Introduction

The goal for this project is to design a path planner that is able to create smooth, safe paths for the car to follow along a 3 lane highway with traffic. The path planner is able to keep inside its lane, avoid hitting other cars, and pass slower moving traffic all by using localization, sensor fusion, and map data.

The project is graded according to these Rubric Points, which have been addressed in my project. A table of contents for the rubric point is below:

Rubric Point Table of Contents

i) Required Files **2**

ii) Quality of Code **2**

Is the code functional? 2

Is the code usable and readable? 2

iii) Model Architecture and Training Strategy **3**

1. Has an appropriate model architecture been employed for the task? 3

2. Has an attempt been made to reduce overfitting of the model? 3

3. Have the model parameters been tuned appropriately? 4

4. Is the training data chosen appropriately? 4

iv) Architecture and Training Documentation **4**

1. Is the solution design documented? 4

2. Is the model architecture documented? 6

3. Is the creation of the training dataset and training process documented? 9

v) Simulation **9**

1. Is the car able to navigate correctly on test data? 9

i) Compilation

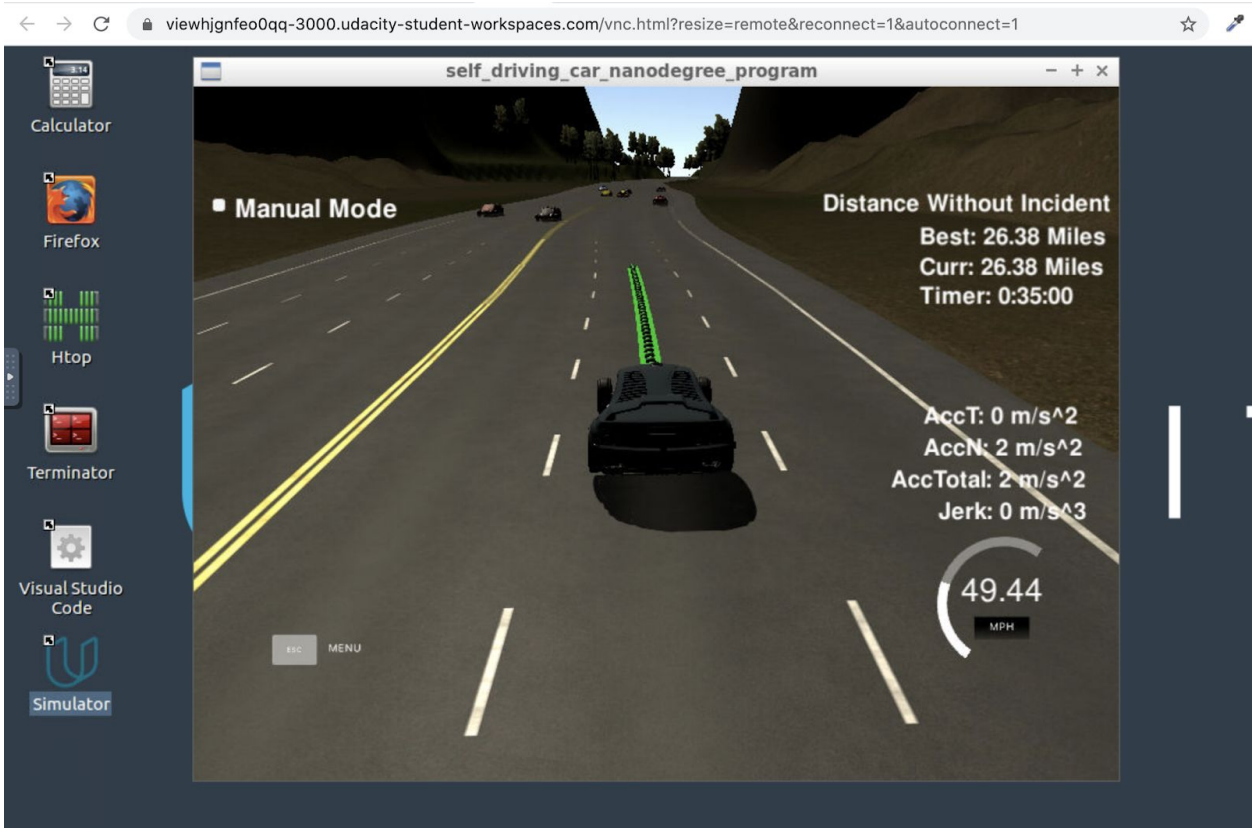
The code compiles correctly.

Code compiles without errors with cmake and make

ii) Valid Trajectories

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

The car consistently drives more than 15+ miles without an incident



The car drives according to the speed limit.

The target speed has been set for 49.5MPH. Trajectories with speeds above that carries a high penalty (1).

Max Acceleration and Jerk are not Exceeded.

Max acceleration has been set at 9.8 in either direction. Trajectories with accelerations above that carries a high penalty (1)

Car does not have collisions.

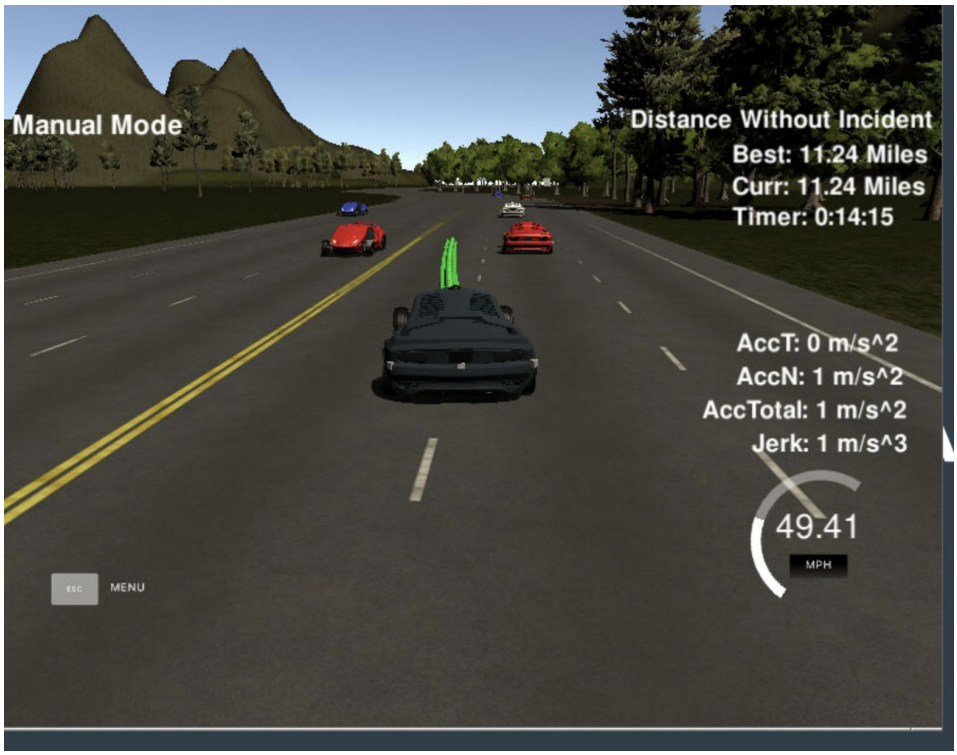
The car is able to consistently drive 15+ miles without a collision

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

The car drives in the current lane by default - using the provided control module. My additional code only provides instruction for changing lanes.

The car is able to change lanes

Lane change maneuver demonstrated in the frame below



iii) Reflection

There is a reflection on how to generate paths.

Udacity's simulator provides inputs to my algorithm such as the vehicle's location, velocity, yaw, speed, frenet coordinates and sensor fusion outputs. Using a spline function I fit a smooth path to 5 trajectory points. I used a pre-existing spline library. After fitting the smoothened path, the 5 trajectory points are fed back to the simulator to drive the ego vehicle.

Step 1: Traffic Prediction
Using data from the sensor fusion module and the simulator, I predict what the other vehicles are likely to do. The sensor fusion data provides information about cars near the ego and I

predict what they are likely to do. The behaviour planning module (below) takes these predictions and determines the optimal path for the ego vehicle.

Step 2: Behaviour Planning for Ego Vehicle
The behaviour planning module (line 120-166 in main.cpp) decision logic works by:

1. **Check ahead for slow vehicle in current lanee:** check 30m ahead of ego to see if there are any other cars ahead in the current lane
2. **Attempt lane change if a slow vehicle ahead in the current lane:** If there is a car 30m or less ahead in the current lane and it is moving slow, then initiate a lane change attempt.
3. **Check if we can safely change into the left lane:** If left lane is free (i.e., contains no vehicle within 30m ahead and 10m behind the car), then initiate a left lane change. We prefer a left lane change first because passing on the left is preferred in right-lane-drive countries.
4. **Check if we can safely change into the right lane:** If right lane is not available, then we do the same check for the right lane. We do this second as passing on the right is less preferred as left lane passing.
5. **If both lanes are occupied, then stay in current lane and slow to speed of car ahead:** We will continuously check whether the left and right lane becomes available at a future point.

Step 3: Trajectory Generation for Ego Vehicle
The Trajectory Generation module creates the target trajectory based on the speed of the ego vehicle, the speed of surrounding vehicles, the current lane position, the target lane position and prior trajectory points.

To smoothen the trajectory I add two prior points to the target trajectory - if prior points are available. If there are no prior points, then I calculate the prior points from the current yaw and ego vehicle's coordinates. If two prior points are available, then we add the next 3 points 30, 60 and 90 meters ahead in the trajectory. These points are transformed into the vehicle's frame of reference.