CarND-Path-Planning-Project

Self-Driving Car Engineer Nanodegree Program

Reflection

Trajectory Planning

In order to navigate a car through a highway, the car needs to know where it is located, and where are the other cars as well. To perform such operation in cartesian coordinates it is a complicated matter since the highway does not follow a straight path and its center coordinates change continuously. Therefore, the Frenet coordinates are used where the center of the highway is the “Zero” point for the “d” coordinate that runs perpendicular to the highway, and the “s” coordinate runs along the highway. Taking this into account, if we want the car to travel on a single lane, the “d” coordinate would remain constant though time, and the “s” coordinate would increase to move forward and decrease to go backwards.

The simulator provides us with the map waypoints, but these are unevenly spread throughout the highway, and since our car will visit one point every 0.2 seconds, the points passed back as trajectory to the simulator need to be spread accordingly in order to achieve a desired velocity.

Instead of using the Frenet coordinates given by the simulator, only three points (30, 60, and 90 mts for “s” and “d” corresponding to the desired lane) are used to achieve a path that is nice and smooth, and does not experience any max acceleration or high jerk values.

To make the car speed up and slow down at a constant acceleration less than 10 m/s^2, the current velocity is gradually incremented or decremented by a constant value until it is matched to the reference velocity.

As mentioned before, the car will visit one point every 0.2 seconds. For that reason, the separation of the waypoints needs to be accordingly calculated so the car can travel at the desired speed. To do this the three previously defined points are converted to the local vehicle coordinate system, having the heading of the vehicle be Zero degrees. By doing this, the spline of the points can be linearly approximated.

Behavior Planning

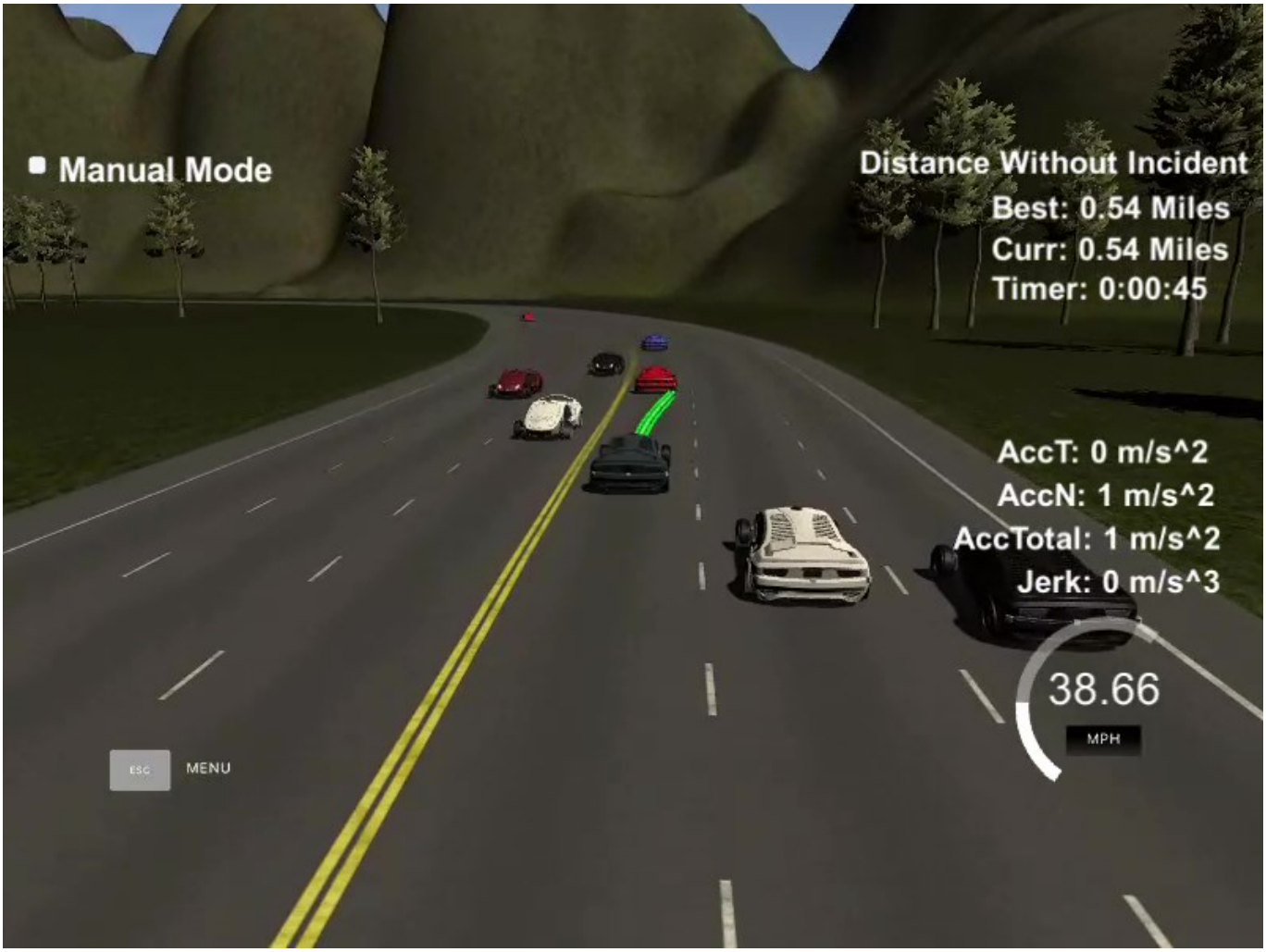
For the behavior planning specific rules are applied. Such rules use a calcualted "Safe Distance" that deppends on the speed of the vehicle.

- If there is no other car in front of the vehicle, the reference speed is set to 49.5 MPH.
- The ego vehicle will try to always drive on the right lane of the road. In case the vehicle is not on the right, it will try to navigate swithcing lanes to the right if the following rules are met:
    - There is no car in the lane to the right blocking its way
    - The car is traveling at least at 40 MPH
    - No overtake procedure is being performed
- If a car is detected in front of the vehicle traveling at a slower speed and closer than the "Safe Distance", the vehicle will initiate an overtake procedure.
    - First it will search for a posibility to move to the left lane (if not already on the furter left lane). The vehicle will not move to the left if:

        - There are cars on the left lane at certain distance to the front and to the back
        - There is a car traveling at least 5% slower than the car in front, and is not at least 5 mts further away
    - If no overtaking on the left is possible, the vehicly will try to pass on the right following the same rules
    - If no overtake maneuver is possible at all the vehicle will follow the next rules
        - Maintain speed if the distance to the vehicle is grater than 2/3 of the Safe Distance
        - Gradualy reduce the speed by 10% of the speed diference if the distance is smaller than 2/3 of the Safe Distance
        - Match the vehicle's speed if the distance is smaller than 1/3 but more that 1/4 of the Safe Distance
        - Travel 5MPH slower that the other vehicle if the distance is smaller than 1/4 of the Safe Distance

The lane change logic is quite simple, if there was a car in front of the ego vehicle then it would see if it was safe to change to the left lane, if the left lane was not safe then it would try to change to the right lane. Some improvements could be made to this lane change algorithm by using a cost function to decide what lane is the optimal. Currently the car was able to go 4.17 miles around the highway without any incidents in 5:10 minutes.

Video



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 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.