6.4200/16.405 SP23 - Final Challenge Report: Coding an Autonomous Vehicle System for Track Racing and City Navigation

Team #20

Aaron Zhu
Jessica Rutledge
Chuyue Tang
Nihal Simha
Shara Bhuiyan

6.4200/16.405 Robotics: Science and Systems (RSS)

May 16, 2023

1 Introduction (Aaron / edited by Jessica)

In the modern era, vehicles are used for wide-ranging applications. From achieving thrilling speeds on race tracks to navigating busy cities for daily tasks, vehicles have many purposes. When designing an autonomous vehicle system, it is valuable to create one that is robust to all such real-world applications.

However, several challenges arise with each unique driving scenario. When cars race at high speeds, for example, the motion must be robust to minimize oscillations and compute commands at a fast frequency. When driving in a city, extra complexities such as following the rules of the road and executing more precise turns must be considered. In all scenarios, however, the upmost priority is ensuring the safety of the vehicle. Especially as testing is done in more real-world scenarios, real-world implications become even more relevant.

For the final challenge, we synthesize all our knowledge obtained from previous labs, culminating in an autonomous vehicle system that can be programmed for two specific scenarios. In the Final Race module, the car is able to race at high speeds around an indoor track while remaining inside a specified track lane. In the City Driving module, the car is able to navigate through a city, appropriately stop at a stop sign, and successfully park in a predetermined region.

This paper demonstrates the following contributions:

- Using perception and control techniques, we discuss how both our Final Race and City Driving implementations allow the race car to safely execute a fast lap around the track and a path through the city, respectively.
- Through empirical evaluations, we demonstrate that our implementations for the Final Race and City Driving modules are robust and ready for the final competition.

We detail our technical implementations of both the Final Race and City Driving modules in Section 2 and evaluate the experimental results in Section 3. We briefly discuss our system's performance on the final Race Day in Section 4 before presenting our conclusions in Section 5 and sharing our personal lessons learned in Section 6. Videos and images associated with this lab are available here .

2 Technical Approach (edited by Aaron)

In this section, we detail the technical implementations for both the Final Race and the City Driving modules. Both are similar in that they involve a perception component to obtain a target point and a controls component to command the driving directions needed to follow this point. We begin by discussing the selected computer vision algorithms appropriate for each module, and how each algorithm precisely obtains the necessary information for the target point. Next, we describe the control algorithms relevant to both modules. Both utilize a pure pursuit controller to command the appropriate steering angles to car, but unique modifications are made to specialize the controller to each module. Lastly, we explain the additional controller logic specific to the City Driving module that allows for certain driving scenarios to be accounted for.

2.1 Perception (edited by Aaron)

To inform of its required movements, the race car system must be able to obtain the necessary features from the environment. For instance, the car needs to detect white track lines (Final Race) and it also must detect orange lines and stop signs (City Driving). To do so, different computer vision techniques are appropriately chosen to obtain the necessary information for each of the two modules. Resultingly, the perception component returns a target point for the car to use in its controls.

2.1.1 Final Race - Lane Detection (Shara)

A key objective of the Final Race is to ensure that the car stays within the lane lines. From the car's point-of-view, the track lines appear slanted, as shown in Figure 1. Thus, a built-in target point is essentially provided via the intersection of the two track lines. As this intersection point is always guaranteed to lie within the lane, this theoretically should ensure the car will stay inside the lane if correctly followed. However, the process of cleanly obtaining the correct lines via the car's camera presents many challenges; for instance, lighting changes and other distracting features existing on the track (as seen in 1) make it difficult to extract the necessary lines.

Color segmentation is too difficult and unreliable in terms of distinguishing the white track lines, as it is hard to exactly tune the HSV color ranges to explicitly differentiate between white and gray on the track. Taking advantage of the white-colored features,

this challenge is remediated by converting the image to gray-scale. This allows for easier tuning as only one parameter needs to be tuned (brightness) and filtering out the gray lines from the white becomes much easier.

(a) White track lane numbers can also be seen by the car, which must be accounted for when doing white line detection.

(b) Other horizontal white lines and other colored features can be present on the track, forcing the computer vision component to be robust to such extraneous features.

Figure 1: View from the car camera's perspective of the track. The white track lines appear slanted, which provides a natural solution to obtain a target point which is the extrapolated intersection of the white lane lines. However, there are other features on the track that must be accounted for when using the vision data. (Figure Credit: Aaron and Shara)

After white features are successfully extracted from the camera as a mask, more challenges arise as there are additional white-colored features such as lane numbers and starting lines that should be differentiated with the white lane lines. To account for this, the top-half and bottom-middle portions of the image are cropped away, as shown in Figure 2. This ensures only features located on the ground (track) are focused on, and features present in the middle of the lane (horizontal starting lines, lane numbers) are also removed. This step is essential as it reduces the chance that white-colored objects other than the lane boundaries are detected.

After the white-colored components of the image have been found, OpenCV's Canny edge detection is used on the masked image to return all possible edges, as detected by the algorithm. Hough Line Transforms are then used on the binary image produced by Canny edge detection to find straight lines in the image. These detected lines are parameterized via polar coordinates, with r and θ as the radial distance and polar angle from the origin to a point p on the Hough Line. This information does not preserve the size of a line, so the length must be scaled by a factor to ensure the intersection point can be obtained. Using r and θ, lines can be converted into slope-intercept form by solving for the slope m and y-intercept b:

$$m = -\cos(\theta)/\sin(\theta) \tag{3}$$

$$b = r/\sin(\theta) \tag{4}$$

After the slopes of the detected lines are computed, thresholding is done to 1. classify lines as potential right lanes or left lanes and 2. remove unwanted lines that may have been erroneously detected. For example, horizontal lines (slopes close to 0) are ignored.

Left lines have a slope between -1.01 and -0.5, and right lines have a slope between 0.26 and 1.01. However, it is possible that erroneous lines will still remain after the thresholding process. To address this issue, the average of the lines is taken for both the right and left sides by averaging the slopes and y-intercepts. With these two lines, the intersection point can be obtained by simply solving the linear system of equations for the two lines. Figure 2 illustrates how the intersection point is obtained.

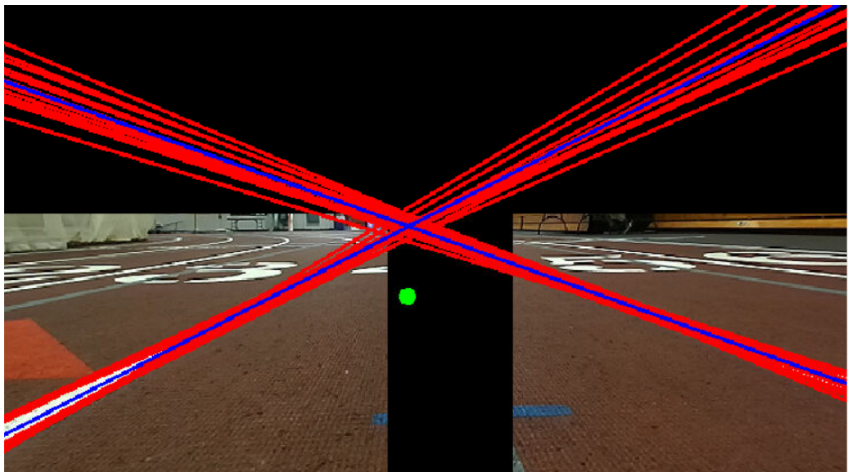


Figure 2: Screenshot from `rqt-image-view` illustrating the cropped image and the target point. The image is cropped to disregard the top half and bottom middle to ensure only relevant parts of the image are considered. This extra measure helps to ignore other distracting features in the environment such as gray lines and lane numbers. All detected lines are shown in red. To account for noise, an averaging of the lines is done for both the left and right sides to obtain the two blue lines. The target point (shown in green) uses the horizontal component of the intersection point, but is vertically shifted down to reflect the lookahead distance (a tunable parameter). (Figure Credit: Shara).

2.1.2 City Driving - Line Detection (Nihal and Chuyue)

The mock city (as seen in Figure 3) provides two possible avenues to design the City Driving module. The orange line (denoting the road) makes it possible for an implementation based on line following to be designed, where the car simply follows the line (as in Lab 4); meanwhile, the boxes (representing buildings) and a supplied map of the city suggest that localization could also be used, where the car calculates and executes the optimal path from a start to end position on the given map. Using localization however presents various challenges. As the path consists of sharp turns, the executed path must be very precise to avoid deviating from the road and possibly colliding with objects. This is hard to ensure as the actual position of the cardboard boxes are 2-4 inches off from the provided map, which could cause the car to erroneously localize itself in the wrong location. Furthermore, the actual boundary of the map does not match that of given map. The provided map is boundary-less, but the surrounding walls and hallways in the Stata basement suggest otherwise. This causes erroneous sensor probability distributions to be sampled (although, this could be fixed by overlaying the city map on the Stata basement map). These reasons suggest that localization is an unreliable method; thus, line following is a better option.

Drawing upon previous work with line following (Lab 4), the first step is to detect the orange line using a cropped image and color segmentation. The image is converted from BGR format into HSV using OpenCV's `cv2.cvtColor`. The upper half of the image is

Figure 3: Picture of the mock city in the Stata basement with a road (orange line), buildings (cardboard boxes), and a stop sign. Although the setup provides opportunities for either localization or line following to be used, localization is largely unreliable due to the imprecision of the box locations and the enclosed walls of the Stata basement. This suggests line following is perhaps a more appropriate design for the City Driving module. (Figure Credit: Nihal)

cropped out (focusing only on the bottom half of the image closer to the car). Then, a mask is used that filters the image according to the HSV values corresponding to the orange line, using OpenCV's `cv2.inRange(hsv_img, lowerb, upperb)` where `lowerb = (0, 120, 90), upperb = (20, 255, 255)`. The mask discards unwanted sections, allowing contours to be extracted from the binary image via `cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)`. Finally, a bounding box of the orange line is obtained using `cv2.boundingRect(largest_contour)`.

To avoid bounding boxes that erroneously include unwanted contours on the brown cardboard boxes in the city, additional thresholding is done to filter out bounding boxes whose width and height are less than a certain threshold value. Ultimately, a target point (in image pixels) is obtained using the top left and bottom right coordinates of the bounding box. A homography transform H tuned specific to the car (see Figure 4) is then used to project the image pixel points $\mathbf{p}$ =(u,v) onto the floor with coordinates $\mathbf{x}$ =(x,y), using the equation:

$$\mathbf{x} = H\mathbf{p} \tag{5}$$

Although the target point could previously be obtained by simply averaging the bounding box coordinates in Lab 4, this method is not robust enough for the City Driving module. As the car executes very sharp turns, color segmentation will obtain a large bounding box that encompasses the whole orange path; thus, simply using the target point as the average of the bounding box coordinates does not give the correct x coordinate in the real world. An incorrect target point will lead to future errors in the control component of the module. To combat this issue, the image is cropped to use only the bottom third, and it is further split laterally into three regions to account for four possible cases:

1. If the top-left point of the bounding box is in the left third of the image, then the x pixel value for the target point is biased to be more left (see Figure 5).

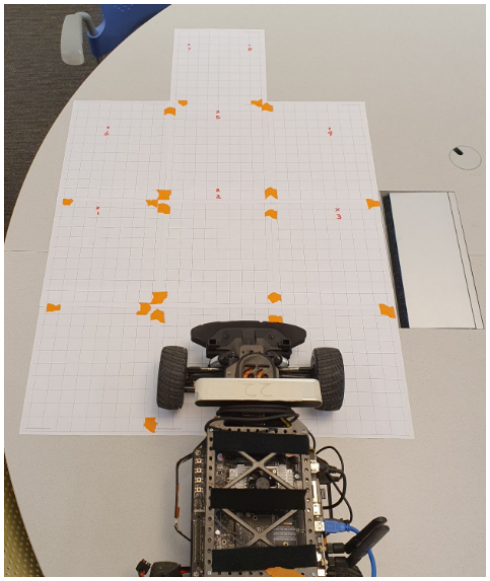


Figure 4: Illustration of how the homography transform is obtained. Since all orange lines lie on the ground plane and have a fixed relationship with respect to the car's camera, a homography matrix H can be precomputed by putting the car on grid paper to obtain corresponding pairs of pixel and real-world coordinates. OpenCV's `cv2.findHomography(src_Points, dst_Points)` function obtains the corresponding homography matrix. Points in the pixel frame can then be projected onto the real-world surface plane using Equation 5. (Figure Credit: Nihal)

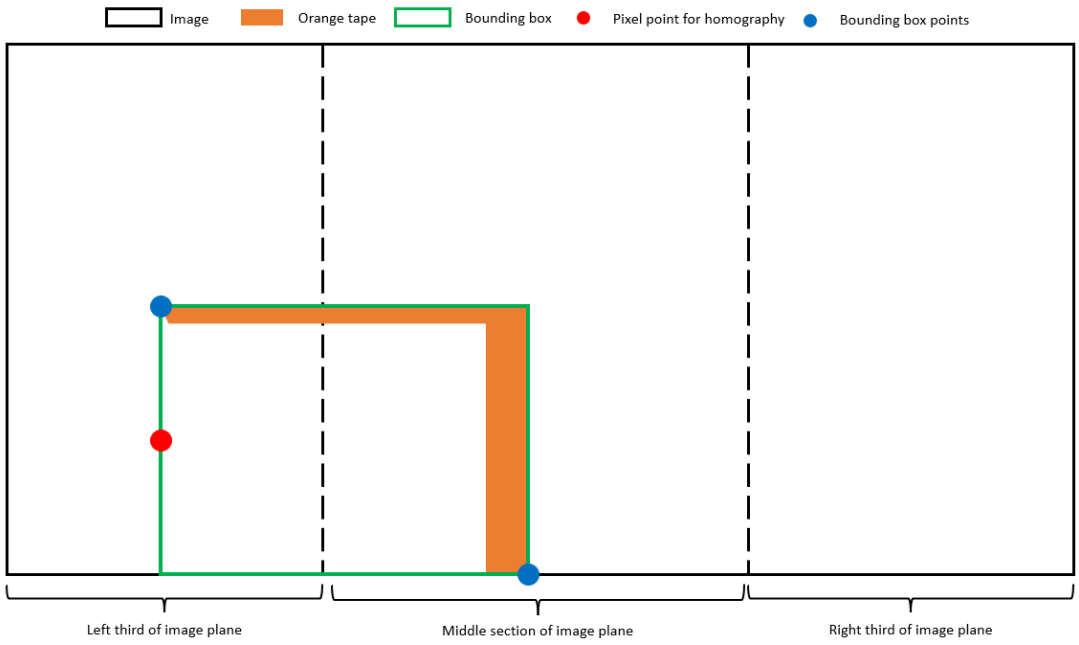


Figure 5: Case 1: Top-left point of bounding box is in left third of the image. By biasing the target pixel point to the left, it ensures a larger steering angle is commanded to turn the car left and ensure the sharp turn can be executed. (Figure Credit: Nihal)

2. If the bottom-right point of the bounding box is in the right third of the image, then the y pixel value for the target point is biased towards the right (see Figure 6).

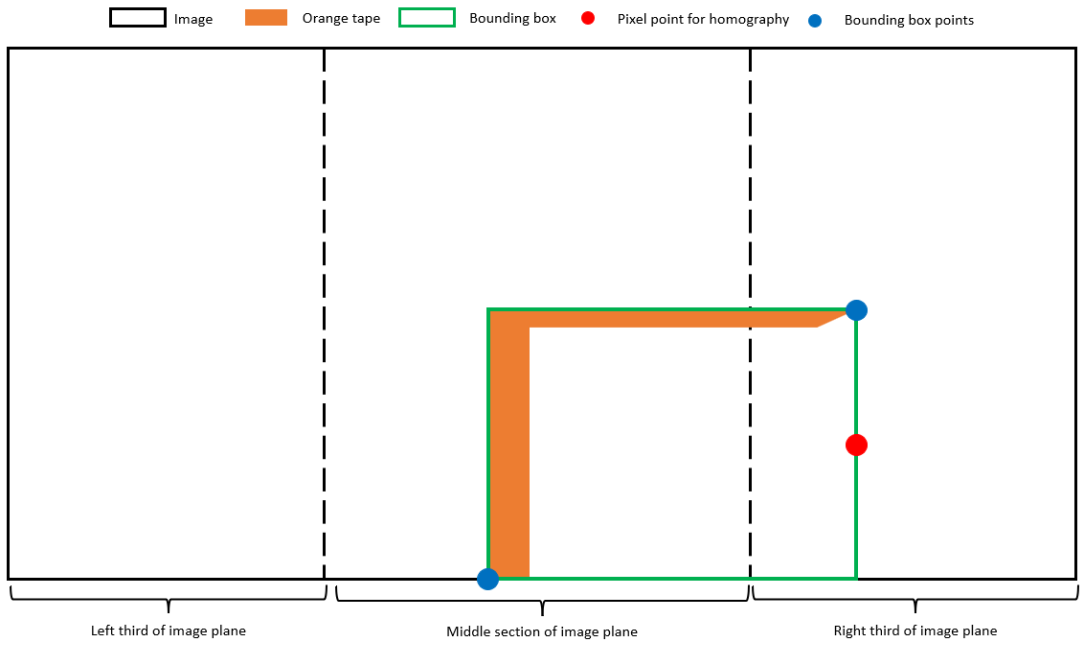


Figure 6: Case 2: Bottom-right point of bounding box is in the right third of the image. By biasing the target pixel point to the right, a larger steering angle is commanded to turn the car right and ensure the sharp turn can be executed. (Figure Credit: Nihal)

3. If the top-left and bottom-right points are in the middle third, then the average of the x and y pixel points are taken as the target point (see Figure 7).

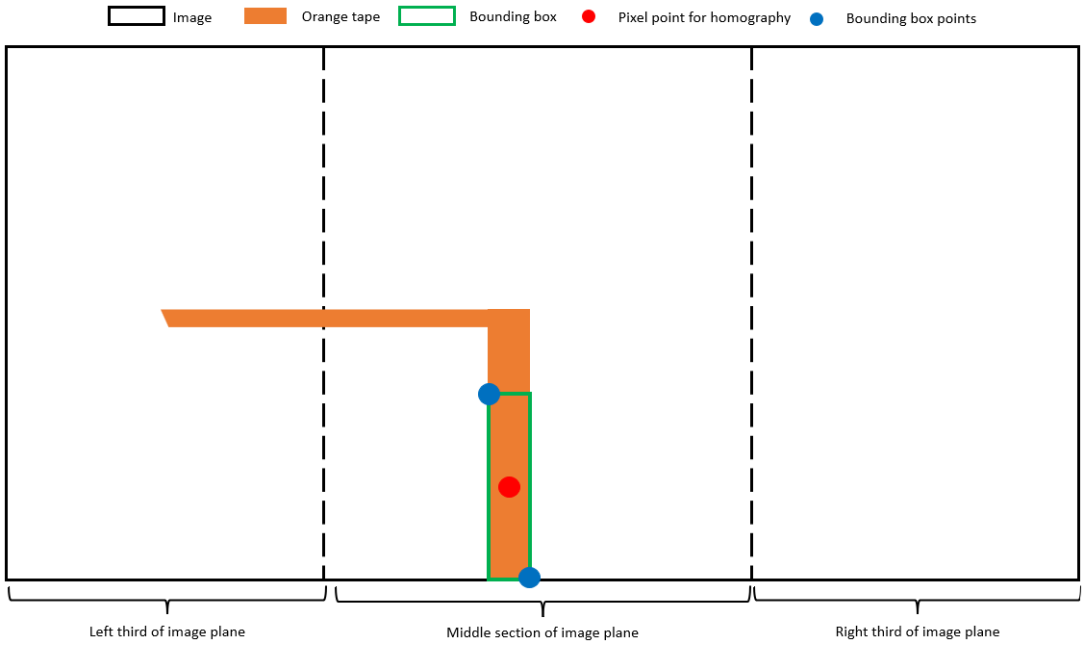


Figure 7: Case 3: The bounding box stays within the middle third of the image. The target point is simply taken as the average of the bounding box coordinates. (Figure Credit: Nihal)

4. If the top-left and bottom-right points are in the left third and right third, respectively, then the average of the x and y pixel points are again taken for the target point (see Figure 8).

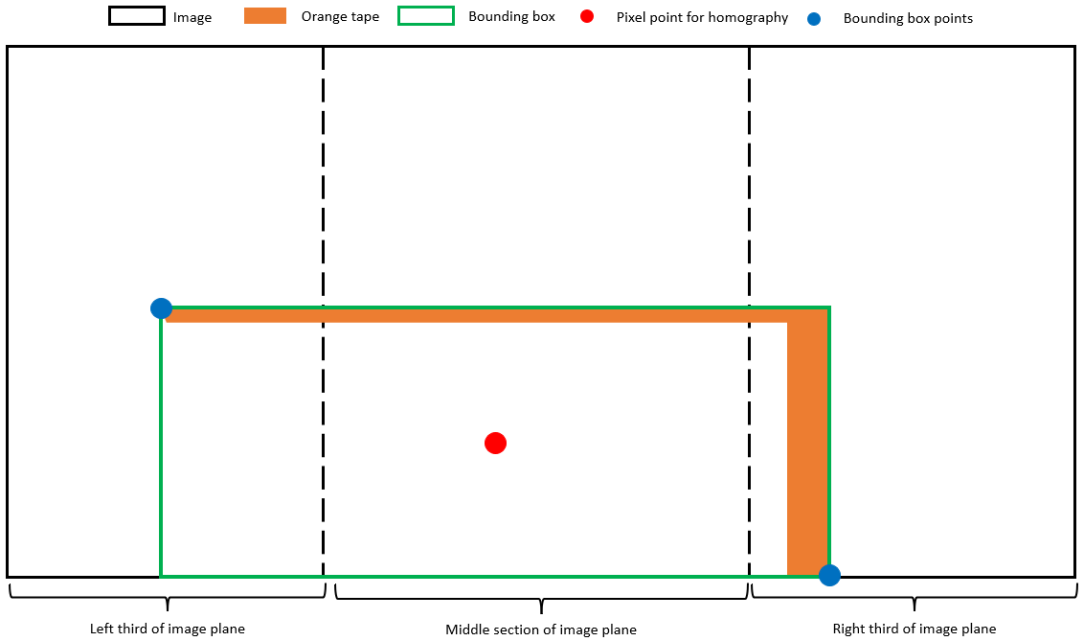


Figure 8: Case 4: The bounding box extends to both the left and right thirds of the image. The target point remains as the average of the bounding box coordinates, and no bias is added. (Figure Credit: Nihal)

Thus, color segmentation in conjunction with additional bounding box logic is used to successfully determine the target point that will be eventually used to control the car.

2.1.3 City Driving - Stop Sign Detection (Chuyue)

Another component of the City Driving challenge is being able to follow certain rules of the road; more specifically, being able to appropriately perceive and stop at a red stop sign. Due to the stop sign's unique red color, color segmentation is again used to filter out the stop sign. Similar to the orange line detection, detecting the stop sign involves obtaining the bounding box, extracting a target point, and converting the target point via homography.

Obtaining the bounding box: To obtain the color segmentation mask for the color red, two masks are overlapped for the hue (as red corresponds to the ranges $h \in [170, 180]$ and $h \in [0, 10]$). The saturation and value ranges are then tuned with respect to the lighting present in the Stata basement ($s, v \in [120, 255]$) to ensure the color segmentation mask can accurately detect the red stop sign.

However, color segmentation may detect red-colored pillars that are present on the ceiling in the Stata basement and erroneously perceive them as the red stop sign. Similar to detecting the orange line, this is accounted for by cropping away the upper part of the image and conducting additional thresholding to discard bounding boxes whose width or height is below a certain value.

Extracting the stop sign's ground point: Color segmentation returns the top-left and bottom-right coordinates of the bounding box in the image frame; however, the homography transform can only project coordinates in the image plane to the real world's

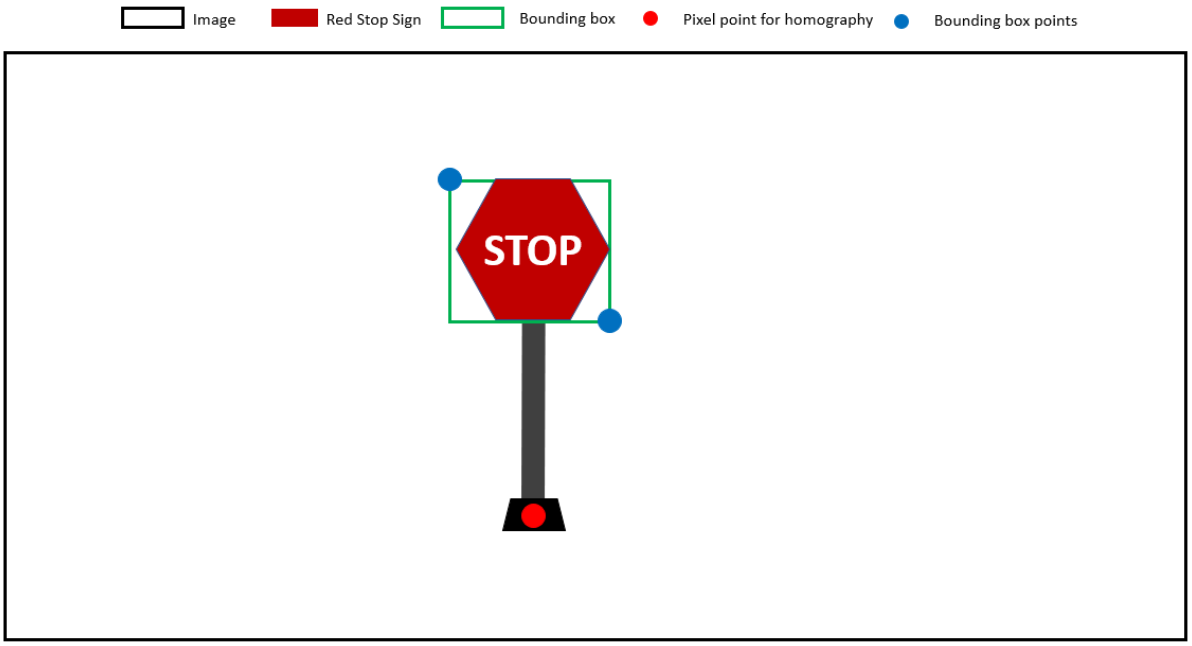


Figure 9: Illustration of the bounding box and base point of the stop sign. Although color segmentation obtains the coordinates of the stop sign (green), the coordinates can not be used in homography because the transform can only transform points from the pixel frame to the ground plane – and the stop sign plate itself is not located on the ground. Thus, measurements of the stop sign's height is used to obtain the pixel point at the base of the stop sign, taking advantage of the fact that scale is preserved under homography transformations. With the base point (located on the actual ground), the same homography transform can be used. (Figure Credit: Chuyue)

ground frame (and the actual stop sign is suspended in mid-air). To obtain the point denoting the base of the stop sign (on the ground plane), the heights of the sign plate h_r and the whole sign h_w (including the pole) are obtained. Since scale is preserved under homography transformations, the coordinates of the base of the stop sign (in the image frame) can be obtained using the following equations:

$$p[0] = p_1[0] + \frac{h_w}{h_r}(p_2[0] - p_1[0]) \tag{6}$$

$$p[1] = \frac{1}{2}(p_1[1] + p_2[1]) \tag{7}$$

Figure 9 illustrates how this pixel point is obtained.

Obtaining the distance to the base of the stop sign via homography: Once the coordinates of the ground point of the stop sign is obtained in the image frame, the same homography transform can be used to estimate the projected position of the stop sign's actual base point in the real world (refer to Section 2.1.2). With the base point obtained in the ground plane, the system can utilize it in the control component of the module to execute the desired actions.

2.2 Control (edited by Aaron)

After key target points are obtained from the perception components of both modules, control algorithms are needed to obtain the appropriate driving commands that allow the

car to execute the desired motions. In this section, we discuss our adapted usage of the pure pursuit algorithm for both modules as well as the additional controller logic needed for the City Driving portion of the final challenge.

2.2.1 Pure Pursuit (Jessica)

For both the Final Race and City Driving modules, a pure pursuit algorithm is implemented (as in Lab 6) due to its simplicity. This algorithm calculates δ, the angle the car needs to drive to reach the target point (provided from the perception component of both modules) head-on. The pure pursuit equation is shown in Equation 8 and its variables are represented visually in Figure 10.

$$\delta = \arctan\left(\frac{2Lsin(\theta)}{Distance}\right) \tag{8}$$

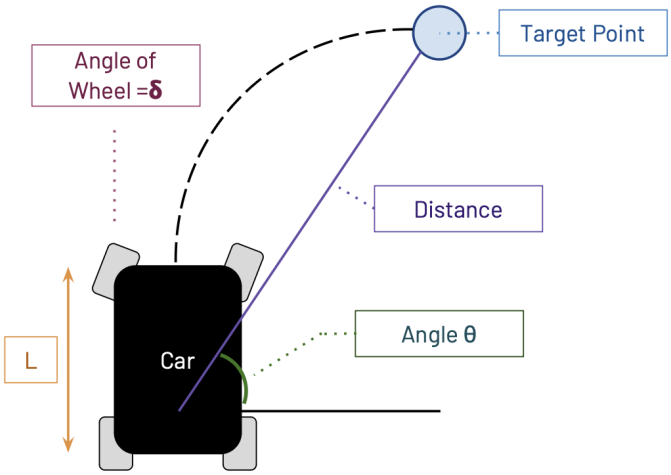


Figure 10: Illustration of the parameters for the pure pursuit equation. The pure pursuit algorithm determines δ, the angle at which the wheels need to turn in order to reach the target point. The equation utilizes the values shown: L, the length of the car; θ, the angle of the car to the target point; *Distance*, the distance from the car to the target point. (Figure Credit: Jessica, reused from Lab 6 Report)

Certain adjustments are made to make the pure pursuit algorithm specific to and more effective for the Final Race and City Driving modules specifically, such as tuning the lookahead distance, accounting for data loss, and increasing the reactivity of the control. Each modification is further explained in the following sections.

Lookahead Distance:
The *distance* component of a pure pursuit algorithm is a tunable value. From past labs, decreasing the *distance* causes sharper turns to be calculated, but this may result in increased oscillations. It is necessary to tune the lookahead distance for both challenges.

- Final Race: The target point is obtained via the intersection of the track lines on either side of the car; however, this intersection point may not accurately account for the actual vertical distance needed. When translated with homography, the intersection point, which generally is in the center of the image, is measured to be around 42 meters vertically away from the car. This large distance causes the car to maintain a straight trajectory regardless of if a turn is needed to stay within the track lanes. To address this issue, only the horizontal y component (with respect to the car's coordinate system) of the target point is kept to provide the necessary information about the car's location in the lane. The vertical x value then becomes a tunable parameter corresponding to the lookahead distance, which can be used to ensure oscillations are reduced and that the car can effectively turn and stay within the track lines.

- City Driving: In City Driving, the target point is essentially obtained via the "average "position of the bounding boxes encasing the detected orange line (see Section 2.1.2). The lookahead distance effectively decides which portion of the image will be used to determine the target point. By removing the top or bottom of an image, the bounding boxes move closer to or further away from the car. Thus, the lookahead distance can be adjusted to reduce oscillations, but more importantly ensure the car can execute the sharp 90° turns specified in the city.

Solutions for Data Loss: As computer vision and control algorithms are imperfect, a loss of data may occur (e.g. no lines are erroneously detected); thus, no target point is obtained for the pure pursuit algorithm. Additional cases are implemented to ensure the car's controls are robust to such situations

- Final Race: The main priority of the Final Race part of the challenge is to be able to finish the lap quickly. If lines can not be detected for some reason, the car must continue forwards with a likely chance of recovering the data; but, it must do so safely and stay within the lines. Observations discovered that lines were often lost when the car became too close to the left side of the track (as it would see the left line as too straight and thus would be filtered out via our implemented computer vision algorithm). As a result, the car would drift left and out of the lane. To address this situation, a special case is added to the logic of the pure pursuit controller to command a very small steering angle of $-0.5°$ when the left line is missing. This allows the car to shift slightly to the right and avoid crossing over into the left lane. The small angle magnitude further ensures that a large correction isn't suddenly forced onto the race car and primarily encourages it to continue straight so it can finish the lap as quickly as possible.

- City Driving: In the city, there are many sharp turns in close succession that the car must navigate; thus, it is also likely that the car loses sight of the orange line when navigating such turns. To allow the car to continue turning and eventually recover the orange line, a solution is implemented where the car maintains its last commanded steering angle if there is a loss in data. This allows the car to make the necessary sharp turns in both the right and left directions without risking the car's ability to safely execute forward trajectories on straights.

Reactivity of the Control: Previous labs demonstrate it is difficult to tune the lookahead distance to precisely execute sharp turns. If the lookahead distance is too close, the

car does not react fast enough to complete the turn; however, if the lookahead distance is too far, the car will cut corners. This is especially relevant to the City Driving module.

- City Driving: Previously, there were less implications with the car cutting corners. In City Driving, however, the boxes prevent wide turns and large corner cutting to happen – otherwise, collisions may occur. To limit deviations from the path during turns, a smaller lookahead distance needs to be used – but to ensure turns can be successfully completed without collisions, a tunable coefficient is multiplied to the car's steering angle. This allows the commanded steering angle to be more sensitive and allow the car to execute turns more reactively even at shorter lookahead distances.

2.2.2 City Driving - State Machine (Chuyue)

Although the pure pursuit controller is sufficient enough for the Final Race module, additional controller logic must be implemented for the City Driving module for it to register different cases pertaining to the rules of the road (i.e. the stop sign). To take care of both line following and stopping at the stop sign, a state machine is used so that the controller can switch between these two cases. A node is created to decide and publish the state of the car, while also subscribing to the topic published by the stop sign detector. Specifically, the two states are:

- State 1: Stopping State (stop sign is perceived)
- State 2: Line Following State (run pure pursuit on the target point)

The state machine logic is as follows, utilizing the stop sign detection process (see Section 2.1.3):

- The system begins in State 2, with the car executing the line follower.
- If the stop sign is detected to be within a specified distance threshold (and the car has not changed into State 1 yet), the state will be changed from State 2 to State 1. For the purposes of this challenge, an assumption that only one stop sign is present in the city is made.
- The system stays in State 1 for no more than 0.5 seconds (as the car must continue moving in order to fully execute the path within a reasonable time). The time is stored upon entry to State 1 to allow the system to know when to transition out of State 1 back into State 2.
- In all other cases, the system defaults to State 2 and follows the orange line.

The controller subscribes to the state machine topic and thus alters the control commanded to the race car depending on the current state of the system. If the system is in State 2 (Line Following State), it will use the pure pursuit algorithm and the target point detected from color segmentation to publish the appropriate driving commands needed to follow the line. If the system is in State 1 (Stopping State), the velocity and steering angle are simply set to 0. Algorithm 1 summarizes the state machine logic in pseudocode.

Algorithm 1 State Machine for City Driving

Require: ROSPY.GET_TIME

1: $state \leftarrow 2$
2: $see\_stopsign \leftarrow False$
3: $threshold \leftarrow 1.4$
4: **procedure** STATE CONTROL(Stopsign)
5: **if** Stopsign.$detected == True$ **and** Stopsign.$distance < threshold$ **and** $state == 2$ **and** $see\_stopsign == False$ **then**
6: $state \leftarrow 1$
7: $park\_time \leftarrow$ ROSPY.GET_TIME()
8: $see\_stopsign == True$
9: **else if** $state == 1$ **and** ROSPY.GET_TIME() $- park\_time < 0.5$ **then**
10: $state \leftarrow 1$
11: **else**
12: $state \leftarrow 2$
13: **end if**
14: publish state

3 Experimental Evaluation

To ensure both the Final Race and City Driving implementations were successful, it was important to empirically examine the resulting motions. In both modules, the resulting motion of the race car should be smooth; in other words, oscillations and deviances from the intended path should be minimized. In turn, this allows the car to achieve the fastest possible time around the track (being held at a speed of 4 m/s) and also precisely navigate the city. Parameters associated with the target point were tuned, and their effects on the car's performance were captured by plotting the lateral distance error from the target point. The results of such qualitative and quantitative evaluations suggested our systems were ready for final testing on race day.

3.1 Final Race (Aaron / edited by Nihal)

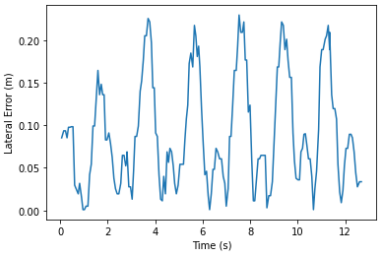
Since the velocity of the car is capped at 4 m/s (via the challenge's specifications), the car's oscillations during the lap must be minimized in order to achieve the fastest time. Additionally, the system must be robust to operate successfully on both turns and straights, as well as different lanes. On inner lanes, the car is required to turn slightly more than on the outer lanes because of the effectively smaller radius on the inner lanes. Theoretically, a larger lookahead distance should be beneficial during straights and on outer lanes, as the target point has a smaller effect on determining the steering angle; on the other hand, a smaller lookahead distance allows steering angles of larger magnitudes to be commanded, perhaps being more beneficial during turns and on inner lanes.

Qualitative observations suggested what value the lookahead distance should approximately be. When the lookahead distance was small (225 pixels from the top of the image), the car behaved in a very oscillatory fashion. This suggested that the lookahead distance needed to be farther away from the car, as larger commanded steering angles attempted to over-correct the car's motion. When the lookahead distance was large (e.g.

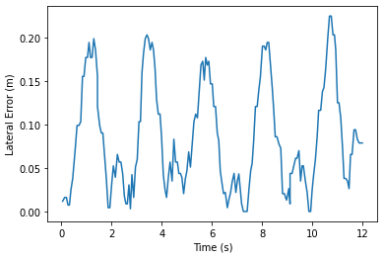
220 pixels from the top of the image), oscillations were reduced; however, this resulted in the vehicle being unable to sufficiently turn and stay within the lane. A lookahead distance in the middle of 222 pixels from the top of the image provided a stable trade-off as it reduced the car's oscillations and allowed the car to stay between the lines.

Quantitative data collected for the lateral error of the car further supported the qualitative observations. The lateral error is defined as the distance between the car's horizontal position and the horizontal position of the intersection point. Thus, a car that is tuned perfectly should theoretically obtain an error of 0 throughout its execution; that is, the commanded steering angle allows the car's horizontal position to exactly match that of the target point throughout the entire motion. Furthermore, the standard deviation of the error should be low, meaning that the spread of lateral error margins is minimized and the car refrains from large oscillatory motions.

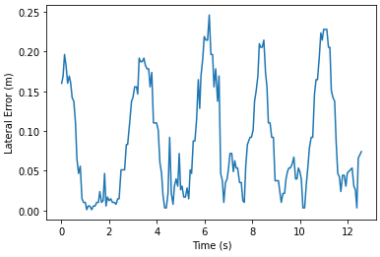
For turns, larger lookahead distances (220 pixels from the top of the image) resulted in smaller standard deviations (0.0640m) but larger average errors (0.0915m), suggesting the motion was less oscillatory but also less able to stay to follow the target point. Smaller lookahead distances (225 pixels from the top of the image) resulted in larger standard deviations (0.0694m) but smaller average errors (0.0875m), suggesting that the car is more able to follow the target point but oscillates heavily to do so. Choosing a lookahead distance in between the tested values (222 pixels from the top of the image) resulted in a better performance that minimized the average lateral error to the target point (0.0902m) via a smoother trajectory (smaller standard deviation of 0.06326m). Figure **??** illustrates the error plots associated with each lookahead value. Similar findings occurred during straightaways as detailed in Figure 12.



(a) Lateral error with a far lookahead (220 pixels from the top of the image). The average error is approximately 0.0915m and the standard deviation is 0.0640m. Oscillations are smaller but the error is larger, suggesting this lookahead value makes it harder for the car to turn and follow the target point accurately.

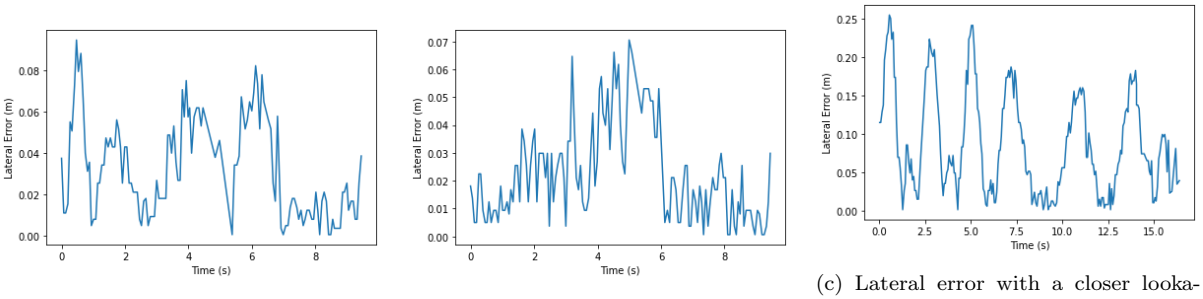


(b) Lateral error with a lookahead of 222 pixels from the top of the image. The average error is approximately 0.0902m and the standard deviation is 0.0632m. Not only is the car able to follow the target point better but also keep oscillations at a minimum.



(c) Lateral error with a closer lookahead (225 pixels from the top of the image). The average error is approximately 0.0875m and the standard deviation is 0.0694m. Although the car can more closely follow the target point, oscillations are larger which may result in the car effectively traveling slower around the track.

Figure 11: Plots illustrating the lateral error of the race car as it executes turns at different lookahead values. Notably, a small change in the pixel values resulted in large performance changes in the race car. The average lateral error is a measure used to determine the precision of the car to follow the target point, and the standard deviation is a measure used to quantify how oscillatory (spread out) the error values are. These tests suggest that a lookahead value of 222 pixels (from the top of the image) is an appropriate value for the race car. (Figure Credit: Shara and Aaron)



(a) Lateral error with a far lookahead (220 pixels from the top of the image). The average error is approximately 0.0321m and the standard deviation is 0.0233m. With a large lookahead, the car has trouble correcting itself via steering when needed (resulting in a slightly larger error) but oscillations are relatively small.

(b) Lateral error with a lookahead of 222 pixels from the top of the image. The average error is approximately 0.0223m and the standard deviation is 0.0169m. This value for the lookahead distance allows the car to correctly steer itself as needed to minimize error from the target point and minimize oscillations.

(c) Lateral error with a closer lookahead (225 pixels from the top of the image). The average error is approximately 0.0876m and the standard deviation is 0.0659m. Although oscillations minimize over the straight, they in total are very large even on a straightaway and thus the car has trouble precisely following the target point with minimal error.

Figure 12: Plots illustrating the lateral error of the race car during straightaways using different lookahead values. Similar to the turning cases as in Figure 11, a middle value (222 pixels from the top of the image) that balances the inherent trade-off in the lookahead point leads to performance benefits. (Figure Credit: Shara and Aaron)

Ultimately, 222 pixels was decided as the best lookahead distance that could ensure that the race car could quickly and safely execute a lap around the track. Figure 13 plots the lateral error for a full lap with this lookahead.

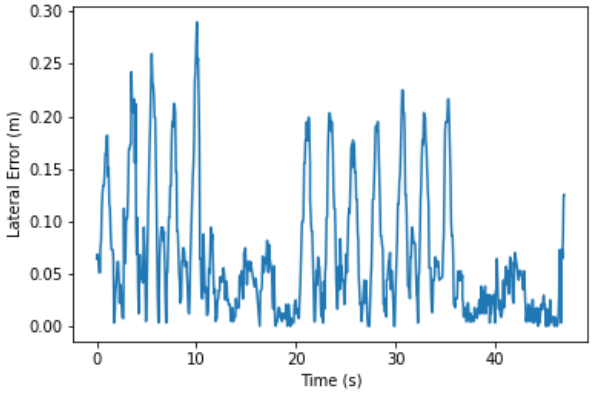


Figure 13: Plot of the lateral error of the race car during the execution of a complete lap. Using a lookahead value of 222 pixels (from the top of the image), the race car is able to smoothly and quickly complete the track without exiting the lines. The average error is 0.6770m from the middle of the track with a standard deviation of 0.0620m. (Figure Credit: Shara and Aaron)

3.2 City Driving (Nihal)

For City Driving, the goal is to ensure that appropriate steering commands are given for the car to accurately follow the line. This is especially challenging, for instance, when dealing with 90° turns, as the resulting steering angle should not be heavily influenced by the increasing size of the bounding box. The car's speed served as the primary tuning parameter to achieve the desired motion of the race car.

In general, the speed was tuned qualitatively through trial-and-error test runs. Initially, the car was tested at 1 m/s, but at that speed the car reacts too late to safely execute the corner turns. Therefore, the speed was progressively decreased until 0.5 m/s, when the car was able to follow the road well enough. Figure 14 displays the lateral error of the car with respect to the target point at 0.5 m/s.

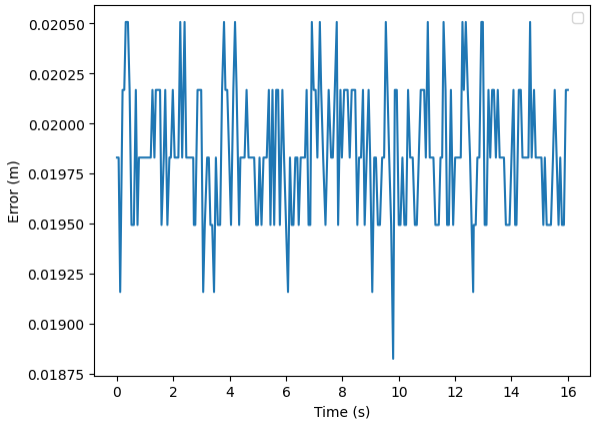


Figure 14: Plot of the lateral error of the race car with respect to the target point obtained via the bounding box encasing the orange line. At a speed of 0.5 m/s, the car is able to follow the line well with an average error of around 2 cm from the middle of the line. (Figure Credit: Shara)

As the stop sign was also an important aspect of the City Driving module, the car was tested to stop at three different locations at a distance of 1m away from the sign: after the first left turn, after the first right turn, and after the second right turn. The car appropriately stopped for the first two test cases; however, the car did not stop in the last case. This was perhaps because the car had difficulty recognizing the stop sign at an angle (as it came out of a very tight turn). Since the stop sign would only be placed at the first turn on Race Day, it was decided that our system was robust enough for the purposes of the final challenge and thus the last case should be left as a future area for improvement.

4 Final Testing on Race Day (edited by Aaron)

Although our systems for both the Final Race and City Planning challenges had been working successfully leading up to Race Day, different executions were seen during the official test runs. An analysis of the resulting performance of our system on Race Day leads to additional lessons and insights that can be applied for future scenarios involving final live demonstrations.

4.1 Final Race (Aaron / edited by Nihal)

Our system had been performing very well throughout testing, achieving clean laps with minimal oscillations; but on the official test laps, the car ended up failing to stay within the lane lines. This was a big surprise to our team, as many successful practice runs conducted the night before suggested our system was working robustly. Upon further investigation, however, we realized that the wrong version of the code had been published

to the car by accident. Upon pushing the correct version of the code, the race car's ability to successfully complete the lap (as it had been previously) was immediately restored.

Although this was extremely unfortunate (as our team had spent a tremendous amount of time to get our car working well and was not allowed a re-run), such an experience suggests a very valuable lesson in taking deliberate care to keep all files organized. Especially in instances where software is needed for a final live demonstration, processes such as code freezes can be very useful to ensure that such simple mistakes do not happen at the time of testing.

A video demonstrating our system's true ability to execute a penalty-free lap in 47.25 seconds can be found here .

4.2 City Driving (Nihal)

Similarly, previous testing suggested that our system should be able to execute the city path in a collision-free manner using the tuned parameters; however, when running the car at the experimentally-tuned 0.5m/s velocity, the car ended up being unable to execute one of the sharp turns. As a result, our team ended up deciding to further reduce the speed to 0.4m/s on our final attempt, to which we were able to complete the city without any collisions. This experience suggests how valuable it is to understand the parameters relevant to the system, especially during final performances. Our thorough investigation and knowledge of the fact that decreasing the velocity allows the car to execute the path more precisely via our experimental testing allowed us to adapt our system last-minute and make it successful for the final challenge. Navigating through the city for the successful last run took 16.2 seconds, and a video taken during the challenge can be found here .

5 Conclusion (Jessica / edited by Aaron)

In this lab, we successfully designed, implemented, and tested algorithms to complete the two modules of the 2023 RSS Final Challenge: Final Race and City Driving. For the Final Race, Canny edge detection was used on a gray-scale image to produce Hough lines. An intersection point was determined and pure pursuit was implemented, allowing the car to quickly drive around the track at high speeds while maintaining inside the lane. For City Driving, color segmentation was used to obtain the target points to follow the orange line and perceive the stop sign. A state machine was then used to implement pure pursuit to follow the orange line and have the car stop appropriately at the stop sign.

Quantitative and qualitative analyses were used to debug our code, tune our values, and ultimately determine the success of our modules. From our computer vision testing procedures, values for color segmentation were tuned to differentiate between the orange tape and the red stop sign. Furthermore, creative threshold was used to filter out unwanted lines on the track. Additionally, testing allowed for the lookahead distance and velocity to be tuned so that the pure pursuit algorithm could allow the car to travel effectively around tight corners and along straight paths. Ultimately, our choices were justified by demonstrating how the car can quickly traverse the track and precisely navigate the city.

Overall, the success of our final challenge represents the fundamental skills we have developed throughout the labs and this class. Not only were we able to build on the technical work done in previous labs, such as pure pursuit and color segmentation, but we also applied the various communication skills we have developed. As a result, we were able to construct a robust autonomous vehicle system that could perform well in the final challenge.

6 Lessons Learned

Aaron: On the technical side, I learned how valuable it is to not only create the code infrastructure for processes to work correctly, but also spend effort and creating methods to visualize the processes. For example, it was important to spend time being able to visualize the averaged hough lines and the intersection point on the image and print out certain values. These methods were extremely helpful in gaining an intuitive understanding of the black-box processes in our code, and also allowed for efficient debugging. On the communication side, I learned how it is important to have an organized plan to modularize work in group settings. Although it is nice if everyone on the team can all be together to work on the same aspect of the project at the same time, it is impractical as individuals' schedules may not line up and such processes can end up being inefficient. Instead, it is important to have an organized approach established by the team so that a plan of action can be done to work on modules in parallel at the same time, allowing each member's abilities to be used more efficiently and the group work to be done more quickly.

Jessica: First of all, I learned the importance of building intuition while testing. Whether you're tuning color segmentation or the lookahead distance, it's easy to mindlessly change values. However, it is much more effective to spend time considering why different values effect your system in different ways. With a better understanding and a more intellectual appraoch, values can be tuned more efficiently. On the communications side, I learned the importance of good documentation. Because we were working on two different projects and trying a range of different strategies to see what worked, it was vital we kept documentation of what had already been attempted and how well it worked in order to save time.

Chuyue: First, a simple yet robust algorithm should be aimed at. The code that theoretically works does need many designs and tunings to make it work as well in the testing situation. If the part to be tuned is complex, it will take us much time to tune without a clear direction. The finally working version turned out to be simple and effective. As for the final race, I realize the importance of being cautious and double-checking before the final hit. As for communication, since people have different schedules and strategies, it would be good to document people's ideas and try them out efficiently.

Nihal: From the technical side of things, I learned how important it is to test in the same conditions as you're going to be assessed. This affected us during the final race when the color segmentation was off since we tested at night due to our schedules not being free in the day whereas the race was during the day and the lighting conditions from the end of the hallway were different. From the communications side of things, I learned about the importance of scheduling with teammates while working on large projects and

better-articulating design work on control systems to a large background, especially when a lot of the work has been spoken about in reports from previous labs.

Shara: During the final challenge, I was able to learn more about how to organize code for big projects as well as work on them in parallel to many team members. Due to time constraints and work load, many team members were not available during the same times so it was important for us to find a way to structure and integrate our code so it could be ready for change and easy to understand. Delegation and organization was an important part of this project. On the more technical side of things, I learned how to tune values for a system more efficiently by observing the impacts changing a single value at a time had on our algorithms performance. I also learned how sensitive some values you tune can be and how dynamic environment variables such as lighting can impact the performance of your algorithms such as a computer vision algorithm. I also learned more about double checking the performance of your system before performing in a challenge/deploying because we had tuned some values on the day before to fit slightly different considitions and this had impacted how well our car did in the race.