Highway Driving Path Planning Project

The goals/steps of this project are the following:

- In this project, your goal is to safely navigate around a virtual highway with other traffic that is driving +-10 MPH of the 50 MPH speed limit. You will be provided the car's localization and sensor fusion data, there is also a sparse map list of waypoints around the highway. The car should try to go as close as possible to the 50 MPH speed limit, which means passing slower traffic when possible, note that other cars will try to change lanes too. The car should avoid hitting other cars at all cost as well as driving inside of the marked road lanes at all times unless going from one lane to another. The car should be able to make one complete loop around the 6946m highway. Since the car is trying to go 50 MPH, it should take a little over 5 minutes to complete 1 loop. Also, the car should not experience total acceleration over 10 m/s^2 and jerk that is greater than 10 m/s^3.
- The project should satisfy the project rubric.

Files Submitted & Code Quality

My project includes the following files:

- main.cpp containing the path planner implementation and the starter code that reads data from the simulator.
- spline.h which contains the library imported for trajectory generation. (Spine library - https://kluge.in-chemnitz.de/opensource/spline/)

Implementation approach

Prediction Step

- The prediction step is implemented in main.cpp from lines 115-163.
- For, the prediction step, the sensor fusion data from the simulator is used to identify and predict the location of all the vehicles.
- Using each vehicle's velocity, we try to predict its location into the future. We use, the longitudinal distance (s) of Frenet coordinates to identify the distance between the ego vehicle and the other vehicles.
- The lateral distance (d) of the Frenet coordinates is to identify the lane in which each of the vehicles is driving.
- If the ego vehicle is too close to the vehicle ahead of it, then we must reduce the speed and check it is safe to change over to other lanes.

- Based on the predictions, we set various flags like *too_close_same_lane*, *too_close_left_lane*, and *too_close_right_lane* to indicate if it is safe to maintain the same speed or to change lanes.

Behavior Planning Step

- The behavior planning step is implemented in main.cpp from lines 165-194.
- For, the behavior planning step, the flags set during the prediction step are used to plan the behavior for the vehicle.
- For, my implementation three states were chosen - keep lane, left lane change, right lane change.
- If the vehicle ahead is too close, then we must reduce our speed. Then we check if it is safe to move into the left or right lane to overtake the slow-moving vehicle in front of us.
- The speed of the vehicle is gradually increased/decreased to minimize the jerk.

Trajectory Generation Step

- The trajectory generation step is implemented in main.cpp from lines 198-320.
- First, the spline library was used to interpolate the waypoints. In the simulator, the waypoints provided are not evenly spaced. So, to get more points, which will result in less jerk, we are interpolating the points.
- We also need to transform the points to the local car co-ordinates, so that the first point in the path is at origin (0,0) and heading of 0 degrees.
- For the path planner, we use the left-over points from the previous path, to ensure a smooth transition. We then append, the new x and y points (in map coordinates) to generate our path.

Output

The screenshot below shows that the vehicle was successfully able to travel the whole track which is 4.32 miles without any incident. The vehicle was also able to change lanes to overtake slower vehicles on the road.

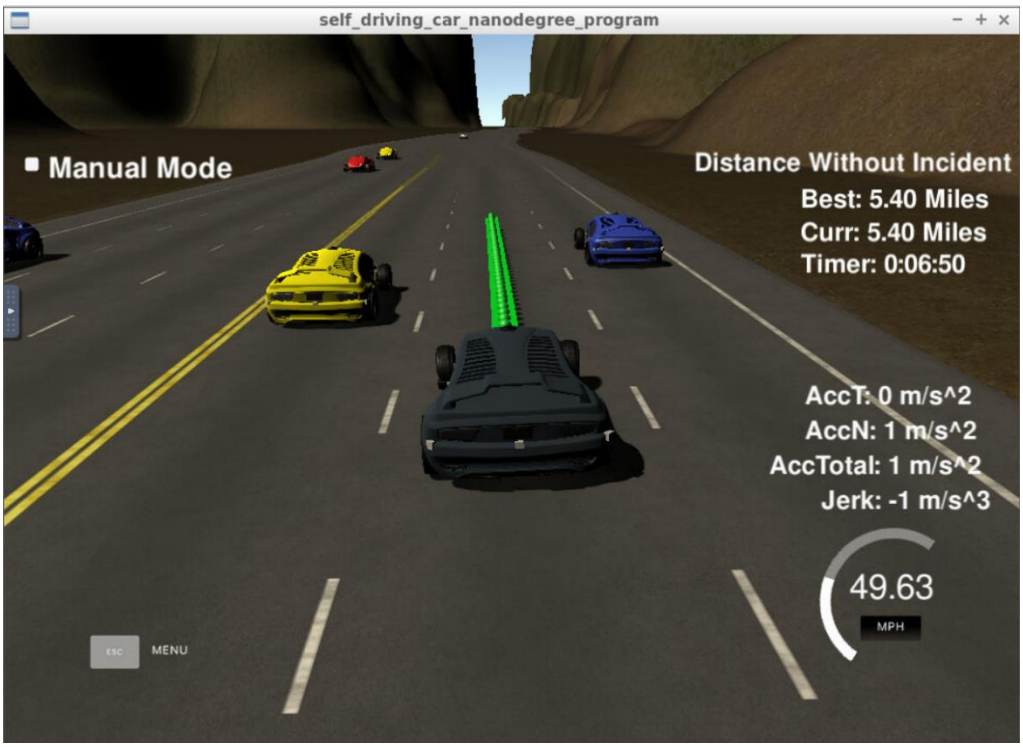

Fig 1. Ego Vehicle has successfully traveled 5.4 miles without incident

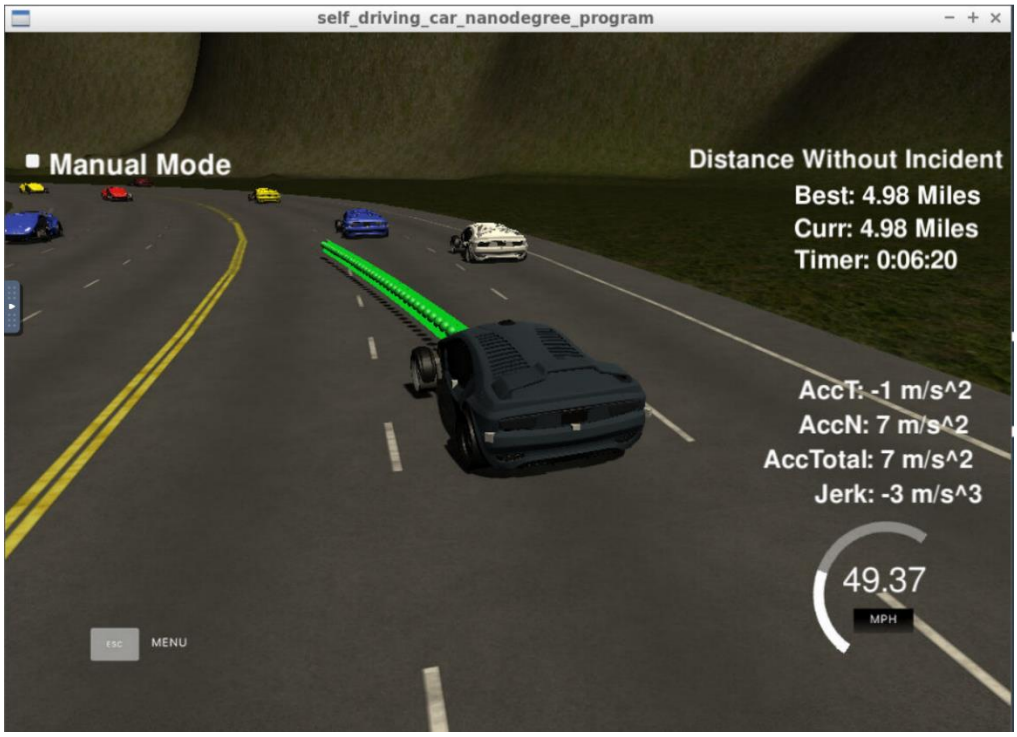

Fig 2. Ego Vehicle switching to the left lane

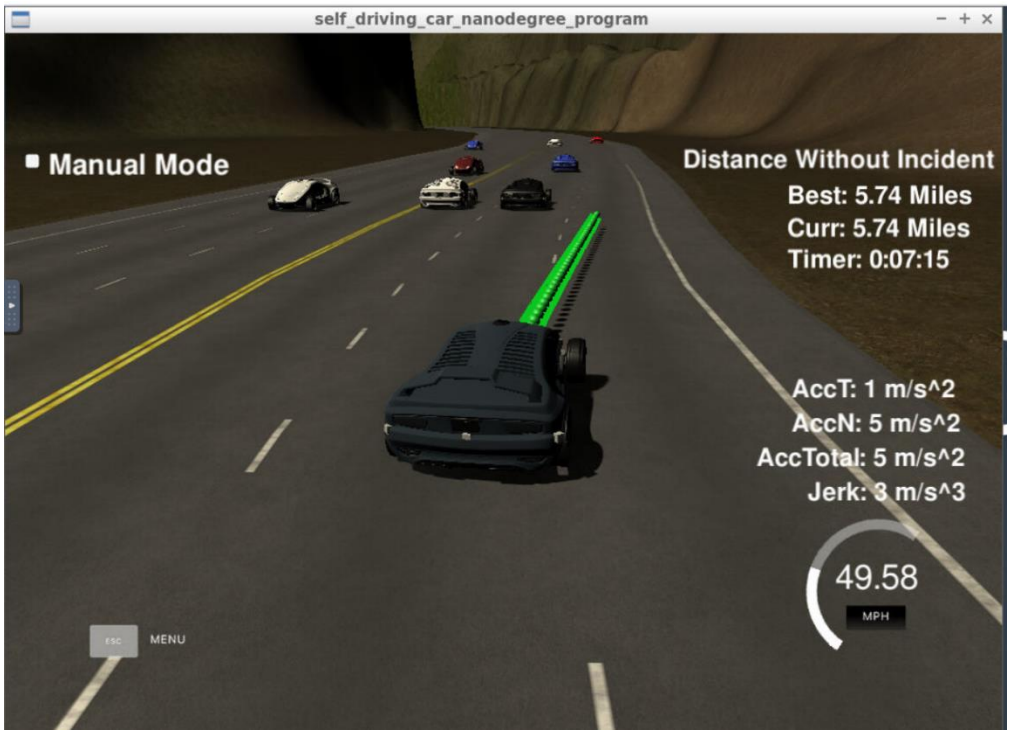

Fig 3. Ego Vehicle switching to the right lane