RSS Final Challenge

Juan Alvarez, Joel Santiago-Baretti, Erica Chen, Mohammed Ehab, Ningshan Ma
RSS: Robotics, Science, and Systems
Team 2

May 13th 2024

In this paper, we present our solution for the two distinct parts of the final challenge of RSS (Robotics Science and Systems) 2024. For Part A, the race car leverages color segmentation and Hough transform techniques to detect and follow white lane marks around the Johnson track, minimizing speed time and number of collisions. Specifically, the car's system isolates the white lines on the track using color segmentation, then applies the Hough transform to interpret these lines for real-time pure pursuit. For second part of the challenge of the challenge, the vehicle transitions to a more intricate environment: navigating through the basement of the Stata building. The race car employs the A* path finding algorithm and localization algorithm implemented earlier in classwork to effectively localize itself and maneuver through the predefined environment.

1 Introduction

Do we need one or are the challenge sections enough?

2 Final Race

2.1 Challenge

In the first part of the challenge, Final Race, we are tasked with traveling around the Johnson track while staying within a specific lane under swiftly. The goal is to minimize speed and number of trespasses of the designated lane. We need to develop control systems that can detect and adjust to the turns of the lane and make sure that the vehicle remains on its intended path without sacrificing speed. The challenges lie in managing the high-speed dynamics of the robot while simultaneously updating driving commands based on real-time camera feedback. We wanted to travel at the upper limit of 4 m/s, which gives high constraint to our algorithm since the system must process and react to visual data almost instantaneously to maintain course accuracy.

2.2 High Level Approaches [Ehab]

At a high level, we needed to implement two subsystems for this challenge: a lane detection system and a controller for steering the robot. For steering, we quickly decided on a pure pursuit controller. We had implemented pure pursuit control in previous labs and demonstrated smooth and successful driving with it in our wall following module, so it was a clear choice.
As a black-box algorithm, a pure pursuit controller requires as input a lookahead point P that the robot keeps driving towards, so our lane detection algorithm has to output such a point P. That point corresponds to the point a distance L (the lookahead distance) from the robot that lies on the robot's desired path. The robot constantly drives towards that lookahead point P and updates it throughout its run, and hence for smooth driving, P has to be as close to the center of the lane as possible and change continuously with minimal noise. For details on pure pursuit, please refer to our report on path planning and following.
To obtain P, we developed two competing approaches in parallel that we will describe next.

2.2.1 Sliver Approach

In this approach, we slice a thin sliver of the image the robot sees that corresponds to the image at a distance of L. That slice should contain two white regions that correspond to the edges of the lane. We employ color segmentation to obtain a black-and-white image where white corresponds to the lane lines. We can then appropriately take the average of the locations of the white pixels to obtain our lookahead point P.

2.2.2 Hough Line Detection Approach

In this approach, an algorithm called Hough transformation is employed to detect lines in the image. The lines are then transformed from pixel space into their locations in the real world using a homography transform. These transformed lines give us a mathematical equations for, approximately, where the lanes are. A lookahead point P is then easier to compute using these concrete equations.

We decided on the latter approach for the following reasons: first, the computed lookahead point P is less noisy in the latter approach because instead of one sliver of the line, the latter approach uses the entire line. This is akin to fitting a line to many data points as opposed to relying on a few nearby measurements in data science: the line fitting approach is better at eliminating noise. Second, for a large lookahead distance L, the robot can no longer see the lanes clearly, and the former approach may no longer work. The latter approach, however, works for larger lookahead distances. This is critical for driving at high speeds where the robot needs to look far ahead before deciding on an action.

2.3 Lane Detection Details

2.3.1 Color Segmentation [Ningshan]

As a first step, we obtain a black and white image where white corresponds to the lanes and black corresponds to everything else. To perform that, we implemented color segmentation which involves setting upper and lower bounds for color values to identify relevant objects within the visual input. Determining these aforementioned values have proven difficulty and necessary of extensive fine-tuning. From previous labs, the process required manually adjusting the values in our program code, observing the effects on the bounding boxes, stopping the program, and repeating the process, which was a highly time-consuming cycle. To address this inefficiency, we developed a real-time system utilizing OpenCV sliders. This system displays the thresolded image of the HSV values as white pixels, and others as black, allowing us to see the impact of changes instantaneously as we adjust the slider values. Moreover, we integrated interactive sliders from OpenCV to adjust the color thresholds in real time and it significantly streamlined the fine-tuning process. This enhancement has not only accelerated our workflow but also improved the precision with which we can select color ranges for segmentation. We tested our slider and visualization algorithm using the orange cone provided to ensure accuracy.

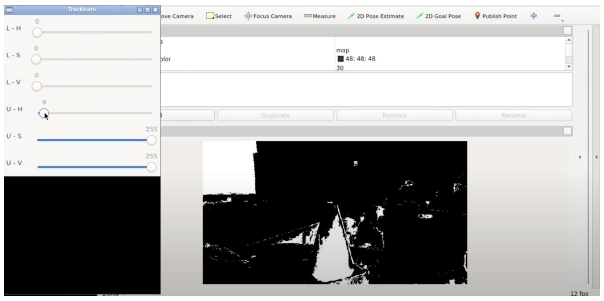
Figure 1: Sliders and fine-tuning process

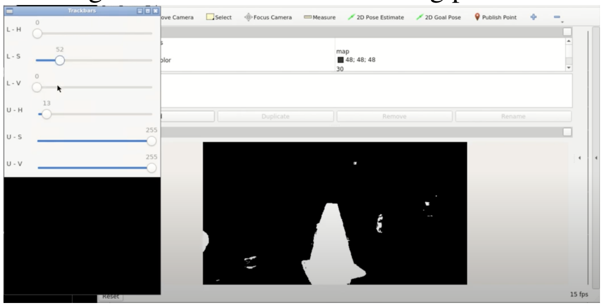
Figure 2: Result of fine-tuning

2.3.2 Hough Transform [Ehab]

After color segmentation, the Hough transform algorithm is employed to detect the lines in the image, giving us mathematical equations to describe the lanes. An example of the output of that transform is shown in Figure 3.

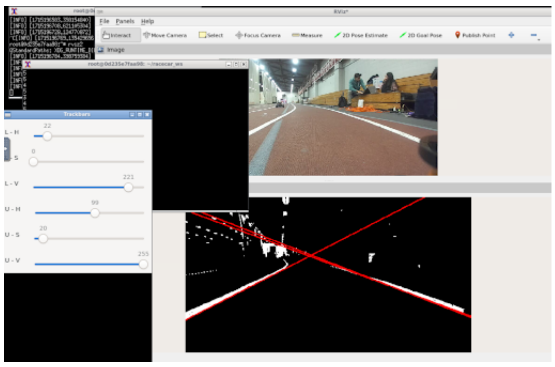
Figure 3: The lanes and background noise are shown in white, and the lines detected by the Hough transform are shown in red.

The red lines are in image space, but in order to obtain information about the lanes, we need to compute where they are in the real world. For that, a homography transform is employed. A homography transform is an algorithm that takes a pixel in image space and computes, approximately, its location on the ground plane. Since a line is defined by two points, taking two pixels on the line and passing them through a homography transform gives us two points on the ground that define the lane.

Next, we need a way to compute the lookahead point P given the lines. This is complicated by the fact that the lines do not correspond cleanly to lanes and can include background noise, or as seen in Figure 3, a single lane can be seen as multiple separate lines by the Hough transform. To mitigate the latter issue, we decided to cluster the lines such that each group of similar lines is represented by only one line, with the goal of picking the two most vertical lines (after the homography transform) to be the lanes.

Throughout experimenting with that, we discovered the algorithm does not always detect both lanes. For example, only the right lane is detected in Figure 4.

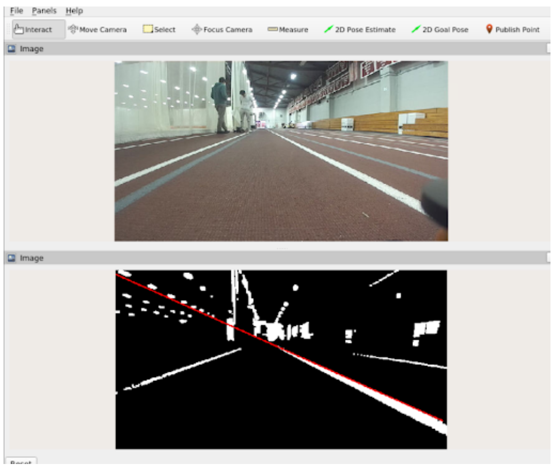

Figure 4: Only the right lane is detected even though the left lane is present in the image.

With that in mind, we needed to design an algorithm that is robust against that issue. We decided to pick the most vertical line to the robot's left as the left lane. If no such line is found, the most vertical line to the right is picked as the right lane. After that, the lane is shifted by 0.45 m (half the width of the track) to give us an equation for the line through the center of the track. The lookahead point P is then computed as the intersection of a circle of radius L around the robot with that line.

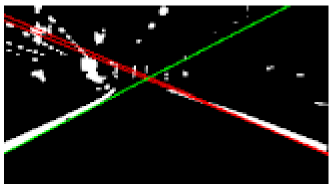

Figure 5: The left lane is highlighted in green.

Note the following important detail in our algorithm: the left lane is picked consistently when possible instead of letting the algorithm choose between the left and right lane. The effect is that the lookahead point varies more smoothly than if a different lane is picked arbitrarily every time.

2.4 Experimental Evaluation [Erica]

After we had our approach for using line detection and extrapolation, we needed to tune our parameters to be adequate at speeds of up to 4 m/s. The primary parameters that we changed was the maximum steering angle that the car was allowed to take and the lookahead distance.

Due to our hardware, our car's axis neutral axis was angled left. Therefore, we decided to account for this by differing the maximum steering angle that the car was allowed to take to the left and the right. At speeds of 2 m/s, we were able to have a lookahead of 1.5m and steering angles of $[\frac{\pi}{50}, 0]$, where the first number is maximium angle to the right, and the left is maximum angle to the left. With this implementation, we were able to successfully circle around the track with zero infractions in 121 seconds. However, since we wanted to prioritize speeds, we tuned our parameters at 4 m/s until we had a lookahead distance of 4.2m and angle limitations of $[\frac{\pi}{47.5}, \frac{\pi}{400}]$. With this set of parameters, for our best attempt in practice, we were able to travel the track at 47 seconds with one long breach into another lane. We deemed this adequate for our purposes, and proceeded with these parameters. In our best official run, we traveled the track in 49 seconds with 3 infractions. We believe that our HSV values were too narrow, and the curtains were moved, therefore casting shadows and lights that we had not accounted for in practice.

3 City Driving

3.1 Challenge

3.2 Path Planning [Erica]

For our path planning, we decided to use the implementation that we had created previously. We take in our occupancy grid that represents the map space and feed it into an A* algorithm with a Euclidean distance heuristic. To adapt our code to follow the rules of the traffic and stay on the right side of the road, we took the trajectory of the middle lane and added it as an obstacle in the map representation. This meant that when A* searched through the map, it would avoid crossing the lane, similar to a wall. Additionally, to adapt our code to stop at the three shells, we plan each path sequentially and stop planning once we reach a radius of 1m from the shell. This 1m buffer in the goal point allows our paths to be smoother and more flexible.

Since we decided to make the entire middle lane an obstacle, our paths would require us to travel significantly further and fully go around the Stata basement before we could make a U-turn and travel back in the right lane. Our solution to this was to see the location of the farthest shell from the start point, then ”cut off” the line barrier so the car could make a U-turn onto the other side of the lane. However, due to time constraints we had to manually cut the lane in our code. While it was successful in getting a significantly shorter path, given more time, we would have implemented this more dynamically.

3.3 Traffic Light and Stop Sign Detection [Erica]

We implemented a similar approach for our traffic light detection using our line detection method for Johnson track. We were able to use our HSV sliders to accurately tune the camera and only recognizes the red light when it is on and not off. However, regardless of tuning, there were objects in the background, such as a bright red board that we were unable to isolate using just color segmentation. So to make our system more robust, we similarly implemented hough circles to detect round objects. This successfully isolated only the red on light while filtering out the background noise.
For our stop sign detection, we integrated the TA created code which would use machine learning to recognize when a stop sign was in the area. However, due to significant runtimes and delays, we elected to prioritize other functions over this.

3.4 Integration and Evaluation [Ehab]

We tested our path planning through the shell locations marked in red in Figure 6. We found a path that makes a loop around the center of Stata then a U turn to return to the start.

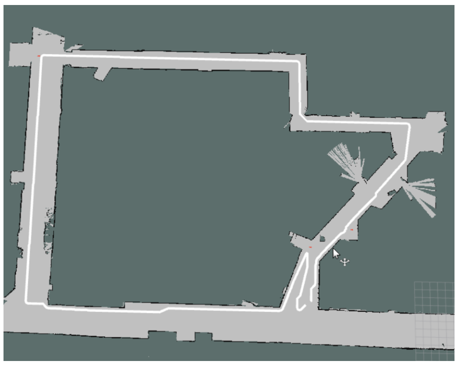

Figure 6: Initial trial of path planning

With the knowledge of the locations of the shells on race day, it was clear there is a shorter path that makes a U turn after the second shell. We shortened the lane to end right after the second shell, allowing the robot to discover that U turn. After that optimization, we managed to obtain a shorter path shown in Figure 7. The main challenge with that path was completing the U turn. The corridor was too narrow for the robot to complete it by steering and moving forward. To amend this issue, a command was added where if the robot is close to hitting a wall, it has to back up. This not only enabled the robot to complete the U turn, but it helped the robot recover from bad positions where it is driving too close to the wall.
With that implemented, we managed to make a run around the State basement where the robot stopped within 1 meter of each of the three shells and returned to the start. The run took 2 minutes and 40 seconds to complete at a speed of 0.75 m/s.

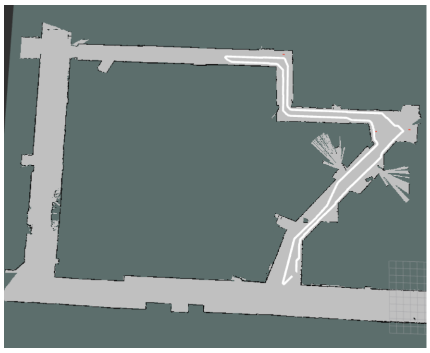

Figure 7: More optimal path planned with a U turn in the middle.

4 Lessons learned by person

- Ehab: I learned that sometimes small practical hacks is all it takes. I was trying hard to get more accurate lane detection or steering control, but limiting the steering angle was just enough and makes a lot of sense given the context.
- Ningshan Ma: I learned it was important to collect data in advance such as videos recorded at the Johnson video or with the traffic light so we don’t have to test things at the very last minute on site. I also learned that not putting all the efforts I could put in could leave regrets. The night before the race day, due to worry for health issues I didn’t pull an all-nighter that some of my teammates did and I felt an extensive amount of regret afterwards because I feel like I should have prioritized my team our performance at the raceday.
- Joel Santiago-Baretti: I was able to learn about broad topics in controls, localization, and how they are integrated into a single system. I was taken very out of my comfort zone since I am not a programmer, however I enjoyed learning about how localization algorithms work and seeing them be integrated into our final challenge.
- Erica: In this final amalgamation of our labs, I realized that there are multiple ways to approach the same problem. What intuitively seems like the

easiest solution may not be the most practical, and it is worthwhile spending time in the beginning to debate the pros and cons. ie. The sliver method seemed simpler due to it's similarity to previous labs, but it was far inferior to our hough method due to the speed and lookahead specifications in the Johnson track scenario.