Model Documentation

Path Planning

Vern Francisco

Methodology

This is the most challenging project so far in the program, and I struggled to put together all the concepts outlined. I ended up starting with the project video walk-thru to get comfortable with starting the project.

Main()

The changes I made to main() are starting on line 347. Until this point the code checks to see, from sensor fusion whether there is a car in front of the ego vehicle. If there is a flag of 'too close' is set. From here I call a function, that I developed called detect_barrier that takes in as input, sensor fusion data, the current lane, the current car position and time into the future. The return of this function is a value that determines if there is a car to the left, right, both sides or if there no cars in either the left or right lane.

Then, at line 350, I check to see if it's safe to do a lane change for at least 3 cycles of checking surrounding vehicles. I did this because I was finding that at times, the ego car would change lanes with just one detection of no surrounding cars and sometimes collide with neighboring cars. Adding this time delay greatly reduced those events.

From line 356, if the counter timer threshold is achieved, it means a lane change is possible and the logic determines the lane change based on current lane position. If the current lane is left, then the ego vehicle will change lanes to the right. If the current lane is right, the vehicle will change lanes left. If the vehicle is in the center lane, it checks to see if there is a vehicle to the left or right, then changes to the lane that is free. If neither lane is occupied, the left lane is chosen based on general 'rules of the road' that prefer passing on the left vs. the right. After the lane change is set the barrier counter is set back to zero to prepare for the next lane change.

After this the code reverts back to the project video walk-thru which regulates speed based on the 'too close' flag and uses the spline function to determine the jerk optimal path

Added function – detect_barrier()

Detect_barrier() inputs and outputs are as described above. This function cycles through the vehicles from sensor fusion. It uses the same methodology as the starter code to predict the s value of the sensor fusion vehicle. Then it checks a defined distance ahead and behind the ego vehicle's predicted position in the left and right lanes. If a vehicle is detected in the left or right lanes, a flag will be set for the detection.

Then depending on which lane, the vehicle detection flag is set to prevent changing lanes into the shoulder or on-coming traffic.

Finally, the status of vehicles is set based on the flags set for left and right detection. 0 for no cars on either side, 1 for at least one car on the left, 2 for at least one car on the right, and 3 for at least one car on either side of the ego vehicle.

Reflection

The ego vehicle can travel more than the required 5 miles before a collision or event but the path planner can be improved significantly.

For the path planner, I would like to improve the code by implementing the quantic polynomial solver and the behavior planner finite state machine. The reason I would like to implement this is that the polynomial solver can help to find a more suitable position in which to change lanes, whereas my simple implementation requires a fairly large gap to open up to safely execute a lane change. Also the current implementation cannot prepare for a lane change directly. A finite state machine can help to implement a speed control to help get into a tighter gap. Currently, there are some situations in which the ego vehicle can get stuck in a lane as gaps pass by.

The other improvement that can be made is to avoid vehicles changing lanes into the ego vehicle's lane at very close distances. In order to improve this, a prediction model should include d_dot of each vehicle to see if the predicted position of the sensed vehicle will place it in front of the ego vehicle.