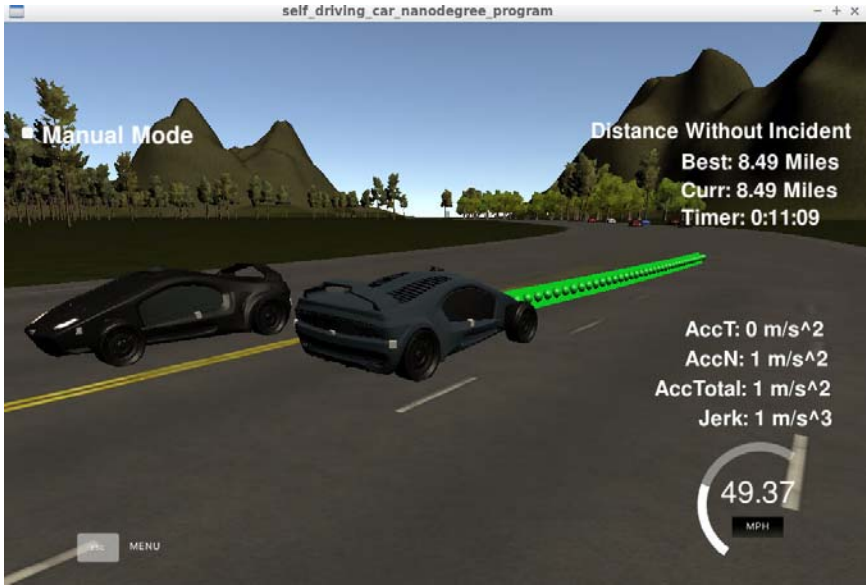
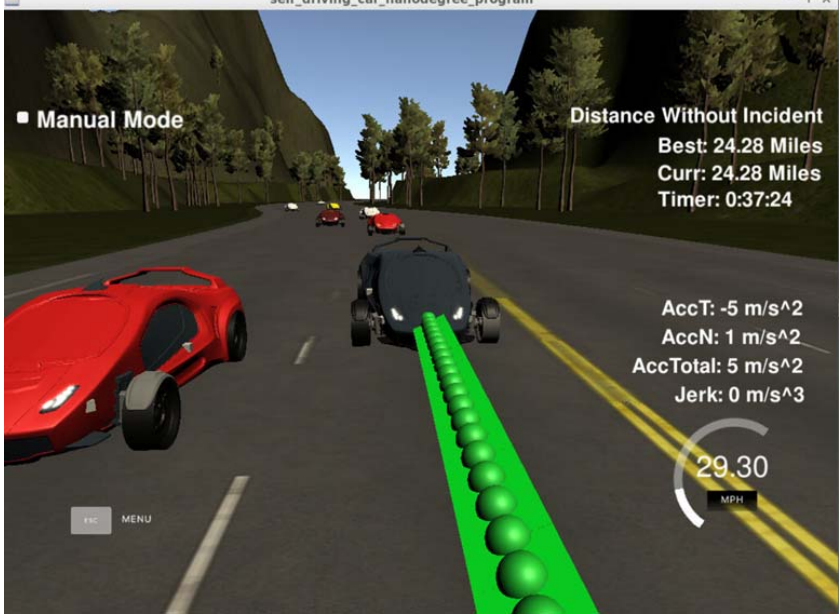


Project Rubric:

CRITERIA	MEETS SPECIFICATIONS
The code compiles correctly.	<p>Code must compile without errors with cmake and make.</p> <p>Given that we've made CMakeLists.txt as general as possible, it's recommend that you do not change it unless you can guarantee that your changes will still compile on any platform.</p> <p>Following project video, I used spline.h (Cubic Spline interpolation implementation).</p> <p>The code complies correctly and fine.</p>
The car is able to drive at least 4.32 miles without incident	<p>The top right screen of the simulator shows the current/best miles driven without incident. Incidents include exceeding acceleration/jerk/speed, collision, and driving outside of the lanes. Each incident case is also listed below in more detail.</p> <p>I ran the code several times with different distances without incident. Below are the screen shot of two of them:</p> 

	
The car drives according to the speed limit.	<p>The car doesn't drive faster than the speed limit. Also the car isn't driving much slower than speed limit unless obstructed by traffic.</p> <p>All good and no red message</p>
Max Acceleration and Jerk are not Exceeded.	<p>The car does not exceed a total acceleration of 10 m/s² and a jerk of 10 m/s³.</p> <p>All good and no red message</p>
Car does not have collisions.	<p>The car must not come into contact with any of the other cars on the road.</p> <p>All good and no red message and no collision</p>
The car stays in its lane, except for the time between changing lanes.	<p>The car doesn't spend more than a 3 second length outside the lane lanes during changing lanes, and every other time the car stays inside one of the 3 lanes on the right hand side of the road.</p> <p>Correct. Changing lane is only for the purpose of passing a vehicle while safe, and it returns to center line in first opportunity. Or vehicle keeps its lane as long as its velocity(50 or 49.5 mph) is maintained or other lanes are not safe</p>
The car is able to change lanes	<p>The car is able to smoothly change lanes when it makes sense to do so, such as when behind a slower moving car and an adjacent lane is clear of other traffic.</p> <p>Yes, as explained above, for passing a car in traffic when its safe, it can change the lane and then return back to center lane in first opportunity. It considers adjusting velocity to new lanes and waiting for an opportunity for safe lane change.</p>

There is a reflection on how to generate paths.

The code model for generating paths is described in detail. This can be part of the README or a separate doc labeled "Model Documentation".

Overall, lots of ideas are generated from courses and video for the project.

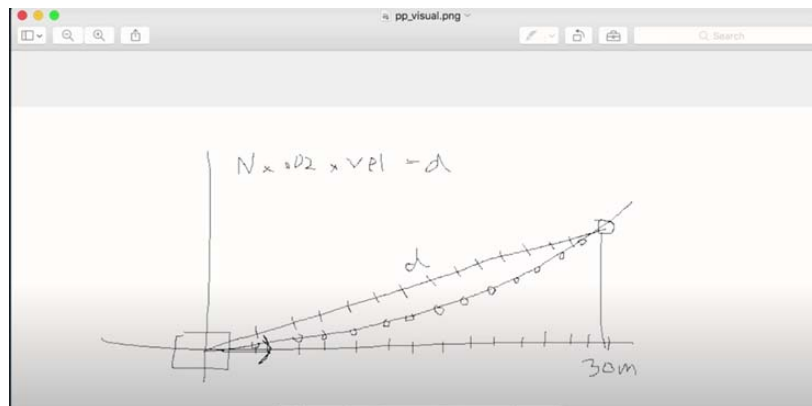
The core of the codes has been written in main.cpp from line 113 to 249.

First, used sensor fusion data for localization and knowing about other vehicles locations e.g. in same line front of us, left, or right of us. Based on video for project, considered minimum 30 meter as safe limit for the car in same lane and front of us.

Then, coding was done for decision making and implementing decisions made while driving in a safe manner.(e.g. increase/decrease velocity, change lanes)

Based on guidance from courses and project video, in order to keep speed as close as possible to 50 miles/hour, considered 49.5 miles/hour as good ideal, and considered .224 (little bit less than half of the difference between 50 and 49.5) to avoid too fast acceleration of the vehicle. Also defined a speed delta which is basically used to coordinate the speed changes during trajectories. In order to increase/decrease speed on each trajectory points instead of doing it for the complete trajectory. This is useful for controlling vehicle speed specially when there is a car in front that needs to avoid collision with it.

Finally it comes to codes for Trajectories. Based on video instructions, last two points were chosen from previous trajectory if available, or using vehicle's own position if trajectories not available. These points were used with three points from far distance for spline initial calculations and by transforming into vehicle local coordinates. Then pass trajectory points are recorded to new trajectory to improve effectiveness of trajectories. Using the concepts learned from courses and project video, spline helps to evaluate the y positions by giving the x positions. (below image from video).



Overall, the path planner works well and satisfactory as shown in screen shots for results above.

CarND-Path-Planning-Project

Self-Driving Car Engineer Nanodegree Program

Simulator.

You can download the Term3 Simulator which contains the Path Planning Project from the [releases tab (https://github.com/udacity/self-driving-car-sim/releases/tag/T3_v1.2)].

To run the simulator on Mac/Linux, first make the binary file executable with the following command:

```
sudo chmod u+x {simulator_file_name}
```

Goals

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 map of the highway is in data/highway_map.txt

Each waypoint in the list contains $[x, y, s, dx, dy]$ values. x and y are the waypoint's map coordinate position, the s value is the distance along the road to get to that waypoint in meters, the dx and dy values define the unit normal vector pointing outward of the highway loop.

The highway's waypoints loop around so the frenet s value, distance along the road, goes from 0 to 6945.554.

Basic Build Instructions

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

Here is the data provided from the Simulator to the C++ Program

Main car's localization Data (No Noise)

["x"] The car's x position in map coordinates

["y"] The car's y position in map coordinates

["s"] The car's s position in frenet coordinates

["d"] The car's d position in frenet coordinates

["yaw"] The car's yaw angle in the map

["speed"] The car's speed in MPH

Previous path data given to the Planner

//Note: Return the previous list but with processed points removed, can be a nice tool to show how far along the path has processed since last time.

["previous_path_x"] The previous list of x points previously given to the simulator

["previous_path_y"] The previous list of y points previously given to the simulator

Previous path's end s and d values

["end_path_s"] The previous list's last point's frenet s value

["end_path_d"] The previous list's last point's frenet d value

Sensor Fusion Data, a list of all other car's attributes on the same side of the road. (No Noise)

["sensor_fusion"] A 2d vector of cars and then that car's [car's unique ID, car's x position in map coordinates, car's y position in map coordinates, car's x velocity in m/s, car's y velocity in m/s, car's s position in frenet coordinates, car's d position in frenet coordinates.

Details

1. The car uses a perfect controller and will visit every (x,y) point it receives in the list every .02 seconds. The units for the (x,y) points are in meters and the spacing of the points determines the speed of the car. The vector going from a point to the next point in the list dictates the angle of the car. Acceleration both in the tangential and normal directions is measured along with the jerk, the rate of change of total Acceleration. The (x,y) point paths that the planner receives should not have a total acceleration that goes over 10 m/s^2 , also the jerk should not go over 50 m/s^3 . (NOTE: As this is BETA, these requirements might change. Also currently jerk is over a .02 second interval, it would probably be better to average total acceleration over 1 second and measure jerk from that.
2. There will be some latency between the simulator running and the path planner returning a path, with optimized code usually its not very long maybe just 1-3 time steps. During this delay the simulator will continue using points that it was last given, because of this its a good idea to store the last points you have used so you can have a smooth transition. `previous_path_x`, and `previous_path_y` can be helpful for this transition since they show the last points given to the simulator controller with the processed points already removed. You would either return a path that extends this previous path or make sure to create a new path that has a smooth transition with this last path.

Tips

A really helpful resource for doing this project and creating smooth trajectories was using <http://kluge.in-chemnitz.de/opensource/spline/>, the spline function in a single header file is really easy to use.

Dependencies

- `cmake` ≥ 3.5
 - All OSes: [click here for installation instructions](#)
- `make` ≥ 4.1
 - Linux: `make` is installed by default on most Linux distros
 - Mac: [install Xcode command line tools to get make](#)

- Windows: [Click here for installation instructions](#)
- gcc/g++ >= 5.4
 - Linux: gcc / g++ is installed by default on most Linux distros
 - Mac: same deal as make - [install Xcode command line tools](<https://developer.apple.com/xcode/features/>)
 - Windows: recommend using [MinGW](#)
- [uWebSockets](#)
 - Run either `install-mac.sh` OR `install-ubuntu.sh`.
 - If you install from source, checkout to commit e94b6e1, i.e.

```
git clone https://github.com/uWebSockets/uWebSockets
cd uWebSockets
git checkout e94b6e1
```

Editor Settings

We've purposefully kept editor configuration files out of this repo in order to keep it as simple and environment agnostic as possible. However, we recommend using the following settings:

- indent using spaces
- set tab width to 2 spaces (keeps the matrices in source code aligned)

Code Style

Please (do your best to) stick to [Google's C++ style guide](#).

Project Instructions and Rubric

Note: regardless of the changes you make, your project must be buildable using `cmake` and `make`!

Call for IDE Profiles Pull Requests

Help your fellow students!

We decided to create Makefiles with `cmake` to keep this project as platform agnostic as possible. Similarly, we omitted IDE profiles in order to ensure that students don't feel pressured to use one IDE or another.

However! I'd love to help people get up and running with their IDEs of choice. If you've created a profile for an IDE that you think other students would appreciate, we'd love to have you add the requisite profile files and instructions to `ide_profiles/`. For example if you wanted to add a VS Code profile, you'd add:

- `/ide_profiles/vscode/.vscode`
- `/ide_profiles/vscode/README.md`

The README should explain what the profile does, how to take advantage of it, and how to install it.

Frankly, I've never been involved in a project with multiple IDE profiles before. I believe the best way to handle this would be to keep them out of the repo root to avoid clutter. My expectation is that most profiles will include instructions to copy files to a new location to get picked up by the IDE, but that's just a guess.

One last note here: regardless of the IDE used, every submitted project must still be compilable with `cmake` and `make`.

How to write a README

A well written README file can enhance your project and portfolio. Develop your abilities to create professional README files by completing [this free course](#).

