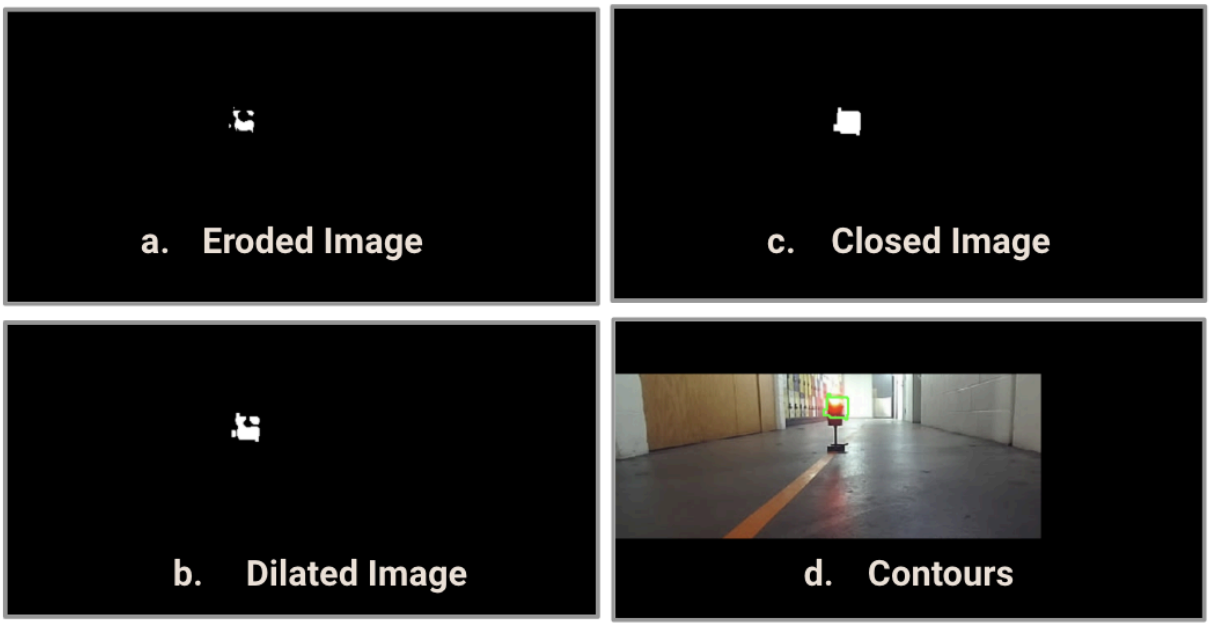

Figure 7: Set of four images displaying the four OpenCV mask transformations performed on the thresholded image mask with erosion, dilation, and closing before performing contour selection. This set of transformations remove noise and then close the points in one solid object that can be detected.

Using the threshold values, we then create a binary mask of the image, and subsequently erode, dilate, and morphologically close it using OpenCV functions. These three operations are displayed in Fig. 4. Eroding the image (a) reduces unnecessary noise, while dilating it (b) helps make the area of interest larger. Once these two operations are completed, closing the image (c) allows us to unify several disjointed patches of color into one solid shape. These solid blocks of color are excellent to use for finding contours (d), which we can use to find the center of the shape.

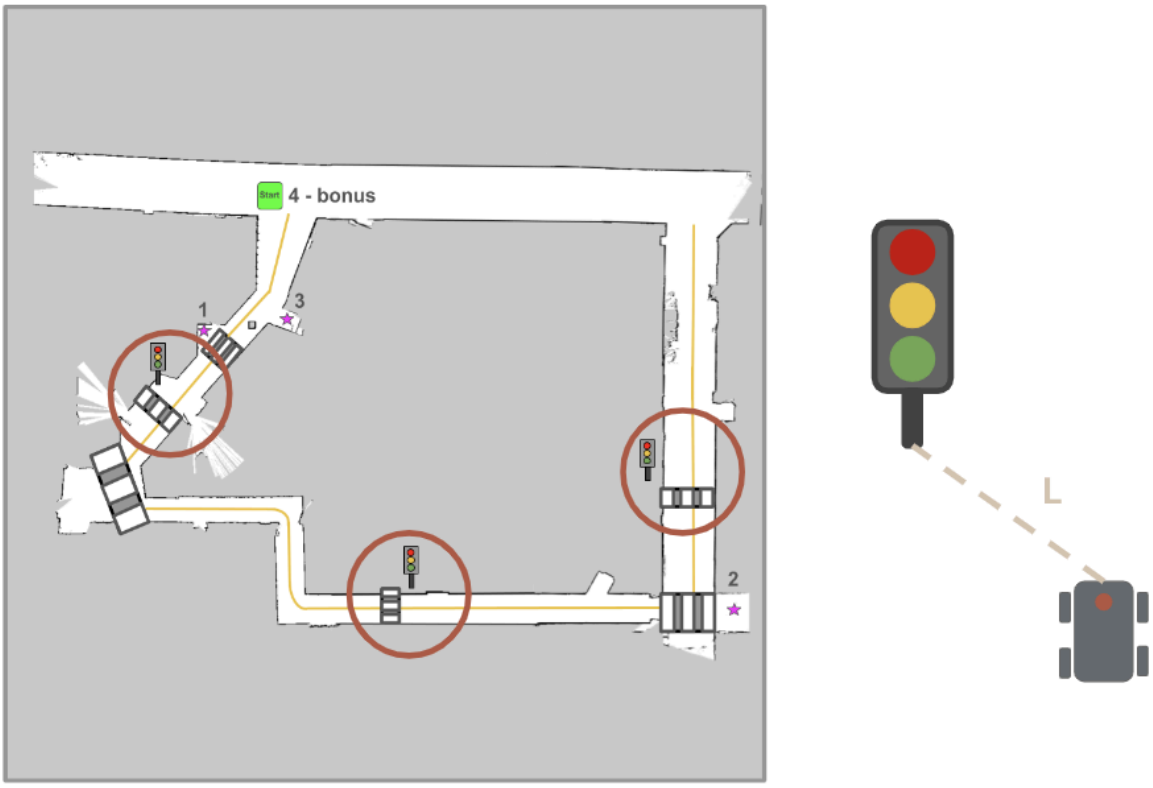

Figure 8: Map displaying the regions in which the car is “within range” of a stoplight (circled in orange). These regions are of radius L, which is the maximum distance away from the stoplight that the car should stop. While the car is outside of these regions it will not stop even if a red stoplight is detected. This prevents extraneous stops.

To prevent any false positives while traversing the entire map, we decided to also check the location of the robot when a red light is detected. By setting a maximum range from the given stoplight locations, we can determine if the car is within stopping distance of a stoplight (within the orange circles or less than length L in Fig. 5), and only publish a stop command if the car is within stopping distance and detects a red light, as illustrated in Fig. 5.

3.4: Stop Sign Detection [Charles]

We also were not able to integrate the stop sign detection with the rest of the Luigi mansion code, but we approached the detection as follows. In order to reduce the impact of car

performance from the computational load of running the stop sign detector too frequently, the stop sign detection callback was called only every three seconds to process the live ZED camera image. An instance of the StopSignDetector object provided in the skeleton code was then called to predict whether a stop sign existed in the current image, and if so, where (via bounding box coordinates) using a YOLO machine learning algorithm. Then, the center of the bounding box was computed in image pixel coordinates and converted to an (x, y) location in the robot frame after applying a homography transformation (explained previously in Section 2.2. We then computed the distance to the intersection d, as well as the time to arrival at the intersection t given the current velocity v, as outlined in Equations (3) and (4):

$$d = \sqrt{x^2 + y^2} \tag{3}$$

$$t = \frac{d}{v} \tag{4}$$

This time-of-arrival to the intersection is an underestimate due to the straight-line-distance to the stop sign used. The algorithm continues at the polling frequency of once every three seconds until the time-of-arrival computed is within seven seconds, during which we increase the callback frequency to once every second until the computed d is less than one meter from the stop sign, in which we send a stop command to the robot.

4: Experimental Evaluation

4.1: Controller Evaluation Mario Circuit [Arianna]

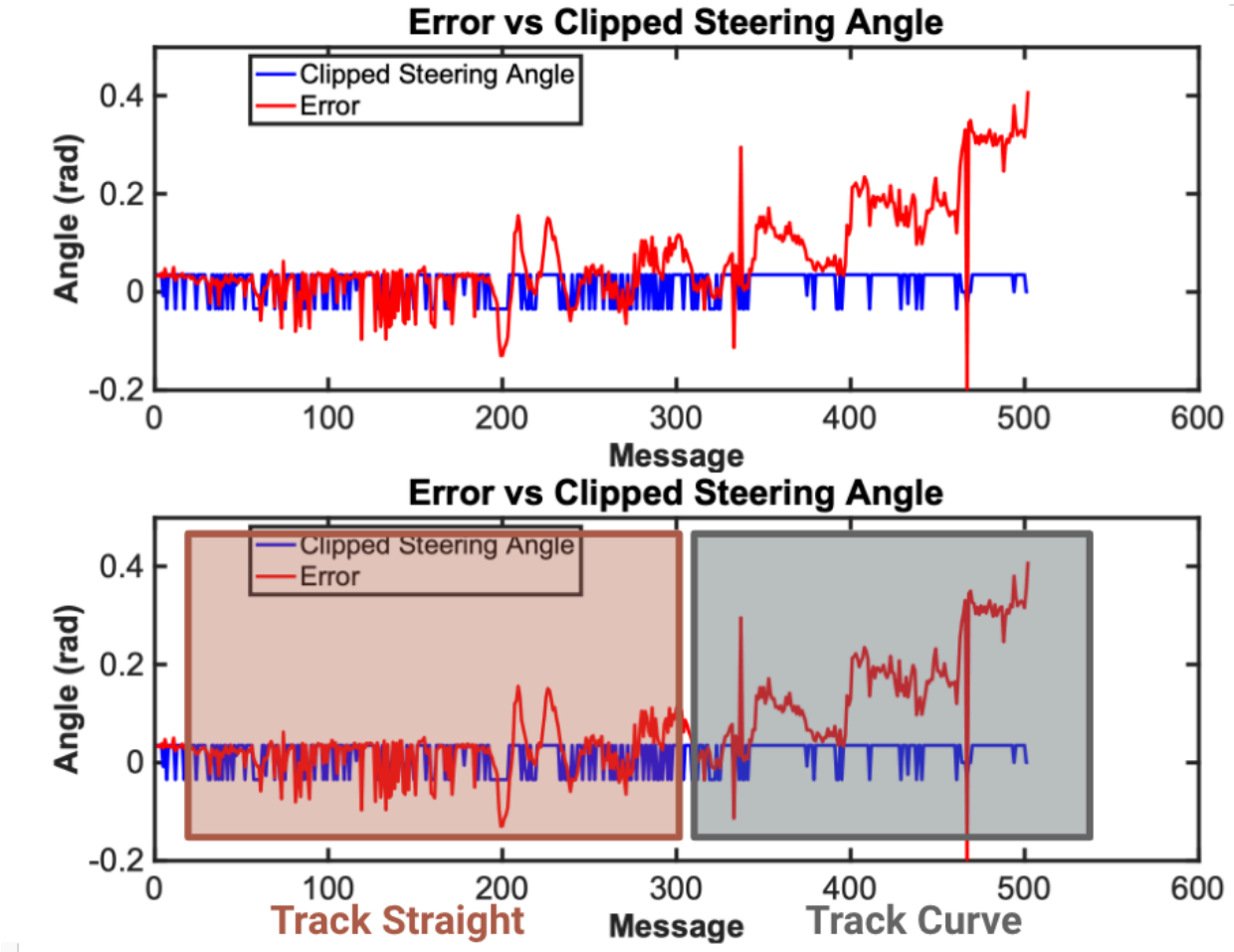

Figure 9: Graph visualization of the error in each message (in red) overlaid by the steering angle published to the car (in blue). In the first half of the graph, highlighted in orange, the car moves along a straight track, while in the second half, highlighted in gray, the car moves along the curved part of the track. From this, it is apparent how the steering angle varies between positive and negative on the straights, but leans heavily positive during the curve.

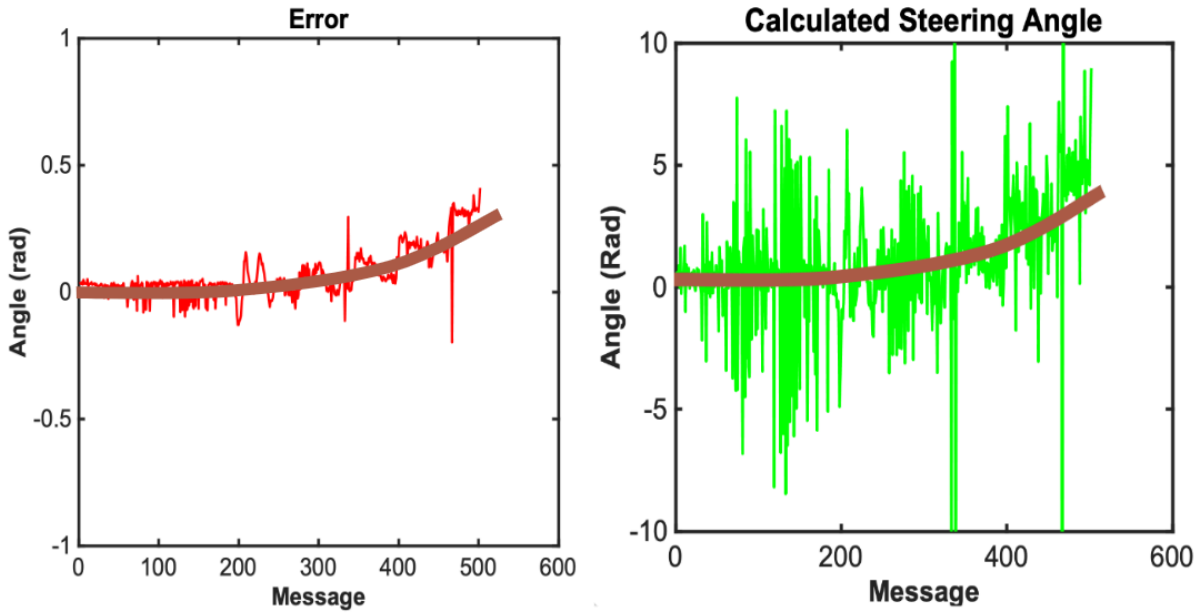

Figure 10: Graphs showing the error in red on the left vs the calculated steering angle (non clipped) in green on the right. This set of graphs displays how the trend in the angles, plotted by the orange line, is the same even if the calculated steering angle is an order of magnitude larger than the error.

When running the robot along the Mario Circuit track, we collected data on the response of the controller to the error. This data allows us to evaluate how our controller responds as the robot moves around the track. In Fig. 6, two different controller states are visualized. In the orange highlighted section of the graph of Fig. 6, the car is moving along a straight section of the track. During this period, the car is steering right and left for approximately equal amounts of time. In the subsequent gray highlighted section, the car begins to move along a curved portion of the track. During this period, we see the measured angle increase significantly, and the steering angle is primarily positive. From Fig. 7, we see again that the calculated steering angle follows the same trends as the error values. The calculated steering angle is a magnitude larger than the error, suggesting that the clipping of the steering angle was necessary. This data suggests that the controller is responding appropriately to both the straight and the curved track.

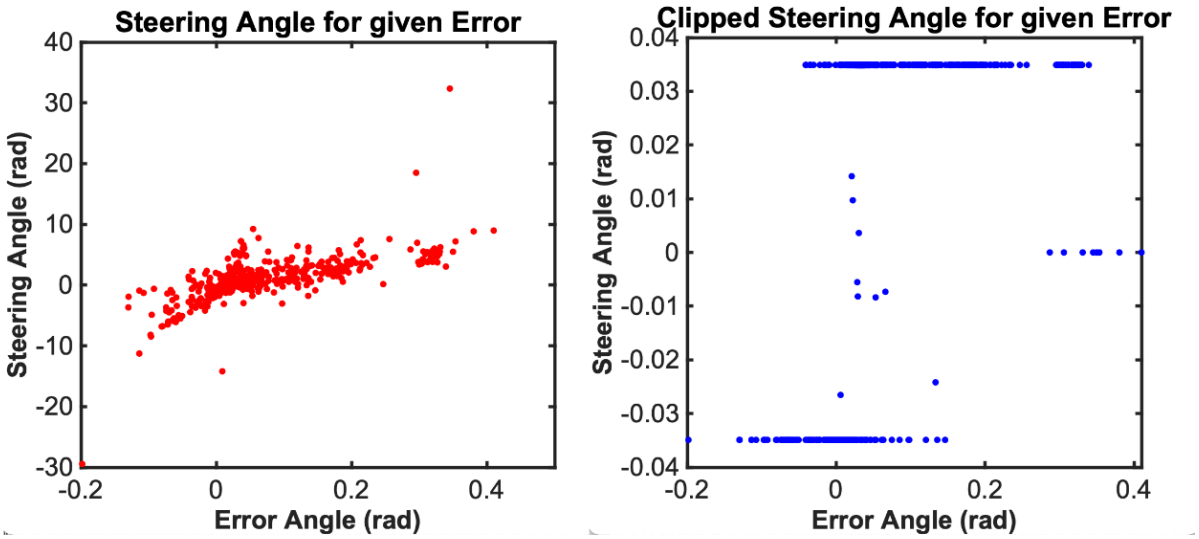

Figure 11: Visualization of Calculated Steering Angle and Clipped Steering Angle values for given error values at 3 m/s. These plots show how these values can vary, despite having the same input, depending on the last measured value, especially near 0 error angle. This also displays that our output command was almost always either positive or negative 0.04 radians, which is the maximum for the clipped values.

From Fig. 8, it is apparent that the derivative value in our controller had a significant impact on the steering values. Around zero error angle especially, you can see that for the same input error angle, the output steering angle can be significantly different. This was critical for high speed control around the Mario Circuit Track, as the derivative component of the controller allowed the car to respond faster to changes while also reducing overshoot in the controller response. While we were not able to tune the car to work for both the straight and curved sections at the highest speed (4 m/s), we were able to tune it to run both correctly at 3 m/s. The use of a high level of derivative control made this possible, and the tuning of our controller reflects that.

4.2: Accuracy Evaluation of Luigi's Mansion [Dora]

When testing Luigi's mansion, we primarily evaluated our car's performance based on the extent of its success in completing the challenge. The metrics we used for our system were the number of stops we could reach with and without manual assistance, the number of times the car crossed into the other lane, and its accuracy in identifying traffic lights, stop signs, and pedestrians.

The path planning algorithm was accurate enough in simulation: it successfully identified stops and followed the trajectory from the start to the first stop, to the second and third, and back to the first. However, the same did not apply to the real car when we first tried the path planner on it; the car experienced problems such as not stopping at stop points and deviating from the planned trajectory. We debugged the real car's problems by examining intermediaries such as the trajectory points, the lookahead point found by the pure pursuit module, and the turn angles in drive commands supplied by the pure pursuit module. For instance, we found that looking too far ahead in the trajectory led to misleading lookahead points found in trajectories that go back along the paths they came from. Thus, the car would be led astray by points close by geographically but far away on the trajectory that had vastly different directions intended on the trajectory. We resolved this issue by only looking "ahead" a certain amount for already-built trajectories to avoid misleading points.

We did not have time to collect data on the stop sign detection algorithm nor the safety controller due to not having time to integrate that portion of the lab. However, we were able to test the stop light and safety controllers by themselves. The safety controller was able to detect every time we stepped in front of it and successfully stopped the car's wheels from moving.

4.3: Stoplight Accuracy for Luigi Mansion [Arianna]

Figure 12: Image displays the distance in pixels from the actual center of the stoplight in each test image to the detected center of the stoplight. For each of the images the error was below 10 total pixels, suggesting that our stoplight detector was very accurate.

Although we did not have the chance to fully integrate stoplight detection, we evaluated our stoplight detector using a set of test images taken from the car's camera. For each of these images, we manually selected the center, then used the detector to determine the center of the stoplight (if there was one). In Fig. 9, for each image we took the difference in actual vs determined stoplight center to determine the accuracy of the stoplight detector. From this data, we found that the detector was accurate to within a few pixels on every image, and accurately determined when no red light was present in the image.

5: Conclusion and Lessons Learned [Dora]

Over the course of this lab we became much more resilient in terms of debugging our code and dealing with unexpected obstacles. We started our lab quite early but spent a lot of time focusing on perfecting our algorithms in simulation, as opposed to failing early by starting

integration early. As a result, while we did start integration relatively early for this challenge, we still fell short of what we hoped to achieve with respect to our success rates because we did not spend enough time on integration.

For the Mario Circuit, we spent an unexpectedly long amount of time fine-tuning our parameters for our controls. We did anticipate a longer amount of time that we'd previously used to integrate, but did not predict the amount of time needed to fine-tune. For Luigi's Mansion, we did not spend as much time integrating because we had to split our single car between both challenges. Thus, we began integration for Luigi's Mansion significantly later than that of Mario Circuit. As a result, we ran into typical bugs associated with integration, such as close calls with the walls, which required manual assistance from us to resolve during track runs.

Ultimately, we have grown greatly in terms of resilience and ingenuity in overcoming obstacles in engineering our cars. The final challenge of this class has prepared us to tackle challenging engineering problems based on entirely new topics and taught us how to efficiently delegate work and communicate to work efficiently.

6: Citations

[1] SPOT Imaging Solutions, a division of Diagnostic Instruments, Inc, "Color Space - SPOT Imaging," *SPOT Imaging Solutions*, Jun. 29, 2017. https://www.spotimaging.com/resources/white-papers/color-space/ (accessed May 14, 2024).