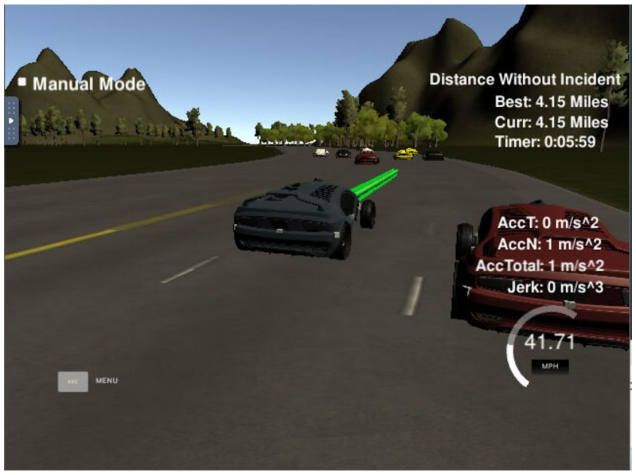
Project: Highway Driving

The goal of the project was to safely navigate a 3-lane highway in simulation, without colliding with other cars or exceeding the speed limit of 50 mph. Lane changes should only be made when they are both safe and help the car progress through traffic. Acceleration should remain under 10 m/s^2, and jerk under 50 m/s^3. The main task of the project can be split into 2 parts: behavior planning, and trajectory generation.

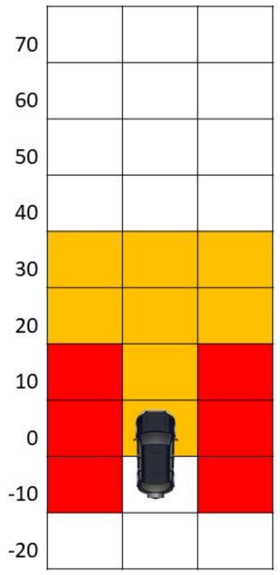


The simulator running. The car's trajectory is shown in green.

Behavior Planning

The output of my behavior planning step was the determination of a target lane (1, 2, or 3) and speed (<50 mph). Breaking this down into 2 steps, behavior planning involved polling for and determining nearby object (i.e. Sensor Fusion), and deciding on a move.

Using the sensor fusion data, I decided whether a car would be present in any of the 3 lanes within a target area at the end of this move. I also kept track of the closest car to the ego car in each of the lanes: its velocity, and its relative distance to my travel. Below is a graphic defining positions where vehicles are found in a lane (colored blocks), and their distance to the ego car at the end of the move (in meters).



Once I defined the presence of other cars, I decided on a move and car speed. A move to another lane was only allowed if the following conditions are met:

- There is another lane in that direction I can move to (the move wouldn't pull me off the road or into opposing traffic)
- There is no car in that lane OR the car in that lane is moving faster than the car in my lane, and it is at least 20 m away from me

Red spaces in the above graphic indicate positions of neighboring cars which would make a left or right lane shift not allowed. The target speed is ideally that of either: the car in front of me, or the maximum speed allowed. However, to avoid sudden large accels or decels, I only incremented the velocity, rather than setting it to the nearby car's speed immediately.

Trajectory Generation

After determining a behavior as a goal lane and move speed, the trajectory generation step determines the actual X & Y values for the trajectory of the car. To do so, I:

1. Find 5 points that the car will visit as it travels down the road towards its goal position
2. Define the 5 points in relative time and extrapolate those points with a spline function
3. Send the simulator the x & y points for each time step of 0.02 seconds

These steps are described below.

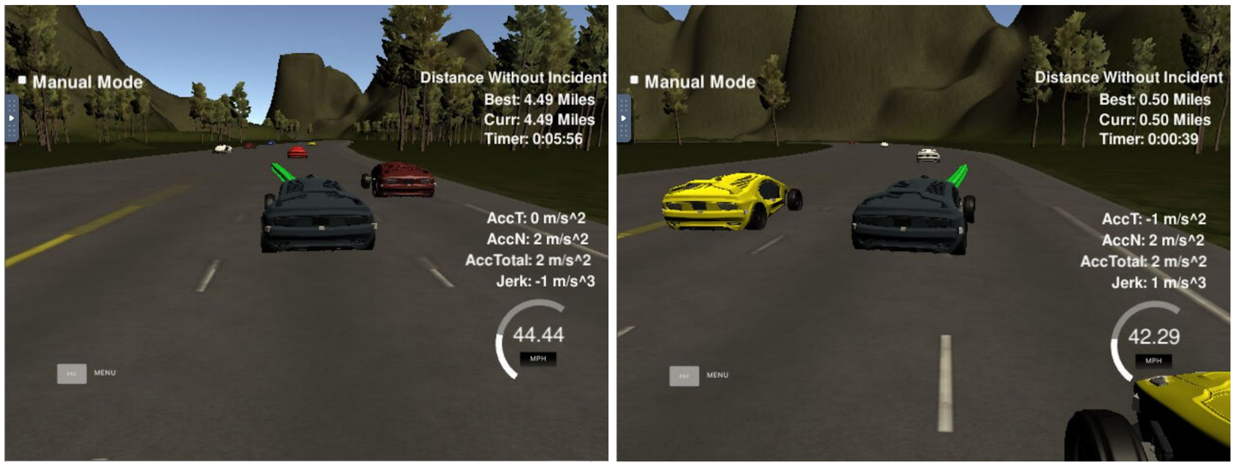
1. The 5 points used as car visit points consist of 2 previous points and 3 future points. The previous points are the last x & y points from the previous path, if there are unprocessed points. If there is no previous path, likely just at the start of the simulator, the previous points are the car's current x & y plus one point behind the car's current s value, in the

same lane. The 3 future x & y points I defined as 30, 60, and 90 meters down the road in the middle of the target lane.

2. The timing of the previous 2 points in straight-forward. The most recent point is time 0, while one step prior is -0.02 second, since each trajectory point is always 0.02 seconds apart. The timing of the next 3 points must be based upon the car's commanded velocity. I calculate the time stamp from the distance between the two points and the velocity at which the car should be travelling, based on: Time = Distance/ Speed. With all visit points defined, I calculated 2 splines: one for x vs. time, one for y vs. time.
3. Finally, I send the x & y points to the simulator. First, I fill the vectors with any remaining, un-processed points from the previous move. I always send the simulator 1 second, or 50 points, of data. Therefore, any open spots after the remaining points are added I fill with points from the splines calculated in step 2.

Outcome

The ego car is able to drive around the track without incident for 4.5+ miles. It had gone 7 minutes before I stopped it.



The ego car detects car to right+ahead moving slowly and initiates a left lane change [Left]; The ego car detects cars ahead+left and initiates a right lane change [Right].

Discussion

I dealt with quite a bit of trial and error in this project. Two of my most time-consuming problems were dealing with the trajectory at initial condition and accounting for the right behavior in all (in reality: most) of the situations.

Initially, I had used an initial condition of a prior point being one time step (0.02 seconds) directly behind the car. However, once I reached the final stages of minimizing acceleration spikes and reducing jerk, this singular point with small time differential was not sufficient. Also, with a start from 0 velocity, the first second does not show much change in position; The first 50 points calculated will be very close together. This also proved to be an issue with the spline function. With relatively constant values of either x or y that increases significantly at larger time values, the spline often curves in the opposite direction of what it should. I ended up using a reference point 30 meters behind the car on the track (car's "s" -30) as the previous point for my initial condition spline calculations. Still, on certain start-ups of the simulator, there will be an initial maximum acceleration violation, which I hypothesize is either due to lag, as changing multiple parameters did not resolve the issue.

My other issue was in deciding behaviors for generalized conditions. It was difficult to consider all the neighbor car conditions and then pick a behavior for each. I wanted to have consolidated code, while still making the best decision at any point in time. This proved more troublesome than I had initially planned for, as the possibilities for current conditions is truly infinite. I believe I did a fairly decent job, and at risk of spending too much time on perfecting this aspect, I will consider the current state of the project sufficient.

I also had utilized a single spline function with only x & y points until my primary remaining problem was an error from the spline function that x points were not increasing. Since the track is circular, sometimes x values decrease with time- the spline function did not like that. So, I switched to calculating 2 splines: both x & y vs. time. This made the timing for the initial condition especially difficult for me.

In the future, I would like to spend more time planning out behaviors and trajectories before actually jumping into coding. I tend to come up with an initial idea, which is generally correct, program it quickly, but then spend copious amounts of time making small alterations and observing the effects, rather than calculating for myself what those effects will be.