CarND-Path-Planning-Project

Self-Driving Car Engineer Nanodegree Program

Overview

In the Path Planning project a vehicle shall be controlled to drive on a highway with 3 lanes in a simulator where other cars have to be observed and some other restrictions like maximum velocity and acceleration apply (refer to section "Goals" for details.

Reflection

The project bases on the code provided by the seed project and the code presented in the project walkthrough. This code realizes the communication with the simulator and the path generation to drive the vehicle along a lane.

Additional algorithms to process the sensor data to extract the position and speed of the other cars and to control the velocity of the ego car have been added.

The position of the other cars are extracted in src/main.cpp line 285 to 325. The relative positions of all cars in respect to the ego car are calculated and for each lane the nearest cars in in front of the ego car and behind are stored. The speed controller implemented in src/SpeedController.cpp uses this information to set the target speed and the target lane of the ego car.

If a car is ahead in the same lane and the distance is less than 50 meters, the speed controller checks if it safe to change to an adjacent lane. If yes it will set the target lane accordingly src/SpeedController.cpp lines 88 to 110). If no it will decrease the speed using a PD controller to prevent a collision with the car ahead (src/SpeedController.cpp lines 37 to 47). The speed controller is also responsible the consider the limits of the velocity, the acceleration, and the jerk (src/SpeedController.cpp lines 48 to 75)

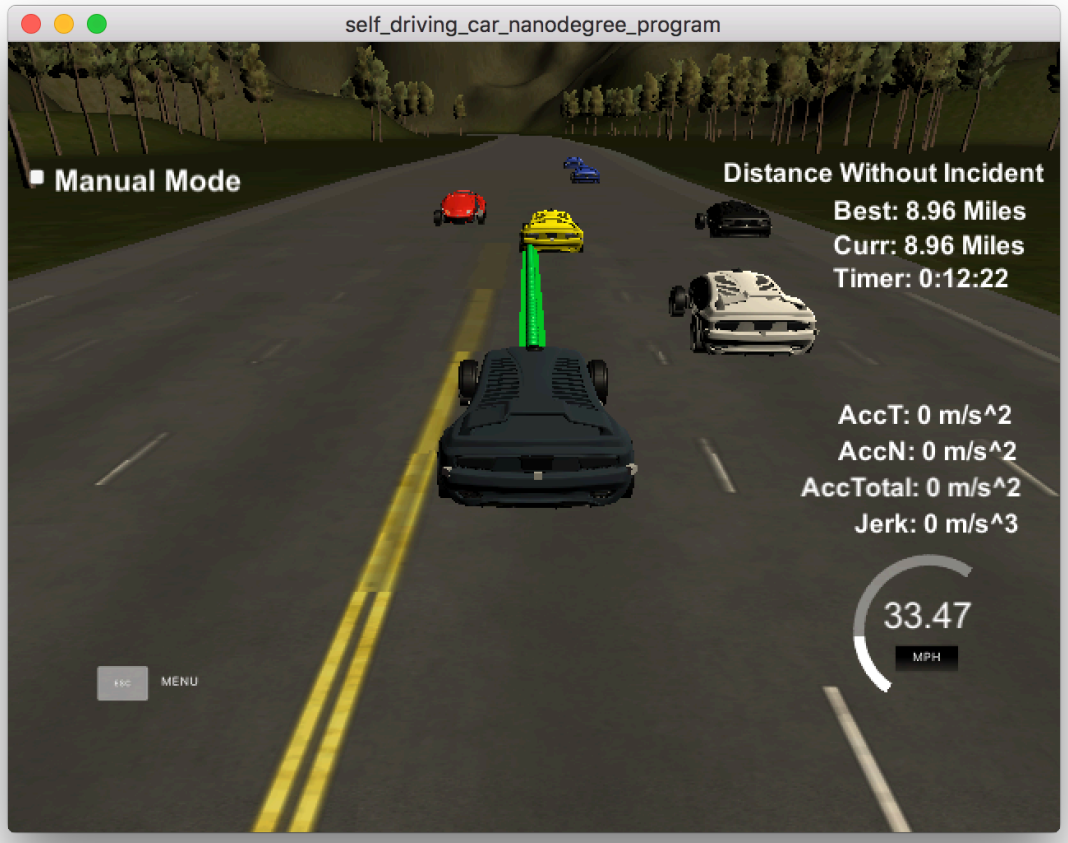
The target lane and target speed will be used to calculate the next waypoints in src/main.cpp lines 375 to 440 as presented in the project walktrough.

The implemented algorithms are able to:

- drive the car over a distance of more than 4.32 miles without incident
- control the car according to the speed limit
- not exceed the max acceleration and jerk
- prevent the car from having collisions
- let the car stay in its lane, except for the time between changing lanes
- let the car change lanes.

Here are two screenshots:


self_driving_car_nanodegree_program
Manual Mode
Distance Without Incident
Best: 4.61 Miles
Curr: 4.61 Miles
Timer: 0:05:59
AccT: 0 m/s^2
AccN: 0 m/s^2
AccTotal: 1 m/s^2
Jerk: 0 m/s^3
36.49
MPH
MENU



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).

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 recieves 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 recieves 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 is in a single hearder 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.