Path Planning Project

Build a Path Planner for Highway Drive

The goal of this project is to build a path planner that creates smooth, safe trajectories for the car to follow. The highway track has other vehicles, all going different speeds, but approximately obeying the 50 MPH speed limit.

The car transmits its location, along with its sensor fusion data, which estimates the location of all the vehicles on the same side of the road.

The goals of this project are the following:

1. The car must not exceed the speed limit of 50mph on the highway
2. The car must not exceed a total acceleration of 10 m/s^2
3. The car must not exceed a total jerk of 10 m/s^3
4. The car must not have collisions with any of the cars while changing lanes or driving inside a lane
5. The car stays in its lane, except for the time between changing lanes.
6. If the car ahead is moving slow, the car shall be able to change lanes.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Compilation

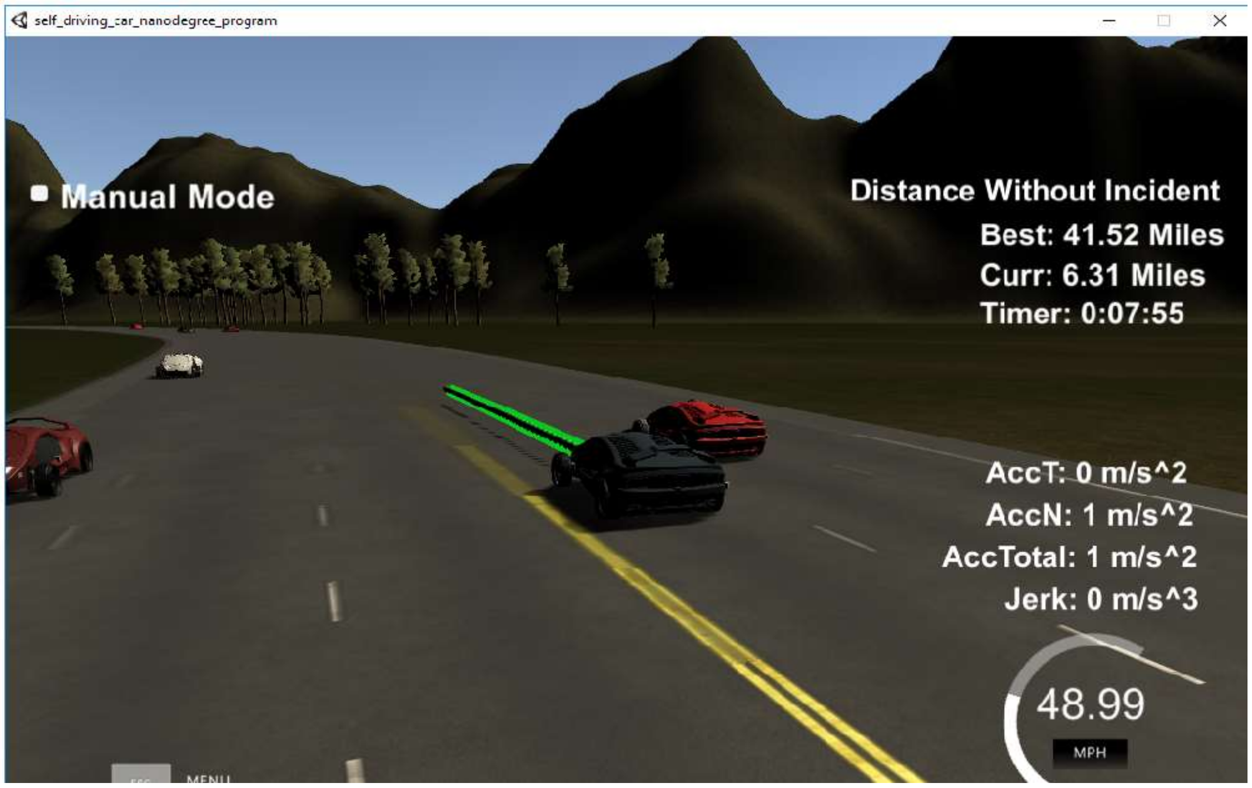
The code compiles correctly.
Code compiles without errors following the project instructions:

1. Make a build directory: `mkdir build && cd build`
2. Compile: `cmake .. && make`
3. Run it: `./path_planning`.

Valid Trajectories

1. The car is able to drive at least 4.32 miles without incident.

The car drove around the highway without incidents for a couple of laps. In the final test the car was driving on the highway for nearly 4 hours. The maximum distance travelled without incidents was 41.52 miles



2. The car drives according to the speed limit.

The car keeps a 49 mph speed if not hindered by traffic.

3. Max Acceleration and Jerk are not Exceeded.

Smooth trajectories are generated that take into consideration acceleration limits and jerk minimalization.

4. Car does not have collisions.

The car makes decisions to avoid collisions both while changing lanes and when hindered by slower vehicle in lane.

5. The car stays in its lane, except for the time between changing lanes.

Lanes are respected and lane changes are done smoothly and as swiftly as possible without validation acceleration and jerk requirements.

6. *The car is able to change lanes*

As explained above the car evaluates the situation and changes lanes if required to be able to travel with the desired or maximum possible speed.

Reflection

There is a reflection on how to generate paths.

It follows below.

Model Documentation

The simulator provides the following data to work with:

- Ego vehicle's localization data (map x, y coordinates, Frenet s, d coordinates, yaw, speed)
- Previous list of x & y points previously given to the simulator
- Previous path's end points in Frenet s & d coordinates
- Sensor fusion data of other vehicles id, position and speed

Prediction

The sensor fusion data coming from the simulator is used to check other vehicle's position and to try to estimate their future trajectories.

Behavior Planning

This is the logic I used for behavior planning.

The car starts in a "Keep Lane" state meaning after a gradual acceleration it maintains a speed of cca 49 mph in the starting (middle) lane. It is constantly monitoring the vehicles' position coming from sensor fusion data. (code lines 132-151) If it approaches another vehicle in its lane within 30ms it makes a decision what to do next. (code lines 156 - 186).

It first evaluates its current state and checks what states it can transition into. (line 158).

If it is in "Keep lane" it can transition to "Prepare to change left" and "Prepare to change right" states. /I used an FSM with these states to ensure smooth movements, so that the car will not start "swinging" between two lanes./

It also checks which lane it is in, so it does not try to change right in the rightmost lane. (lines 285-307 in helpers.h)

If the state machine and the position both allow a lane change then it checks if the desired lane to change is free to go to (lines 165 - 182). It also calculates a cost of the lane change.

Cost function

I used a very basic cost function to ensure maximum possible speed is preferred. That is the ratio of the actual speed over the target speed (49 mph). I subtracted it from one so that higher speeds result lower cost. I combined with a little offset for the keep lane state to slightly favor keeping the lane (so that the car does not change lane if it can go 0.1 mph faster in that lane). This was meant to be a very very basic safety check introduced in the evaluation -on the assumption that lane change is considered “riskier” in general than staying in the lane therefore lane change is performed when “it is worth it”.

The possibility and cost are calculated for all possible successor states and the state with the minimum cost is selected. (lines 196 – 210).

The trajectory is calculated for the selected state, only. (For trajectory generation please refer to the next point.)

Ensuring lane change

In contrast to the exercises in the classes in the simulator the lane change did not happen instantly, rather it needed some time to complete -during which a trajectory regeneration may cause problems, even different decision can be made so the car may end up “swinging” on the road or driving on the lane marks for some time instead of driving in either of the lanes.

To overcome this difficulty after a lane changing trajectory was generated the planner keeps sending the same trajectory (minus the already travelled points) to the simulator until at least 80% of the trajectory is travelled. (see the big if statement in line 121).

Trajectory generation

For trajectory generation I used the spline.h header offered spline as described in the walkthrough.

As it produces very smooth trajectories that also care for the jerk limitation, I did not modify anything on the implementation. (line 217)

Known issues, room for improvement

There are a few shortcomings of the present realization of the project that I did not have time to address (due to a relatively short time I have to finish the course).

A very basic way to improve the planner is a good refactor of the code. At the beginning I expected it to be a smaller scale problem therefore I decided to use a functional-like implementation but as it got quite complicated by the end it would have been clearer to use a class structure. (Hope, the code is still understandable with the descriptions, though.)

Another improvement needed is the state machine that should include the case of the car needing time to change lanes.

There are some further safety concerns to consider if it was a more realistic project: in several cases for simplicity I used constant distances which are set empirically. Although I set them so that they are safe for the “valid” speeds in a real environment (minimum) speed-dependent variables would be a better choice.

I also think that the prediction module could be improved further by keeping track of the “problematic” vehicles’ movement other than taking a snapshot and trying to predict from there (that the current implementation is doing).

And there sure are other issues I didn’t even think of… 😊