CarND-Path-Planning-Project

Self-Driving Car Engineer Nanodegree Program

Overview & Implementation:

- The two major challenges of this project were
  1. To design a state machine that makes a reasonable decision in terms of staying in lane (KL) and lane change (LC)
  2. To design trajectory points that do not violate jerk requirement during a lane change
- These are the key hyper-parameters I had to tune:
  - **min_gap** is the distance threshold, in meters, between main-car and other-car. when **min_gap < 25** one second in the future, the main-car may attempt to change lanes
  - tighter bounds are applied on adjacent lanes: if no car is detected between **distance_front = 15**, and **distance_back=-10** then the adjacent lane may be available
  - I set maximum speed to be **max_speed = 21.58** meters per second which corresponds to 48.5 < 50 miles per hour
  - I set maximum longitudinal acceleration **max_acc = 8** meters per second square. I used this variable to set velocity change. I did not explicitly measure and control jerk. I controlled jerk using spline and the tweaking the distance between sparse samples
  - Not to continuously change lanes, I introduced **freeze_set=30** which is set when a lane change is decided. Then for 30 samples I do not make additional decisions
  - I used **N=60** trajectory points which seems like a reasonable trade-off between smooth right and response time
  - For spline I used five points to set the coefficients: two for the starting point and **N_sparse=3** additional points placed **delta_s_sparse=30** meters apart

Valid Trajectory:

(1) Speed limit:

- As discussed above, the maximum velocity was set at 48.5 mph which is < 50 mph
- If there is no car in front obstructing the lane, the main-car attempts to achieve 48.5 mph

(2) Maximum Acceleration:

- As discussed above, maximum acceleration was set to $8\ \mathrm{m/s\hat{}2}$ which is less than 10
- The maximum change in speed in interval Δt was set to $\Delta v = 8\,\Delta t$
- At the beginning, where the car is at rest, to get the car accelerated quickly, I used the dynamics

  $$s = \frac{8}{2}t^2$$

  to determine the non-equal trajectory intervals, where s is longitudinal displacement

(3) Maximum Jerk:

- Jerk is problematic during lane changes
- I did not use specific car dynamic models to constrain the jerk
- the following features addressed the jerk constraint of $10\ \mathrm{m/s\hat{}3}$
    1. use spline interpolation to generate a smooth trajectory
        - used spline.h library as recommended for project
    2. to construct spline, I used 30 meters intervals so that the trajectory is smooth
        - I was getting jerk violations with 25 meter intervals
    3. I used previous path, to sustain continuity of the trajectory

(4) Collisions & sate machine:

- Collisions is a concern when
    - there is a car in front with slower velocity
    - the main-car is performing a lane change
- Car front:
    - When there is a slower car in front, the main-car will attempt to make a lane change
    - If lane change is not an option, the main-car will slow down

(5) Staying in lane, changing lane & state machine

- Staying in lane is simple, we just set the displacement in SD car coordinate system to 2, 6, or 10 depending on which lane we are
- Lane change:
    - As discussed in Overview, for lane change to occur, two conditions need to be satisfied to prevent collision during the lane change process:
        1. there should be no current cars, in that lane, between 25 meters forward and -10 meters backwards, with respect to main-car
        2. there should be no other cars, in that lane, between 15 meters forward and -10 meters backwards, with respect to main-car
- The state machine of the car is simple: it has two states, stay in lane or change lane
    - state machine is controlled by boolean variable mode_lc
    - If the main-car is in middle lane and want to change lanes, then the priority goes to left lane change. If left lane is busy then right lane change is checked
    - Furthermore, memory into the car state machine lane change was introduced by using a clock, where for 30 samples no lane change is allowed once a decision is made to change lanes

Reflections:

- One of the challenges of this project was to deal with 4 different coordinate systems and transforming between them:
    1. XY car coordinates
    2. XY map coordinates
    3. SD car coordinates
    4. SD map coordinates

In addition to getFrenet and getXY functions that were already available, I added the functions **XY_map2car** and **XY_car2map** functions to transform between car and map coordinates

- Another challenge was designing the state machine, to decide when to switch lanes, which was discussed above
- The main challenge was to get the smooth projection path using the spline, which also was discussed above
- Waat can be improved:
    - The state machine may be improved to make it smarter,
        - for example consider both lanes and make choose the better
    - Path generation can also be improved during lane changes
        - for example generate multiple projection paths, associate each with a cost function and pick the path with least cost introduce cost

Generic Information:

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).

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.