Scott Thomson - 3/7/18
SDC Term 3 - Path Planning Project
Model Documentation

Goal
The goal of this project is to successfully navigate a car down a highway with slower-moving traffic while maintaining the target velocity as closely as possible, all while avoiding collision and obeying all traffic laws.

Inputs
As input, the program takes the current position and velocity of the our car, as well as the positions and velocities of other nearby cars. It also receives a series of waypoints from the simulation that mark the center of the road.

Generating a Path
This part of the program takes a target lane, and generates a series of points outlining a path for the car to follow that will allow it to smoothly reach that lane (whether this is a nearby lane or the current lane). To do this in a way that minimizes jerk, the trajectory needs to be smooth.

First, we take the last two points on the path from the last cycle. Then we add three more points out down the road (which we can do easily by using s coordinates and converting them to XY using the provided functions). With these five points, we create a “spline”, which allows us to output a y value that follows that path for any given x value. By including two points from our previous path, this ensures that the path is smooth.

We then use this spline to add more points to the end of our future path. Because we know the target velocity, and we know that each point will be reached every 0.02 seconds, we can do some math to calculate exactly how far apart these points should be spaced to maintain this speed. We also have to convert these points back into a global reference frame before pushing them into the path vector.

There is a tradeoff in how many points our path should be. Too many points, and the car won’t be able to respond to sudden changes (such as a car entering into our lane). Too few points, and the car may move more jerkily as it has less time to respond to new commands or avoid obstacles. In practice I found 50 worked best.

Picking a Lane
My lane targeting system utilizes three main functions. The first “closest_cars”, takes the sensor fusion data, sorts the cars by lane, and identifies the car in each lane that is closest to our car. (since we don’t have to worry about cars that are past this car)

The second function looks at a specific adjacent lane, and determines whether there is space for our car to change into this lane. It does this by checking all cars in this lane, and using their

current speed to project where they will be one second into the future. If this position is within 7 meters of us, it is considered not clear.

Finally, our main function takes all this data and uses it to pick the best lane. I tried a number of functions, but the simplest worked the best:

The core of this function identifies which lane is the fastest. Speeds above our target velocity are ignored (since we'll never catch them), and cars that are more than 30 meters ahead are considered to simply be moving at our target velocity. (since we'll have space and time to respond to them)

In the event that multiple lanes all will allow us to move at our target velocity and one of them is our current lane, we simply stick with our current lane.

In most cases our car plans ahead and never needs to slow down, but in the event that it is unable to pass a slower moving car, or a car changes into our lane, we quickly lower our reference speed by subtracting 0.224 m/s during each clock cycle.

Future Work

My program is pretty good as is! (as I'm writing this, my car has driven 15 miles without incident)

One serious weakness is that the car only checks adjacent lanes when deciding whether a lane change is safe, which means it can get into a collision if another car is also trying to merge into the same lane that we are. In practice this happens pretty rarely in the simulator, but if I had more time it's what I would try to work on next. (although it's tricky, since you'd have to find a way to predict which cars may be about to change lanes).