I completely followed udacity lectures and the last video on the assignment. I started coding as Aaron suggested in the video. I could understand that the entire process is divided into several steps. I started with some code to make the car move as suggested in the video. I started with path planning with the starter code provided in the class room. Basically we created points distant at 0.5 so that the car can move at a constant velocity. But the car went on a straight line and since the acceleration increased rapidly there was a jerk. Then I moved ahead as per the walkthrough video.

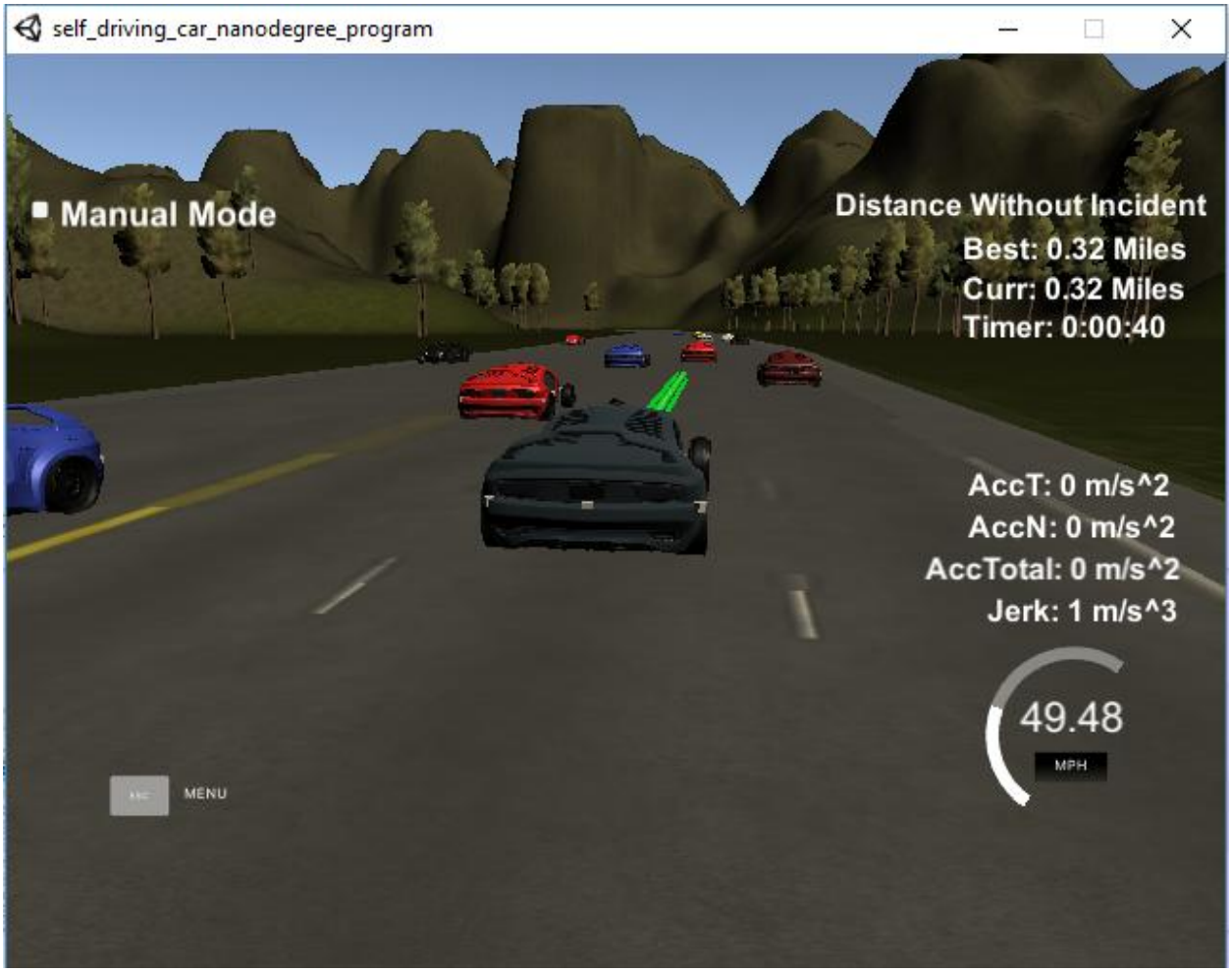
As we know, we need to use Frenet s,d coordinate system so that we can have the car move relative to the road and the curvature of the road will not create problem in calculation. There were provided routines to convert global coordinates to Frenet and vice versa. As suggested I used spline for high resolution way points. The code is like this.

```
tk::spline
s;
                        s.set_points(ptsx, ptsy);
                        // Output path points from previous path for continuity.
                           vector<double> next_x_vals;
                           vector<double> next_y_vals;
                        for ( int i = 0; i < prev_size; i++ ) {
                          next_x_vals.push_back(previous_path_x[i]);
                          next_y_vals.push_back(previous_path_y[i]);
                        }
```

Using the rough x, y points to fit the spline tool, which will create a smooth function. The distance between first x to last x is made up by interval times by number of intervals. The simulator returns instantaneous telemetry data for the vehicle, but it also returns the list of points from previously generated path. This is used to project the car's state into the future and a "planning state" is determined based on the difference between points at some prescribed number of points along the previous path.

The sensor fusion data received from the simulator in each iteration is parsed and trajectories for each of the other cars on the road are generated. These trajectories match the duration and interval of the ego car's trajectories generated for each available state and are used in conjunction with a set of cost functions to determine a best trajectory for the ego car.

Actually the goal of the project is to find the best trajectory . This is the most difficult project for me so far. I have gone through the videos several times to solve this and I also sought my mentors advice. More material on this will make me more confident. A snapshot of the car while drive is shown below.


self_driving_car_nanodegree_program
Manual Mode
Distance Without Incident
Best: 0.32 Miles
Curr: 0.32 Miles
Timer: 0:00:40
AccT: 0 m/s^2
AccN: 0 m/s^2
AccTotal: 0 m/s^2
Jerk: 1 m/s^3
49.48
MPH
MENU