Final Technical Report for Team Star

CS 3892 – Autonomous Vehicles
Siddharth Shah, Raj Chopra, Teyon Herring, Aditya Shrey

April 26, 2024

1 Introduction

Our immersion project focused on developing an autonomous taxi system within a simulated miniature city environment. We programmed a virtual taxi to navigate this environment autonomously, relying solely on raw sensor measurements (e.g., GPS, IMU) and the bounding box output of object detection algorithms. The core challenge involved achieving robust perception and localization, allowing the taxi to understand its surroundings and precisely determine its position within the map. This perception data then fueled dynamic routing and motion planning. By leveraging this information, the taxi could navigate a designated sequence of pick-up and drop-off locations while adhering to a predefined set of traffic regulations (lane markings, stop signs, etc). We competed against other teams in a live event, and our success hinged on our ability to effectively combine localization, routing, motion planning, and perception to achieve safe and efficient autonomous navigation. Following our demo at the immersion showcase, we identified areas for improvement including resolving edge conditions with localization and fully-integrating perception.

2 Localization

Localization represented the core component of our autonomous vehicle technical stack by enabling the car to have an understanding of the state of the vehicle. In the vehicle simulation, we received noisy GPS and IMU measurements. Through our implementation of a particle filter, the state of the vehicle, represented by position, orientation, velocity, and angular velocity, could be determined.

For our localization pipeline, we began by initializing all 500 particles to be randomly distributed around the first measurements that were received. All the particles were initially given the same weight. We would then either receive a measurement from our GPS or our IMU. In the event that we received a GPS measurement, we would update all our particles with our simple bicycle dynamics model. In our dynamics model, we add some noise to our model. This ensures that we do not converge easily to a single point in our particle filtering process, which was a problem we initially encountered. We would then assess the likelihood of these measurements given by our GPS. The likelihood for our GPS measurements took latitude, longitude, and heading measurements into account. All were assumed to follow a Gaussian distribution around the GPS measurement. In the event we received an IMU measurement, we would follow a similar step except we would assess the likelihood of the measurements given by our IMU. For the IMU likelihoods, we took the velocity and angular velocity into consideration. Once again, we assumed the likelihoods would follow a Gaussian distribution around the IMU measurements. After assessing the new weights for each particle, we then normalized the weights and resampled. The process of receiving measurements, updating the weights of the particles, and resampling would continue for the duration of the vehicle simulation.

Average errors for state variables (position, yaw, velocity, and angular velocity) converged when the vehicle was still and diverged when the vehicle was moving. This dynamic can be seen in *Table 1*.

Motion Type	Position error after 10 seconds	Position error after 20 seconds	Position error after 30 seconds
Standing Still	0.4717	0.4281	0.4198
Moving	1.2119	2.2924	3.050

Table 1: Magnitude of Position error over time

3 Routing

For the routing module, we broke down the implementation into the following tasks: (1) identify the current road segment given position measurements, (2) map an efficient path from the current road segment to the target loading zone, and (3) construct a polyline representation connecting the midpoints of each road segment along the determined path.

To identify our starting road segment, we iterated through every road segment in the map and generated rectangular borders based on the corresponding leftmost and rightmost lane boundaries of each road segment. We then implemented a simple conditional check to confirm if the x and y position of the ego vehicle (either ground truth measurements or localization estimates) fell within the particular rectangular boundaries and returned the respective road segment id if the condition was satisfied. This implementation works for all straight and loading zone road segments, however, has room for error when the initialization of the vehicle occurs on a turn – in which case there is a 50 percent possibility of identifying the road segment turning in the opposite direction. This case was first witnessed by us during the multi-vehicle live simulation and could be handled by adding an additional check for curved road segments using the radius and approximated center of curvature.

Once identifying our starting road segment, we employed the A* heuristic search algorithm to map an efficient path of connected road segments to our target loading zone. This was achieved by importing the A*StarSearch* Julia library and using the '*map[segment_id].children*' member variable to create a simple graph representation of the map in which each node [segment_id] had children pointing to all valid road segments [next_segment_ids] that follow. Our lightweight astar search call employed the default goal-check heuristic to operate like a slightly improved breadth-first search. Once we determined the optimal path, represented by an array of road segment ids, we created a polyline representation of our path with the MidPath struct in pathfinding.jl. The polyline object contained two lists of points, which the corresponding i's of both lists containing the points midP and midQ for the i^{th} line midP → midQ. This implementation is universal for all road segments in the path to the target loading zone, deriving the midP and midQ points from the respective lane boundaries. Both turns and loading zones are handled through the creation of partial diagonal segments, specifically the diagonal segments for loading zones turns are 65% of their original length.

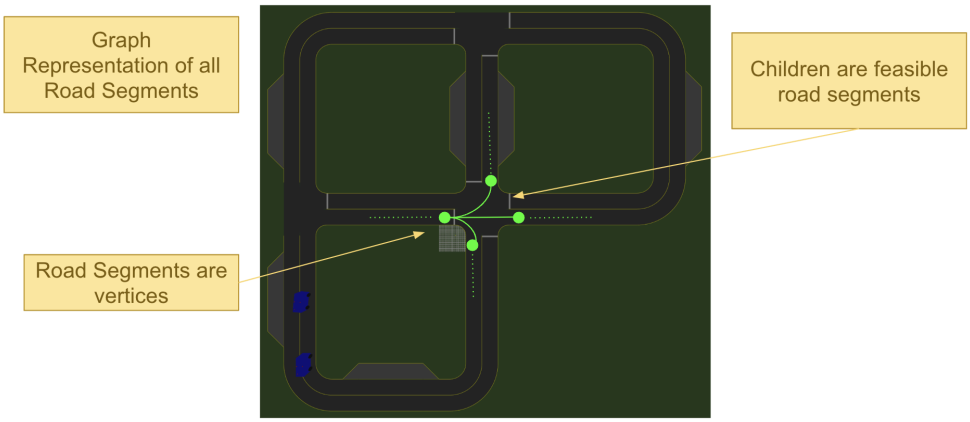

Figure 1. City map with potential actions visualized if approaching a 4-way intersection from the left.

4 Motion Planning

The idea of motion planning involved making decisions on vehicle controls based on given constraints: lane boundaries, speed limits per individual lane, stop signs, and other vehicles. After breaking down the constraints, we then went to consider what actually made these constraints. Starting with the lane boundaries, the simple straight lanes were made of two points (a, b) and had a curvature of 0. With this, we could create a direction vector and a normal vector serving as a bounding area where the vehicle can be. Doing this with both lanes surrounding the vehicle created a bounding lane for our vehicle. When considering turning lanes, these had curvatures greater than 0 so we had to think about how we would need to adjust our speed and/or steering angle to handle each one accordingly. In addition our bounding constraint would be two circles based on the lane's curvature and using that concept we created boundaries for the turning lanes. Next was the speed limits, in this case, we simply had to configure our vehicle velocity upper bound to be the road's speed limit, though ideally we wouldn't be driving that fast. For stop signs, once we arrive on the road segment with the stop sign, we can detect it and therefore stop when we reach a certain distance away from the stop sign. Finally to handle other vehicles, initially we would adjust our speed based on the car in front of us by knowing their speed to avoid collision while also avoiding completely stopping on the road unless we were forced to. However due to time constraints, a simpler method was created to recognize and handle other vehicles where we would assume if the vehicle is in front of us within a certain range, we will stop. Further details of this will be explained in section 5.

With the information above, we originally used an IPOPT solver however in our final decision making implementation we used a stanley controller. How the stanley controller works is that it calculates the front car axle's signed distance from the expected path as a distance error. Next it calculates the angle the path compared to its wheel axis and calls that the theta error. With those two error calculations, using the function shown in figure X, the car calculates the angle needed to correct itself at its current velocity.

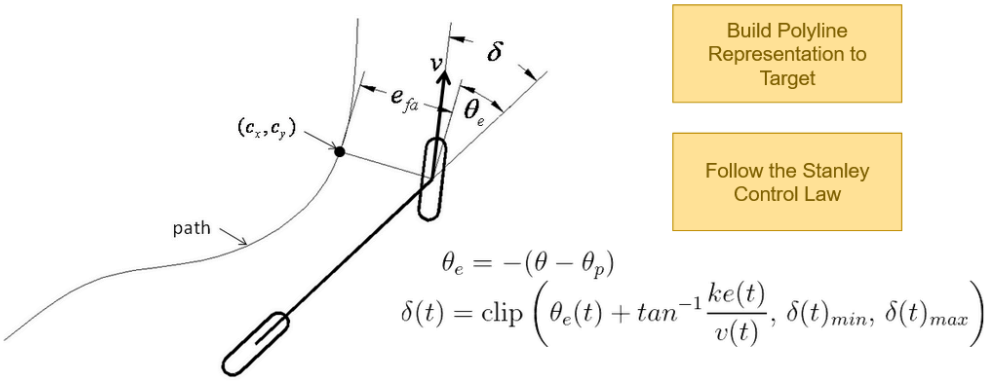

Figure 2. Diagram of a Stanley Controller.

After switching to the stanley controller, we needed a midpoint between the lanes to provide our vehicle a track to follow which was provided by routing. Then we created a function for calculating signed distance and theta error. With that complete, we had witnessed a failure rate of less than 3% and above 95% replicability rate. In the beginning, there were lots of slow oscillations between the road path so we had done fine-tuning on the bounding angles for straight lanes and turning lanes to adjust the oscillation to cause less than a 0.1 theta error. With these adjustments we were able to smoothly arrive at multiple pick-up/drop-zone across the map while following all the given constraints of the road.

Onto the discussion of failures, the initial plan was to use the IPOPT (Interior Point OPTimizer) solver. However, the initial visual tests resulted in the car giving bad drive controls during turns causing the car to always fall out of lane boundaries. Initially, we were doing these calculations based on the center point of the vehicle so instead we attempted to improve the results by using the four corners of the car since these would be the major points necessary to keep the car within the lane boundaries. After implementing this however, the vehicle seemed to stop turning during tests. It would drive straight through turns sent by routing and eventually drive off the course. What we noticed through testing was that the IPOPT solver seemed to freeze until the turning sequence was over, implying that the optimization problem may have been too difficult to solve quickly. As a result, we reduced the points to two points, those being the top left and bottom right corners of the vehicle. This also caused the same freeze when it came to the turning segment. So we fell back to the algorithm with only the center point solver. Then after doing further testing, we calculated that it was both very inconsistent with its drive control results, having a replicability rate of less than 30%. In addition, it had above a 95% failure rate when it came to turns. With these statistics, we had transitioned to the Stanley Controller which produced much better results overall.

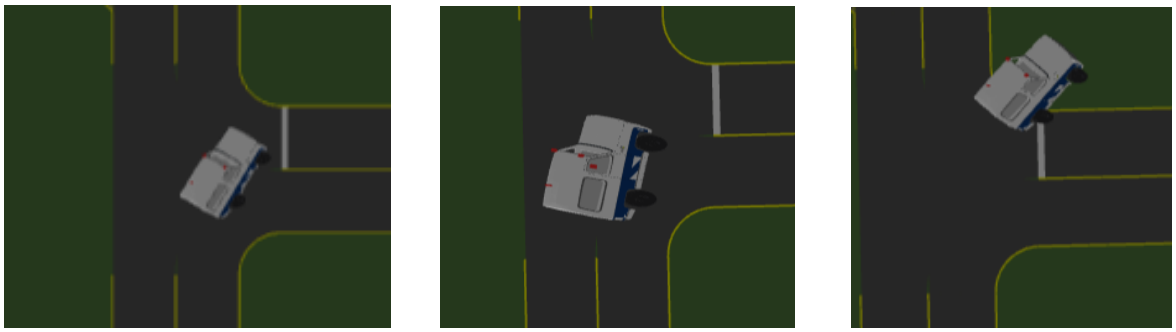

Figure 3. Failed instances of the IPOPT solver.

5 Perception

Initially, we planned on implementing an Extended Kalman Filter for our perception module. As we received bounding boxes from our measurements, we would use the EKF and determine the state of the surrounding vehicles. Given the time constraints and issues with geometric transformations, a decision to pivot to a simpler method.

The new method that our autonomous vehicle employs is the “Box = Stop" method. The idea with the “Box = Stop" method is rather simple. We take our bounding box and take the intersection with a subset of the image. If the intersection area is greater than some threshold, we stop the vehicle. The method works in most cases of our simulation environment as we are primarily concerned with not hitting other vehicles as opposed to making the most logical decision. In other words, stopping more times than necessary was an acceptable decision. For the parameters of determining what subset of the image we would consider for each of the two cameras, we had to fine tune based on experimenting with the simulation. In the real world setting, this method is rather nonsensical as there may be times when we receive a large bounding box, perhaps a larger vehicle, and stopping would not be appropriate. However, for the purposes of this project, we want a simple implementation of perception that could work.

Our perception implementation can be considered as a more brute force approach to the problem of identifying other vehicles in the city environment. Given our bounding boxes, a boolean would be returned, indicating whether our vehicle should continue moving or stop. As such, the testing phases were primarily concerned with determining the parameters for the area to intersect with the bounding boxes. We ultimately determined that for both vehicles, an area greater than 10,000 pixels *Figure* 4 illustrates a basic implementation of our perception.

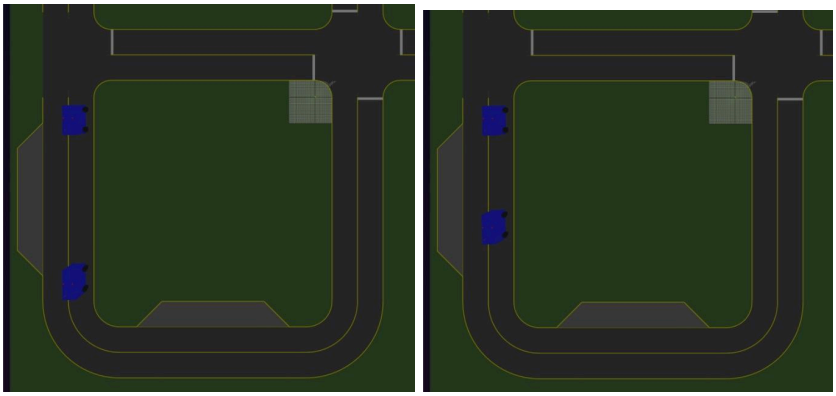

Figure 4. On the left, the ego vehicle (displayed at the bottom) has the stop boolean as false, indicating the vehicle to continue moving. On the right, the ego vehicle is now close enough to the other vehicle and the boolean is set to false.

6 Live Performance

During the immersion showcase, our team went head-to-head with our peers to distinguish the best performing autonomous taxi control system. The competition was broken down into two phases: (1) individual taxi navigation in the simulated city map and (2) simultaneous navigation of all taxis within the same simulated environment. The aim of the first phase was to identify the baseline functionality of the autonomous taxi, specifically regarding the routing and motion planning necessary to map an efficient path to the target loading zone, drive while obeying appropriate traffic regulations, and repeat the process towards the next goal segment after completing the first stop. The second phase, while retaining all of these aspects, added the additional challenge of perceiving any nearby vehicles and modifying your planned trajectory accordingly. Finally, both segments could be tested with either exact ground truth measurements or localization estimates of the ego vehicle's position.

Our autonomous taxi control system successfully executed during both phases of the live competition when provided with ground truth measurements for the position of the vehicle. This, along with a prior uninterrupted run for over 30 minutes, validated our implementation of routing and decision making. Our performance, however, was not completely seamless. During the 1st phase of the demo, our vehicle made abrupt stops at stop signs, swerved harshly into the target loading zone, and could have been faster at certain moments. Another team was able to resolve some of these concerns by implementing a velocity ramp up function, progressively slowing down as opposed to stopping abruptly, and driving slower when entering and exiting the loading zones. These slight limitations, however, are minor when compared to those that followed.

During phase 2 of the competition, our taxi successfully navigated the cluttered city environment without any major collisions. While tracking our ego vehicle, we noticed one particular instance of it stopping when about to run into a turning taxi at a four way intersection. This single instance, however, does not reveal the glaring limitations of our current perception implementation. By only relying on front-camera bounding boxes to stop, we fail to address the dynamic motion-planning and trajectory modifications necessary for autonomous vehicles in the real world. Furthermore, we cannot account for the behavior of other vehicles to our sides and behind us. 360-degree dynamic

perception and trajectory planning for other vehicles in our surroundings could enable our ego vehicle to dynamically react to a much larger array of situations and avoid collisions.

Finally, our autonomous taxi failed to accurately distinguish its relative position in the city map when relying solely on localization code. While the localization part of the competition was short, we witnessed another team successfully implement position estimates from sensor data into their routing and motion planning. While discussing, we learned the other team had implemented an EKF which may be an avenue for future exploration. Moreover, being more deliberate with our implementation and testing of our filtering methods would hopefully allow us to overcome our difficulties.

7 Team Contributions

Teyon and Siddharth worked primarily on the routing and motion planning modules. Raj and Aditya worked primarily on the localization and perception modules. Each member contributed 25% of the code created and all members contributed equally.

8 Recorded Demonstration

Here is a link to the a video demonstration of the routing and motion planning modules making use of ground truth: https://youtu.be/OBskx_j5HS4