

Final Challenge Report: Final Race and Obstacle Avoidance in TESSE

Team 11

Adi Mehrotra
Devin Murphy
Emma Griffiths
Tong Zhao

6.141 Robotic Science and Systems (RSS)

July 30, 2024

1 Introduction (Emma)

Cars need drivers. Or do they? As the capabilities of robots advance, fully self driving cars are inching closer and closer to reality. The prospect of self-driving vehicles comes with the possibility of safer roads in the future. The National Safety Council estimates 38,000 people are killed in car accidents every year and an additional 4.4 million are seriously injured.¹ The ultimate goal is to have the vehicles sense obstacles more effectively than humans and communicate with other vehicles on the road to avoid accidents and reduce fatalities. This fact is one thing that inspired and motivated us to take this class.

In order to create a collaborative self driving car network, we need a way to determine the location of each vehicle, including its position and orientation. This is the problem of localization. We solved this problem in Lab 5. Once we knew where our car was, we needed to learn how to drive and navigate. In Lab 6, we took on path planning and control. We used Pure pursuit control to navigate along a planned trajectory. Now that we've solved some of the major issues that are paired with the operation of autonomous vehicles, we can use the knowledge we've collected to push the boundaries of our system and test its potential and capabilities. This will be in the form of two final challenges.

The first challenge we took on was a race around Windridge City. The race course path is shown below in Figure 1. Our goals for this challenge were to successfully complete a lap around the city as quickly as possible and with no collisions. On a very high level, this was to be done by designing the final race

¹<https://www.nsc.org/road-safety/safety-topics/fatality-estimates>

trajectory ourselves, using the TESSE car’s ground truth localization, and using pure pursuit path following. Collisions were also tracked via logging, which was very important to do since VDI was used for this challenge, and we could not see the car’s point of view. We will go more into detail about our overall design and optimizations for this challenge later in the report.

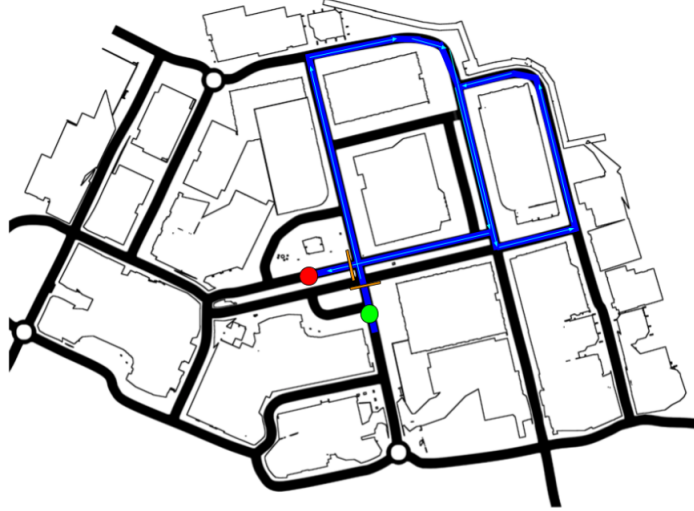


Figure 1: A lap around Windridge City. The final race course is shown in blue, with arrows marking the trajectory direction. Green marks the start and red marks the end of the path. The orange lines denote when race timing will begin and end.

The second challenge we took on was a fast obstacle avoidance challenge. The course path for this challenge is shown in Figure 2 below. Our goals for this second challenge were to successfully navigate along the entire obstructed road course to the destination with minimal collisions, ideally none, going as quickly as possible. Additionally, we wanted to ensure that our solution was robust to varying numbers of obstacles. On a high level, this was done utilizing TESSE’s LiDAR to detect obstacles, path planning around the obstacles, and tracking collisions. We will go more into detail about our implementation throughout the report.

Ultimately, our solutions to each challenge will be evaluated based on how quickly the challenge was completed and how many collisions occurred. Additional evaluation metrics will be further discussed throughout the report.



Figure 2: The course path for the fast obstacle avoidance challenge. The obstructed road is to be populated with a random number of obstacles fewer than 19 that do not obstruct more than 50% of the road width each.

2 Technical Approach

2.1 Final Race (Emma)

As mentioned earlier, the final race track path controller used was pure pursuit. Pure pursuit is a control method chosen to closely and smoothly track a reference trajectory, which made it a great choice for the final race challenge. A reference point is first chosen on the desired trajectory a predefined look ahead distance from the car. Then, a steering angle and velocity are determined so that the car would hit the chosen reference point if the angle was kept constant. Figure 3 shows a simplified example diagram of the Pure Pursuit method, and this will be used to explain how Pure Pursuit was implemented in the following sections.

2.1.1 Determining a Reference Point (Emma)

There are two major steps in selecting the reference point for Pure Pursuit. First, we determine which segment of the trajectory is currently closest to the car so that we have a better sense of how far along the trajectory our short term navigation goal point should be.

The shortest distance between a point and a line can be found by drawing a perpendicular line to the original that goes through the point. The distance between the point of intersection of the two lines and the original point will be the shortest distance. An example of this being used to calculate the shortest distance between the car and each trajectory segment is shown in Figure 4. If the

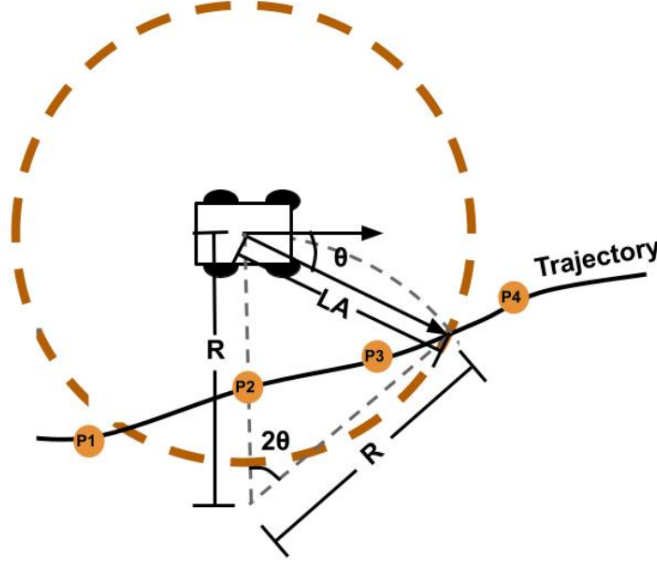


Figure 3: The Pure Pursuit method. LA marks the chosen look ahead distance. θ marks the steering angle. The desired trajectory is marked in black, and the orange points outline the different segments within the trajectory. The reference point that would be chosen is marked in red.

perpendicular intersection does not lie within the boundaries of the trajectory segment, as seen by the P3-P4 segment in Figure 4, the endpoint closest to the intersection with the perpendicular line will be used to calculate the shortest distance. In the case of the P3-P4 segment, P3 would be used. Then, the distance between the car and each of the intersection points will be determined for each trajectory segment. The shortest distance determined will correspond to the closest line segment. Next, the reference point for the controller will be chosen. Only trajectory segments starting with the closest to the car will be considered.

A tunable look ahead distance (LA) was used to select a reference point on one of the trajectory segments. As shown in Figure 3, a circle is drawn centered at the car's position with a radius equal to the look ahead distance.

Any point on a line segment can be written as

$$P1 + t(P2 - P1)$$

where P1 represents the segment start point vector, P2 the end point, and t is a scalar between 0 and 1. Because the look ahead radius is known, it can be set equal to the distance between the intersection point (call it X) and the car location (call it C). Let $V = P2 - P1$. Once t is solved for, the intersection point

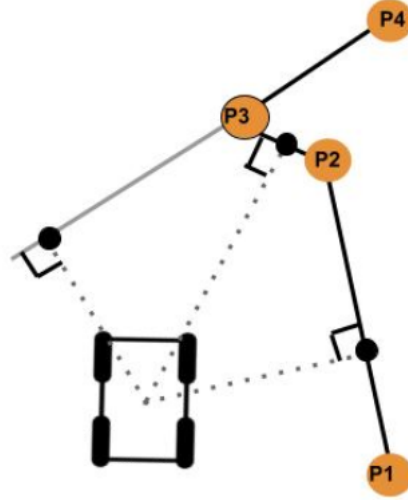


Figure 4: Determining the closest trajectory segment. The orange points outline the start and end points of each trajectory segment. The dotted gray lines show a perpendicular line drawn from the center of the car to each trajectory segment. Trajectory segments were extended as needed, shown by the solid gray lines. The intersections between the trajectory lines and the perpendicular lines are marked with black points.

can be determined.

$$|X - C| = LA$$

$$|P1 + t * V - C| = LA$$

$$|P1 + t * V - C|^2 = LA^2$$

$$(P1 + t * V - C) * (P1 + t * V - C) = LA^2$$

$$t^2(V * V) + 2t(V * (P1 - C)) + (P1 * P1 + C * C - 2P1 * C - LA^2) = 0$$

This leads to a quadratic equation with values for the coefficients of t.

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$a = V * V$$

$$b = 2(V * (P1 - C))$$

$$c = P1 * P1 + C * C - 2P1 * C - LA^2$$

First the discriminant, $b^2 - 4ac$, is calculated. If this value is negative, there is no real intersection between the look ahead circle and the trajectory segment. If the value is not negative, the solutions t_1 and t_2 can be determined.

$$t_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$t_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

If multiple intersections were found for the same segment, then the intersection with a higher t value was chosen, as it is further down the trajectory. Likewise, if multiple segments had intersections, then the t value for the intersection of the segment that was further down the trajectory and closer to the end point was used. Once the t value was determined, the initial equation for any point on a line segment, $P1 + t(P2 - P1)$, was used to calculate the location of the chosen intersection point.

2.1.2 Planning the Final Race Path (Devin)

A very important part of making sure our car completed the final race as fast as possible and with no collisions was tuning our planned trajectory. Using the trajectory builder utility provided by the staff in Lab 6, we created three iterations of paths for our car to follow using the pure pursuit controller.

When we designed our path for the first time, we kept in mind that we wanted to have long and straight segments. Our car performed a lot better with these types of segments in Lab 6, and we figured this would allow our car to reach higher maximum speeds with less oscillations. The initial path design can be seen in figure 5.

Our main issue with this design was that the 90° angles between segments were often causing our car to overshoot turns. This is why we chose to implement racing lines in our next iteration. When race car drivers take turns, they move to the outside lane and take the turn very close to its apex as depicted in Figure 6. This allows the driver to take the turn quickly and avoid collisions. Our implementation of these lines on the second iteration of the path can be viewed in Figure 7.

Our robot's performance on this path was better, but we noticed that it was still overshooting a couple of turns when we tried to increase the speed. The robot's speed around these turns was greatly impacting the fastest time we could get around the track, as the car was already often accelerating as much as it could on the straightaways. This led us to designing a third track with turns that were consistently overshoot implemented as two point turns. With this new implementation, the path segments on problematic turns were much closer to the inside of the track, which ended up correcting a lot of the overshooting we were noticing. Our final race path can be viewed in Figure 8.

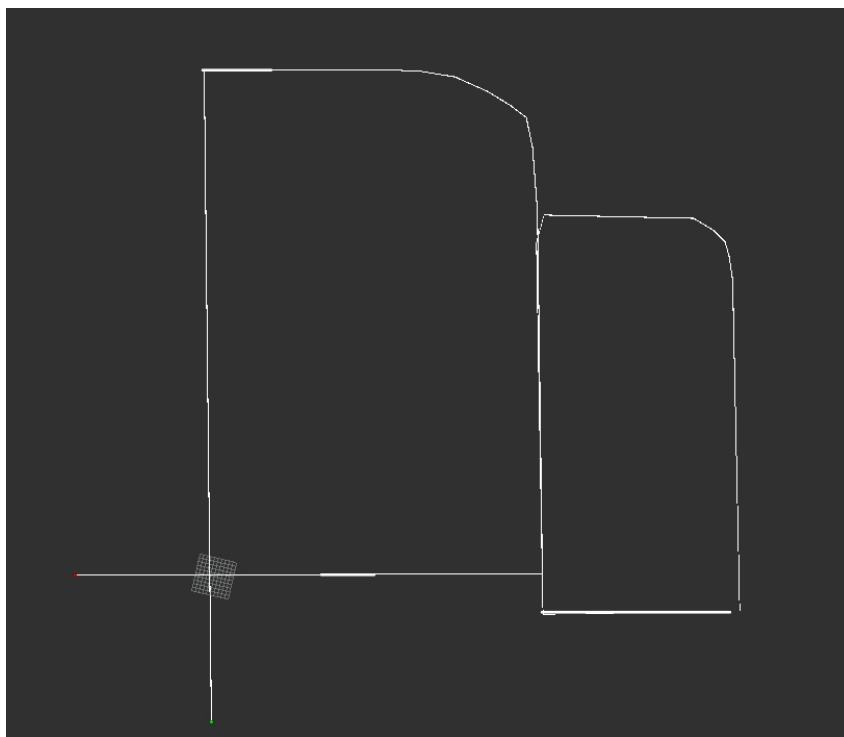


Figure 5: Our initial implementation of the race path

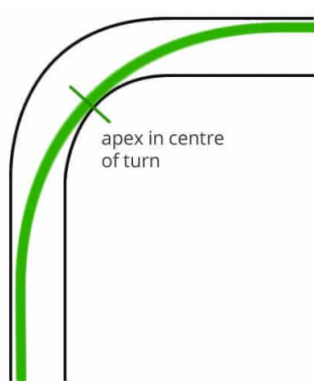


Figure 6: Example racing line, with apex marked

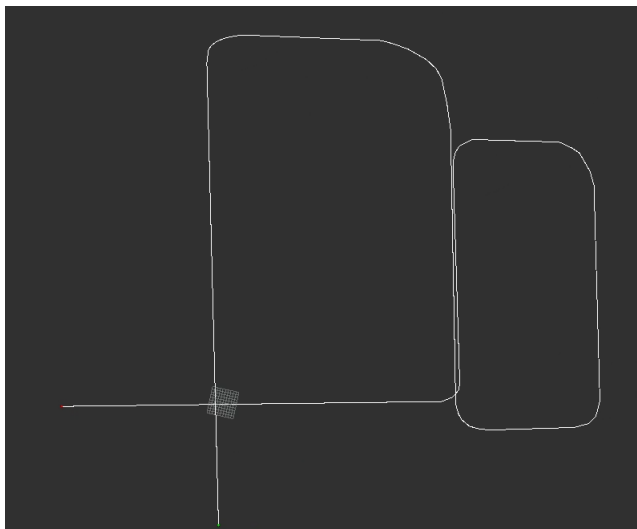


Figure 7: Our second iteration of the race path

2.1.3 Tuning the Controller: Determining the Look Ahead Distance, Steering Angle, and Velocity (Devin)

The process of determining the steering angle for the robot is almost identical to the process we used in Lab 6. The reference point we determine and car position we initially receive are in the map frame. If we transform the reference point to the car's frame, the geometry and math for determining the steering angle δ becomes much easier. The rotation matrix we used for such a transformation is listed below for reference.

$$\begin{bmatrix} x_{rp}^r \\ y_{rp}^r \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_{rp}^m - x_r^m \\ y_{rp}^m - y_r^m \end{bmatrix}$$

Using this point (x_{rp}^r, y_{rp}^r) , our look ahead distance d , the car's wheelbase L , and the geometries depicted in figures 9 and 10 we can determine the steering angle δ . The main update we made to this δ from Lab 6 was dividing it by some factor Cd to dampen oscillations, as we did for obstacle avoidance.

$$\theta = \arctan\left(\frac{y_{rp}^r}{x_{rp}^r}\right)$$

$$R = \frac{d}{2 \sin \theta}$$

$$\delta = \arctan\left(\frac{L}{R}\right)$$

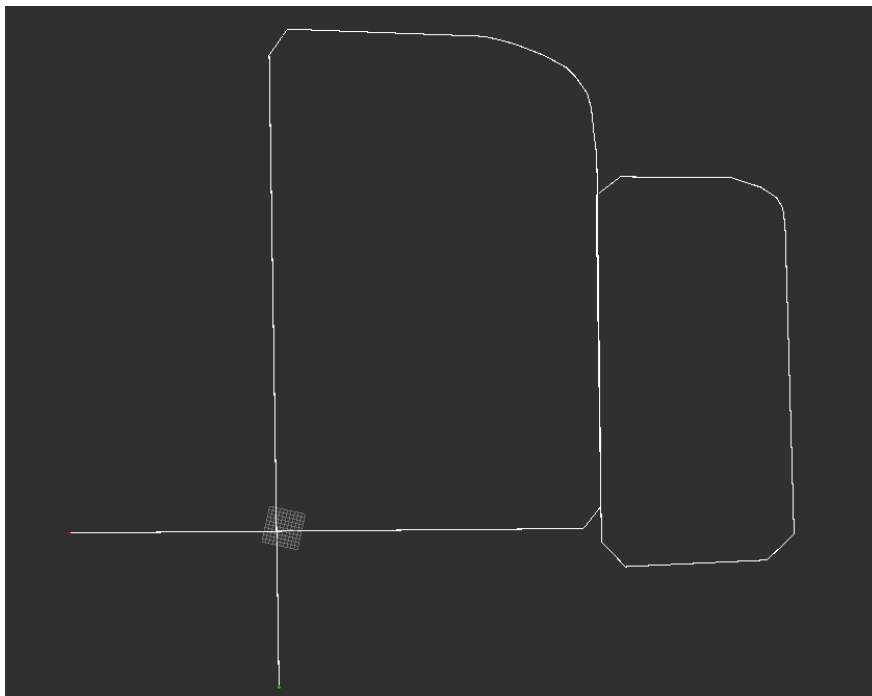


Figure 8: Our final iteration of the race path

Determining the reference point for the car as it moved along the path was a difficult task. If we made the point too close to the car, we noticed oscillations increase on straightaways. If we made the point too far away from the car, we noticed that the car would often not converge to our path quickly enough to avoid collisions on turns. We tried to opt for a solution that would allow us to customize these reference points to the unique needs of different segments of the path.

In order to complete the race as fast as possible, we knew that we wanted to maximize speeds on straightaways. This would require the car driving with as few oscillations as possible, so for long straight segments we initially set the reference point to be the end of the segment. However, as we approach the end of these segments we need to switch over to a more adaptive method of finding a goal point, as the car needs to change its steering angle quickly for the turn. Thus, we return to our circle intersection method of finding goal points as discussed in section 2.1.1. Figure 11 shows a model of our method for determining reference points.

We also implemented a variable look ahead distance control for when the car returns to using the circle intersection method. The car starts off using a look ahead distance equal to the distance at which it stopped using the end of

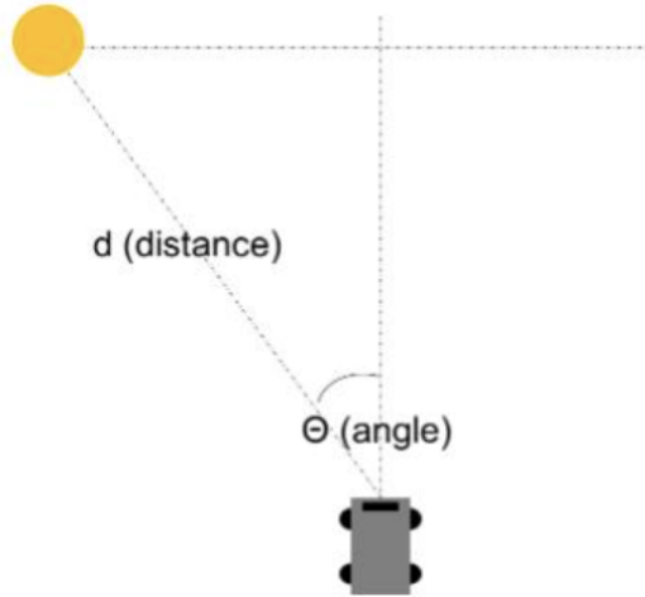


Figure 9: Reference point in the car's frame of reference

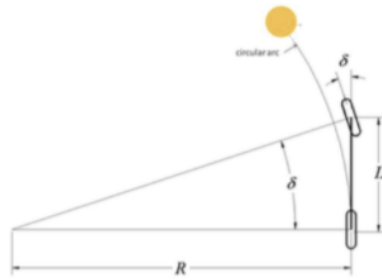


Figure 10: Steering angle δ needed to drive along the circle of radius R that intersects the reference point

the segment as a reference point. As the car approaches the turn, we decrease the look ahead distance to a minimum value for accuracy purposes. The nature of this control is depicted in Figure 12.

Our final task was to decide on a velocity for the car. As mentioned earlier in

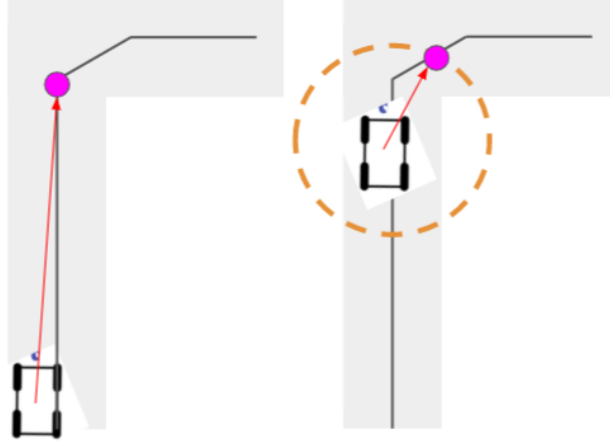


Figure 11: Model depicting how we determine the reference point on straight-aways versus turns

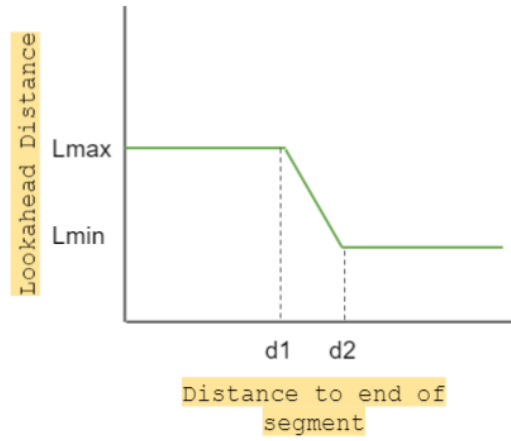


Figure 12: Graph showing the look ahead distance varying with distance away from the end of the path. d_1 and d_2 represent the distance at which we start and stop decreasing the look ahead distance, respectively.

this section, we really wanted to maximize our speed on straightaways, because we knew there would be a limit to how fast we were able to take turns without crashing. For our velocity control then we generally decided to increase the

velocity to a maximum value on straightaways, and decrease the velocity just before turns. On areas of the track that were particularly difficult for the car to pass through without collisions, we capped the speed at a lower value than the maximum of 30 m/s that we set for the track. A model of the velocity control is depicted in Figure 13. Metrics of our speed throughout the course of the track can be viewed in our Experimental Evaluation section.

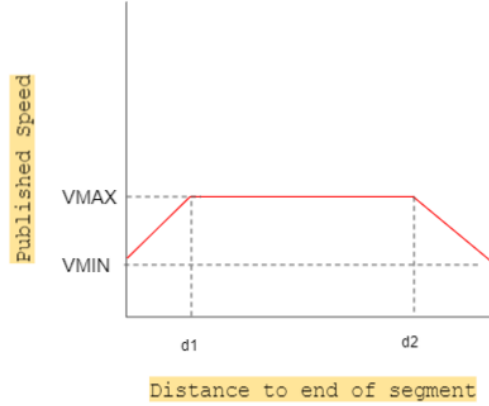


Figure 13: Graph showing the published speed varying with distance away from the end of the path. $d1$ and $d2$ represent the distance at which we stop increasing and start decreasing the speed, respectively

2.2 Fast Obstacle Avoidance

2.2.1 Algorithm (Tong)

In order address the fast obstacle avoidance challenge, we implemented a waypoint planning system. The overarching control loop looks something like this: drive towards the waypoint using dampened pure pursuit (i.e., regular pure pursuit with the output angle scaled down by some factor) until the car is within a certain distance from the waypoint, then plan the next waypoint and repeat. The waypoint planning algorithm is the most interesting part about our system. On a high level, it first proposes a set of waypoints, then uses the input lidar to determine which of those waypoints are viable, and finally selects the “most optimal” waypoint. The exact details can be found in our team’s codebase, but there are several high level ideas that we will elaborate on in this report.

When the system first proposes a set of waypoints, it considers a discretized set of points which are a certain lookahead distance away from the car, limited to a certain range of angles with respect to the car’s front axis. We limit this

range of angles in order to prevent the car from taking sharp turns that it cannot safely execute at high speeds. Then, in order to determine which of these waypoints (if any) are viable waypoints to drive towards, we check if there is a free lane to the point.

The idea of free lanes requires some elaboration. Essentially, we check that, when we consider a fixed width rectangular region starting at the car and ending at the waypoint, we check whether we detect any obstacles in this region. This can be done using some simple geometry, as shown in Figure 14.

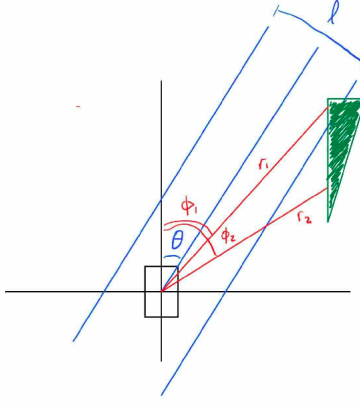


Figure 14: Diagram showing an example of calculations done to determine whether there is an obstacle in a particular lane using LiDAR data.

In this figure, θ is the angle of the lane, ϕ_1 and ϕ_2 are the angles at which we are calculating occupancy, r_1 and r_2 are the LiDAR measurements at those angles, and l is the lane width. In order to calculate whether there is an obstruction at ϕ , we compare whether the corresponding r is longer or shorter than a reference distance, where the reference distance is the distance we would expect to measure at ϕ . We can calculate this reference distance to be $\frac{l}{2 \sin(|\phi - \theta|)}$, which we compare to r ; if r is less than this reference distance, this indicates that there is an obstruction on the lane at angle ϕ ; if not, then there is no obstruction. In Figure 14, we can see that the lane shown has an obstruction at ϕ_1 , but no obstruction at ϕ_2 . Thus, the lane at θ is not a free lane, because there is an obstruction detected.

After the valid waypoints are determined, the algorithm uses chooses the valid waypoint that is closest to the goal point as the destination waypoint. This heuristic optimizes the distance the car travels in getting to the goal point, allowing for efficient navigation to the goal point.

In order to optimize our approach for speed on the obstacle course, we added several risky, course-dependant elements. The first element was choosing the

goal point far away from the finish line in the direction of the course, essentially allowing the car to pick waypoints closest to the finish line instead of closest to a single point on the finish line. The second element was to increase the speed after the car got within a distance of the finish line, as we observed that obstacles were much sparser in the second half of the course than the first half. These two adjustments allowed us to make the trade off decreasing the safety (increasing the probability of collision) with increasing the speed, a gamble which ultimately paid off in the final obstacle course.

2.2.2 Parameter Tuning (Adi)

The above controller resulted in multiple tun-able parameters for our system. We will start by listing these parameters before going into explaining their effects on the system and how they were tuned.

- $V_c \rightarrow$ the commanded velocity of the vehicle in the drive command
- $\alpha \rightarrow$ the LiDAR look-ahead angle
- $d \rightarrow$ the look-ahead distance
- $l \rightarrow$ the lane-width
- $R \rightarrow$ the re-plan distance
- $C_d \rightarrow$ the angle damping coefficient

These parameters are shown in the figure below but intuitively their meaning can be inferred from the description of how the controller works above. More formally, the commanded velocity V_c is the velocity published to the drive command over ROS, that is fairly straight-forward. During the controller's operation, the system searches for open "lanes" as to ensure when the car selects a path through the obstacles the car may be able to fit through this space, we define the width of this lane l to be always wider than the car's width. The α angle is the LiDAR angle through which the controller actually searches for lanes which we will describe in more detail later. The look-ahead distance d is essentially how "long" of a clear path should there be before the controller accepts the lane. And the re-plan distance R is defined as what distance from the next way-point should the car be before attempting to re-plan.

The final parameter to discuss is the C_d damping coefficient. Essentially the controller plans a way-point and sends it to the pure pursuit controller that was defined in previous labs and used in the final race. We found that this controller is very reactive especially when dynamically re-planning points and this causes large oscillations in the vehicle motion at high speeds. Essentially, we take the output steering angle from the pure-pursuit controller and divide it by a damping coefficient C_d and set that as the steering angle.

$$\theta_{set} = \theta_{ctrl}/C_d$$

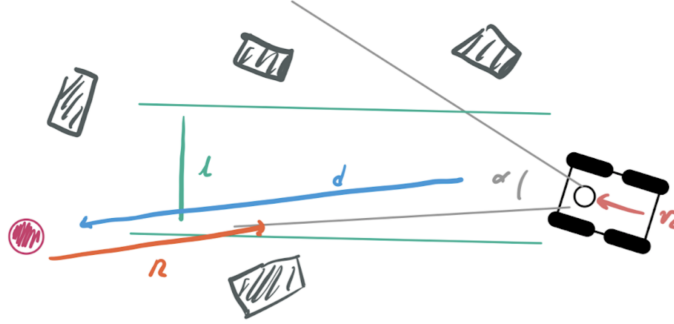


Figure 15: Visual description of the parameters to tune.

$$V_{set} = V_c$$

After playing with the parameters through many runs we settled on the following turning strategy. This strategy also explains why we had a parameter α that limits the scan angle to a certain look-ahead slice.

- $V_c \rightarrow$ set this as high as possible before the system goes unstable or the consistency (as defined in a later section) is undesirable.
- α → the alpha value allows us to only choose lanes within a certain LiDAR slice in-front of the vehicle. This allows us to ensure the next way-point chosen was not off to the side of the car effectively preventing the car from making sharp turns which is key at high speeds. This was a simple way to account for vehicle dynamics. We made this value fairly small.
- $d \rightarrow$ since we were travelling at high speeds, we made the look-ahead distance fairly large to allow enough time for the car to move without hitting an obstacle before re-planning.
- $R \rightarrow$ we made the re-plan distance large (i.e. we re-planned quickly) as re-planning quickly allowed the car to react to the changing environment quicker and allowed us to increase the speed even more.
- $l \rightarrow$ we set the lane width to just over the width of the vehicle. We chose a fairly narrow width as this allowed the car to slip in-between smaller obstacles and weave through tighter gaps but still prevented the car from side-swiping obstacles.
- $C_d \rightarrow$ the damping was tuned by setting a large damping coefficient and reducing the value until the oscillations became a problem and then slowly raising until C_d fixed the oscillations. The reason for this is damping is important so the oscillations don't cause unnecessary collisions but too

much damping means the car cannot turn as quickly to avoid obstacles either.

The final values for the system are available in the *obstacle_avoidance.py* file of the code-base. All these parameters are specific to the V_c command and are re-tuned for each V_c as well as the number of obstacles on the track. The parameters in the code-base now were tuned for 10 obstacles with a V_c of 19m/s.

3 Experimental Evaluation (Collaborative)

3.1 Overview of Design Goals

To develop specific evaluation metrics for any system, it's always important to start by reviewing the design goals. In both reviewing design goals and developing evaluation metrics we will separate our discussion based on the Final Race and Fast Obstacle Avoidance.

3.1.1 Final Race Design Goals

The main goal of the Final Race challenge is to complete a lap of Windridge City as fast as possible. The recommended time by the TA's was defined to be around or less than 45 seconds. This overall goal came with a few specific design goals we will list below.

- Define an optimal path for the car to take around the Windridge City track in order to maximize the speed of the vehicle.
- Develop a velocity control strategy that maximizes the vehicle's overall average velocity while maintaining stability around the turns.
- Reduce vehicle oscillations so the car does not crash into obstacles or drive a longer distance than it should (reducing the overall lap time).
- Precisely tune the Pure Pursuit Controller to follow the prescribed path.

3.1.2 Fast Obstacle Avoidance Design Goals

The main goal of fast obstacle avoidance is to drive through a random number of obstacles at randomly determine positions as fast as possible. The itemized goals for this are below.

- Develop an algorithm to accurately detect the positions of obstacles and open space using only the LiDAR sensor.
- Precisely determine which direction to drive based on the above information while driving towards a desired finish line.

- Account for vehicle dynamics and prevent the vehicle from selecting paths that cause it to turn sharper than possible or pass through spaces that are narrower than the vehicle.
- Tune the vehicle controller to an accurate speed and to quickly set the steering angles to turn to the desired positions.
- Ensure the vehicle oscillates only a minimum amount to ensure it does not hit obstacles while attempting to traverse the planned path.

3.2 Evaluation Metrics & Test Results (Collaborative)

The below sections describe our evaluations metrics and the results gathered for both the Final Race and Fast Obstacle Avoidance in both VDI and on the day of the final competition.

3.2.1 Final Race

In order to evaluate the performance of our solution to the Final Race, we collected graphs of both the distance error of our car from the planned path as well as the speed of the car as compared to the number of the path segment it was on.

The plot of the distance error (measured in meters away from the planned path) is shown in Figure 16. We managed to never stray more than about 3 meters from our planned path. A lot of the higher areas where our car strays from the path and our error increases is during turning. This was often intentional, as we found it beneficial to sometimes overshoot our diagonal path planned at turns to preserve momentum and complete the turns at a higher speed. Another thing to note is that we initially saw quite a lot of straight away oscillations in our implementation. These were dampened by dividing the steering angle determined by pure pursuit control by a factor greater than 2 (depending on which segment we were on this may vary and be larger).

We wanted a way to measure the speeds that the car was actually achieving as opposed to the speeds we were publishing as part of the drive command. We chose to plot the linear x velocity of the car as a metric for this, as the car's X axis is always pointing directly ahead of the car. The plots in Figure 17 show the velocity of the car and the segment number it is closest to as it completes the final race path.

These plots helped us verify that the velocity control was working as we expected. For example, we notice on the long straightaway of the first segment, segment 0, the car accelerates to a top speed of about 25 m/s before braking quickly for the turn represented by segment 1. This trend of accelerating on longer segments can be seen throughout the rest of the race. Another interesting feature of our velocity control that can be seen in the plot is the capping of speeds on segments that were particularly difficult for the car to make it through without collisions. This can be seen in the figure when the car reaches the numerous short segments from 3-11. Around these segments, the speed was

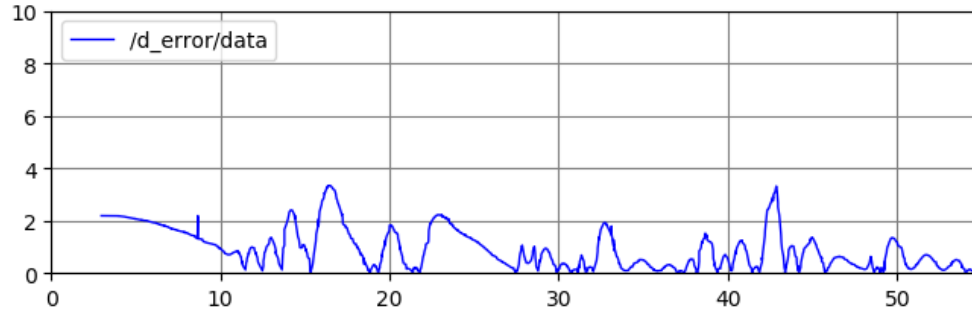


Figure 16: A plot of the distance error (measured in meters, the distance between the TESSE car and the planned path) over time. A video of the final car performance on our local machines can be found here: <https://drive.google.com/file/d/1W5P2dd9CdH0xZsHFGYJ8pdXAAAd1Osk5k/view?usp=sharing>.

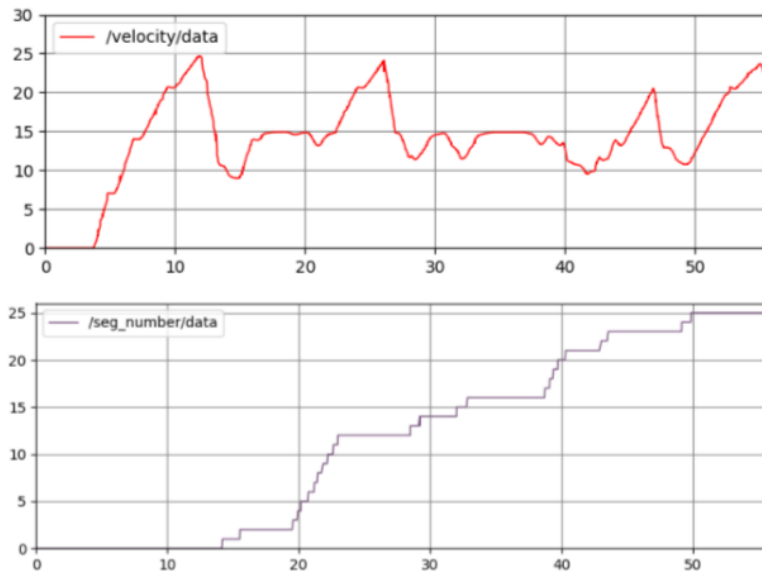


Figure 17: Plots of the speed of the car and the number of the path segment it is closest to versus time

capped at 15 m/s. The various locations where speed is capped also are useful to visualize because they help us determine parts of the path that can be further

optimized to achieve an overall faster final race time.

3.2.2 Fast Obstacle Avoidance

For fast obstacle avoidance, our team focused on two main evaluation metrics that we felt were more important than all-else. These were the metrics of average speed (or in VDI the commanded speed), and consistency.

The speed of the vehicle was evaluated by looking at the velocity parameter we set in the above sections (exactly the commanded velocity). While we know the actual average speed of the vehicle in this course will be less than the commanded velocity, since our controller commanded a static velocity for most of the course (before accelerating across the finish line) we can accurately assume the average velocity will increase with increasing commanded velocity assuming the car makes it through the course.

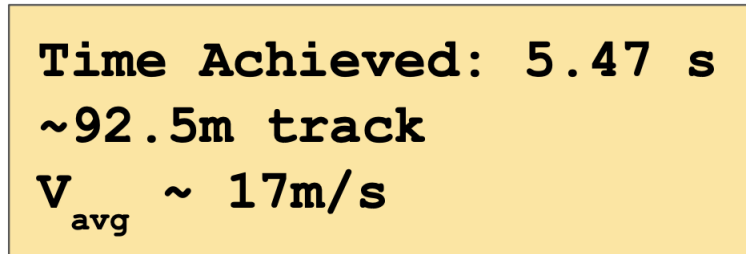
Consistency was equally important and we defined the consistency metric as follows. Where C_i is the consistency of controller i , N_i is the number of successful runs, and F_i is the number of failed runs for a specified controller.

$$C_i = N_i / (N_i + F_i)$$

A failed run was defined as a run where there were greater than zero collisions with any object. A collision with an object was defined as when the collision detector published a collision to the "\tesse\collisions" topic.

The strategy we took to tune the controller was to start with a low-speed stable controller and calculate the consistency while tuning the controller. We tuned the controller until it was "consistent" which is defined as a $C_i \geq 0.7$ then we would increase the speed and re-tune. When the final challenge details were released, we re-tuned the controller one last time using the final number of obstacles of 10, before the details were released we assumed a worst-case-scenario of 19 obstacles.

These evaluation metrics seem very simple but for our obstacle avoidance controller they were more than sufficient to accurately describe the controller's performance.



Time Achieved: 5.47 s
~92.5m track
V_{avg} ~ 17m/s

Figure 18: Final fast obstacle avoidance performance on the TA machine.

The above figure shows the final performance the system was available to achieve on the TA machines on race-day. The vehicle made it through the course in 5.47 seconds with zero collisions. On race day out of three races we had zero failed attempts which is a consistency of 1 (the highest possible). However, three attempts is not enough to determine an accurate consistency. On the VDI machine we ran 25 test runs of the final obstacle avoidance controller tuned for 10 obstacles and out of 25 runs the system was successful 18 times which is a consistency of 0.72 (fairly good).

We also found it important to visualize the path the car took as it gave us more insights into what was failing and why it may have been failing as well as which use cases the car can handle. We discussed some of this in previous sections but a quick example of this is in the diagram below. The below image shows our system can handle complex organizations of obstacles at high speeds.

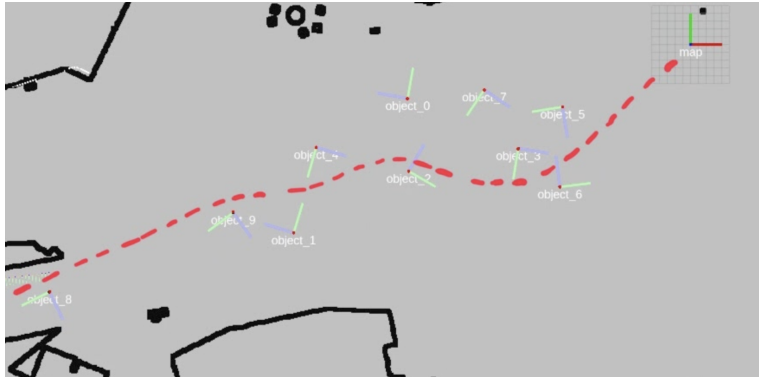


Figure 19: Obstacle avoidance threading thorough complex narrow paths.

3.2.3 Overall VDI Implementation vs. Final Race on TA Machines

To briefly touch on any performance drops between race day and running on the powerful VDI cluster, we saw no noticeable performance drop on the obstacle avoidance between the TA machine and VDI. On the final race on the other hand, the car was roughly 2 seconds faster on the VDI as compared to the TA machines. We are attributing this to less processing power and the additional graphics requirement from running tests with the graphical output. However, we will use VDI as the accepted performance abilities as it is most accurate to the controller's capabilities.

4 Conclusion (Emma and Adi)

Overall, we were successful and able to meet our goals for both final challenges.

We successfully completed a lap around Windridge city in under about 49 seconds with no collisions. Our solution was also pretty reliable, as we tested it over 5 times in a row consistently and continuously saw times under 50 seconds with no collisions. This challenge was completed by using the ground truth TESSE localization and pure pursuit to follow a planned path. The path was self planned and underwent numerous iterations for optimization. We determined that two point turns tended to work the best, as they allowed us to take advantage of the car's momentum and increase turn velocity. Our solution was also further optimized using segment-dependent variable look ahead distances and velocities that were determined by visual inspection and logging performance metrics. On straightaways, the look ahead distance was much larger to minimize oscillations and was decreased during turns. Our velocity controller increased speed as the car started a segment, reaching a maximum in the middle of the segment, and reduced velocity towards segment ends. We found that capping the velocity on problematic turns that often resulted in collisions was successful and prevented further collisions. We believe that we could optimize our solution further by increasing the minimum velocity and tuning appropriately.

We successfully completed the fast obstacle avoidance track. We did this with no collisions in just 5.47 seconds, averaging a speed of 17 m/s. This challenge was completed by using TESSE LiDAR to detect obstacles and path planning around them. We found that for high speed obstacle avoidance, complicated dynamic path planning may be needed to achieve good performance. Throughout designing our final solution, it was important to choose available parameters correctly to accurately tune the controller for the desired performance. Our obstacle-avoidance algorithm is very expensive to run. We believe that there is future work that could be done to make lighter versions of our algorithm.

5 Lessons Learned

5.1 Adi

In this lab I learned a lot about combining many individual aspects of a system to achieve a complex goal. I also learned about the importance of not oversimplifying a problem especially in relation to our obstacle avoidance controller. Our initial attempts were too simplistic to be reliable (simply turning away was not enough). But by observing where the system failed and what cause it to fail we were able to develop a better solution. That's the final thing I learned about, how to closely observe problems in a solution to make the system better.

Thank you so much for an amazing semester in RSS! I've learned a lot and loved the class so much!

5.2 Devin

In this lab I learned the value in having a variable look ahead distance as part of our pure pursuit controller. In Lab 6 we only used a constant look ahead distance, and this led to our car being unable to adapt to different parts of the track well. With our varying look ahead distance we were able to target problematic areas of the path and tune those look ahead distances accordingly. I also learned that planning talking points for briefings are especially important when they are longer, as it can be easy to get caught up in all of the things you want to mention.

5.3 Emma

I learned that creativity in technical implementations is often rewarded and simple changes may make a huge difference. Just because pure pursuit spits out a value for steering angle doesn't mean that you need to use it in this exact form. Simply dividing the pure pursuit steering angle to reduce the value improved our implementation significantly and significantly dampened oscillations (in a way that tuning other parameters wouldn't be able to). For this lab, it made sense to split up team members between the two final challenges. I learned that although we may be working on different things, it's still important to all come together and check in. This ensures that you get fresh perspectives. Someone may have a great suggestion for something you've been struggling on that you wouldn't hear otherwise.

5.4 Tong

It was really nice to see our team come together for this final challenge; we were able to divide up the work between the two challenges better than in any of the previous labs, and even though we all worked really hard to see our project succeed, I don't think us was tasked with too much. It was also really nice to see that we were at the point where we didn't really need to explicitly communicate who was doing what, as we all understood what everyone was working on, and didn't need to coordinate things with calls as much as we did in the previous labs. I also was reminded that idea generation is important and that it doesn't take someone familiar with the technical minutiae to generate a good idea. In terms of the technical work that was done, I enjoyed setting up infrastructure and enabling my teammates to save time during development; I will try to do this in future teams as well.