UDACITY

PROJECT

Path Planning

A part of the Self-Driving Car Engineer Program

PROJECT REVIEW

CODE REVIEW 1

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Dear Udacian,
Congratulations on completing this project with your very first submission.👏🏼
The ideas implemented were very good and computationally efficient which improved the smoothness of the vehicle's driving. The documentation was also very explicit and outlined the various steps and techniques used in planning a safe path for the ego vehicle. It was delightful reviewing your work, and I exhort you to keep up the good work which will make you an outstanding Self Driving Car Engineer. Best of luck with the rest of your projects.

Extra material

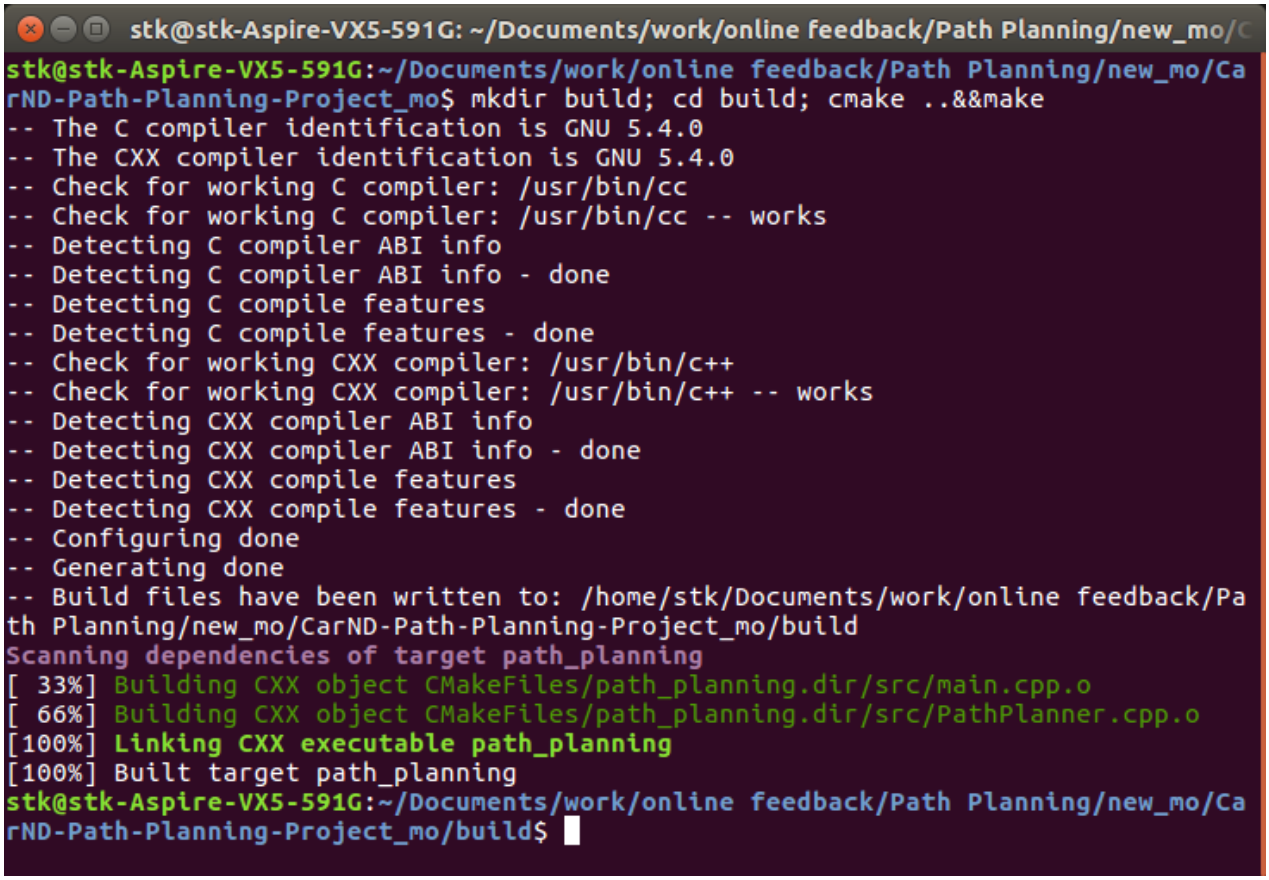
- Udacity CS373: Programming a Robotic Car Unit 4: Motion
- Path Planning and Collision Avoidance
- Safe Motion Planning for Autonomous Driving
- Local and Global Path Generation for Autonomous Vehicles Using Splines
- Medium- Path Planning in Highways for an Autonomous Vehicle
- Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions

Compilation

Code must compile without errors with `cmake` and make.

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.

Great work writing code which compiles successfully with cmake and make. This is evidence that no compilation errors were recorded in the code. 👍



Suggestions and Comments

To know more about cmake and make, visit these links:

- Cmake FAQS.
- Using make and writing Makefiles.
- Youtube set of tutorials on using make and writing Makefile.
- MakeFiles

Valid Trajectories

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.

The top right screen of the simulator successfully shows the best miles driven without an incident. Good work!

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.

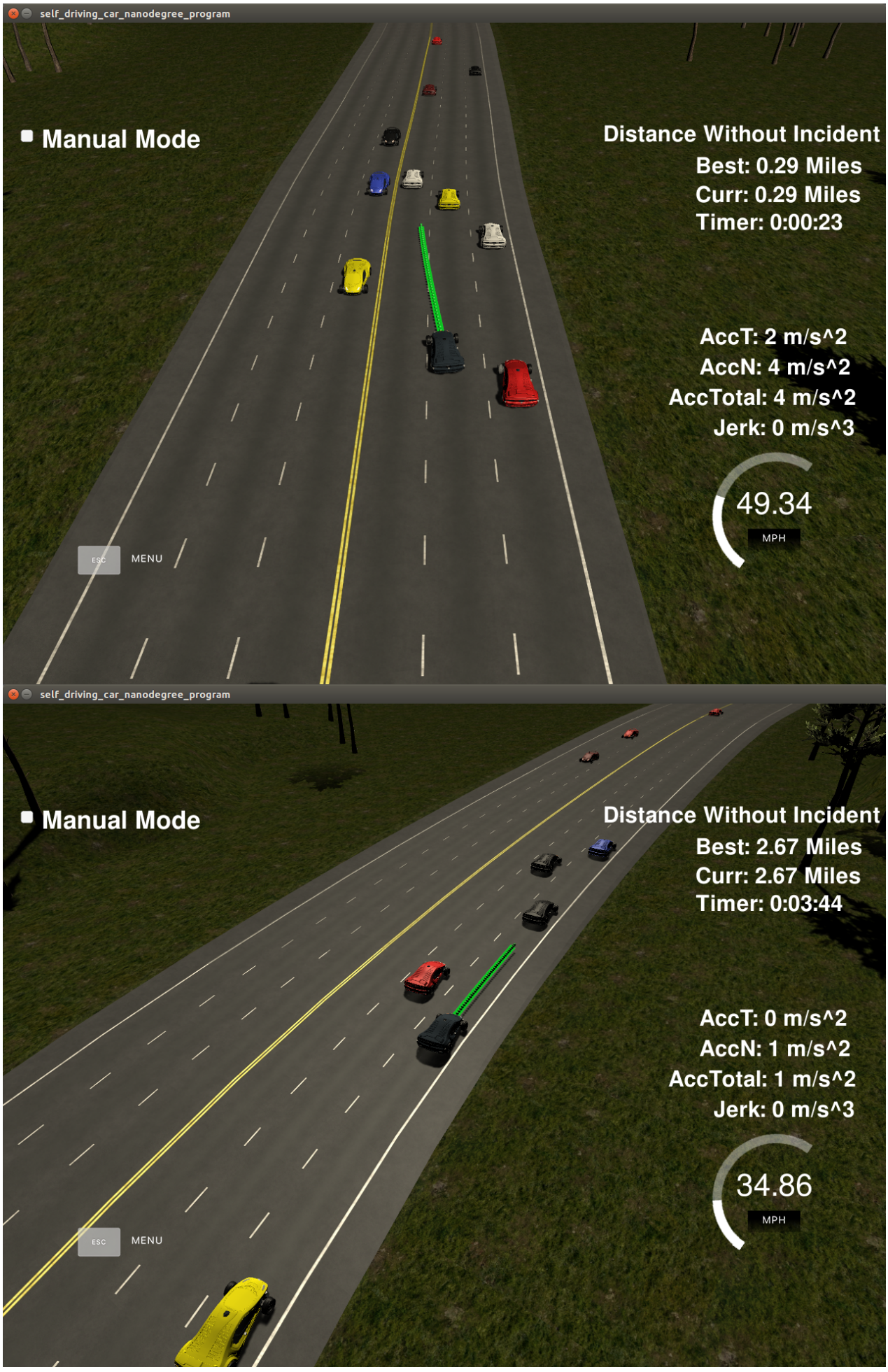
The vehicle coasted through a full lap of the highway without exceeding the speed limits set on the highway. This is very good work and shows how much time and commitment was put into completing the project. Keep it up!

The car does not exceed a total acceleration of 10 m/s^2 and a jerk of 10 m/s^3.

The vehicle successfully drives through the highway without exceeding the total acceleration of 10m/s^2 and jerk of 10m/s^3. Excellent work!

The car must not come into contact with any of the other cars on the road.

The vehicle slows down when close to the leading vehicle. It also changes its lane when a lane change is possible and safe. This helped in avoiding any collision with other vehicles on the highway. Good work!

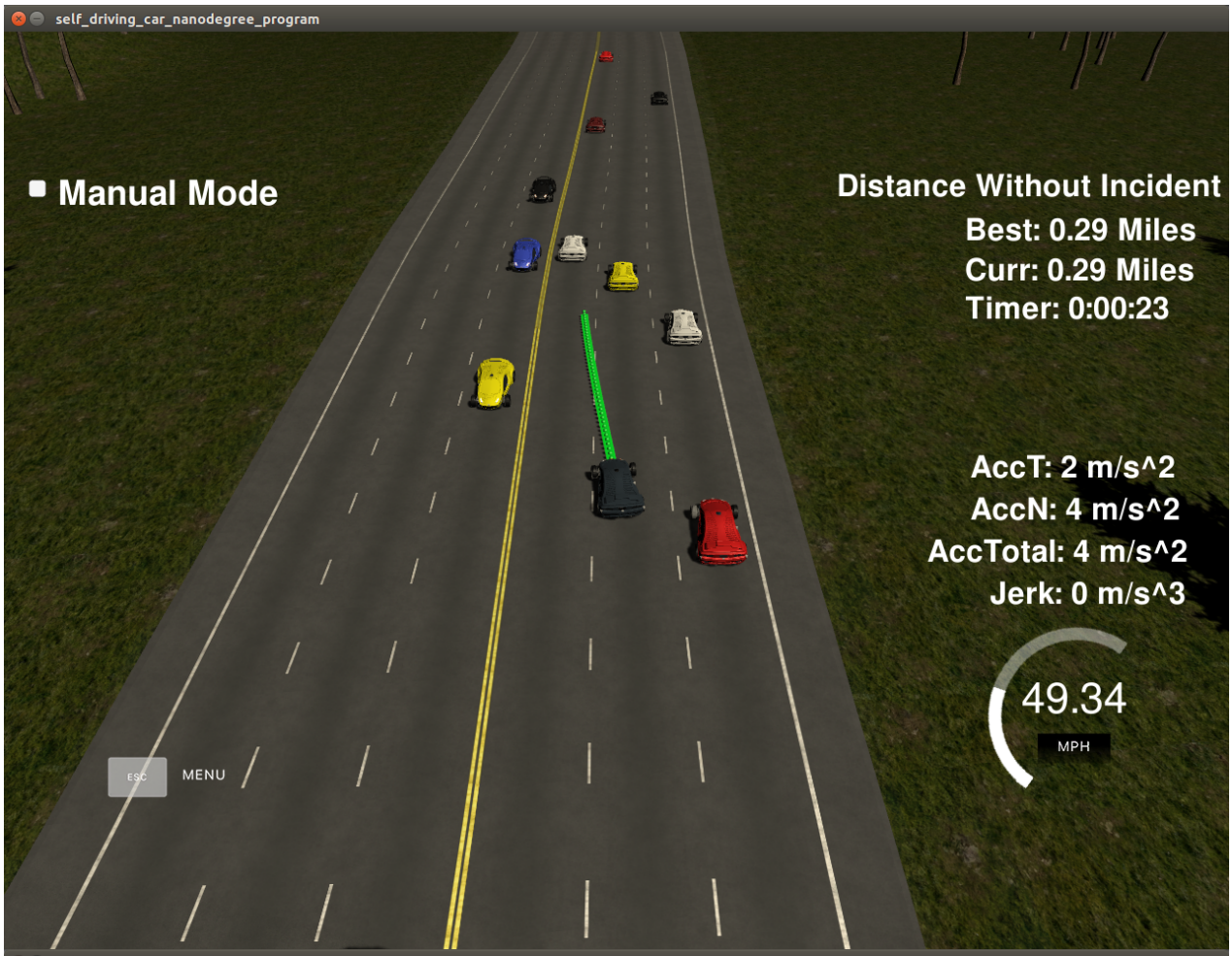

self_driving_car_nanodegree_program
Manual Mode
Distance Without Incident
Best: 0.29 Miles
Curr: 0.29 Miles
Timer: 0:00:23
AccT: 2 m/s^2
AccN: 4 m/s^2
AccTotal: 4 m/s^2
Jerk: 0 m/s^3
49.34
MPH
ESC
MENU
self_driving_car_nanodegree_program
Manual Mode
Distance Without Incident
Best: 2.67 Miles
Curr: 2.67 Miles
Timer: 0:03:44
AccT: 0 m/s^2
AccN: 1 m/s^2
AccTotal: 1 m/s^2
Jerk: 0 m/s^3
34.86
MPH
ESC
MENU

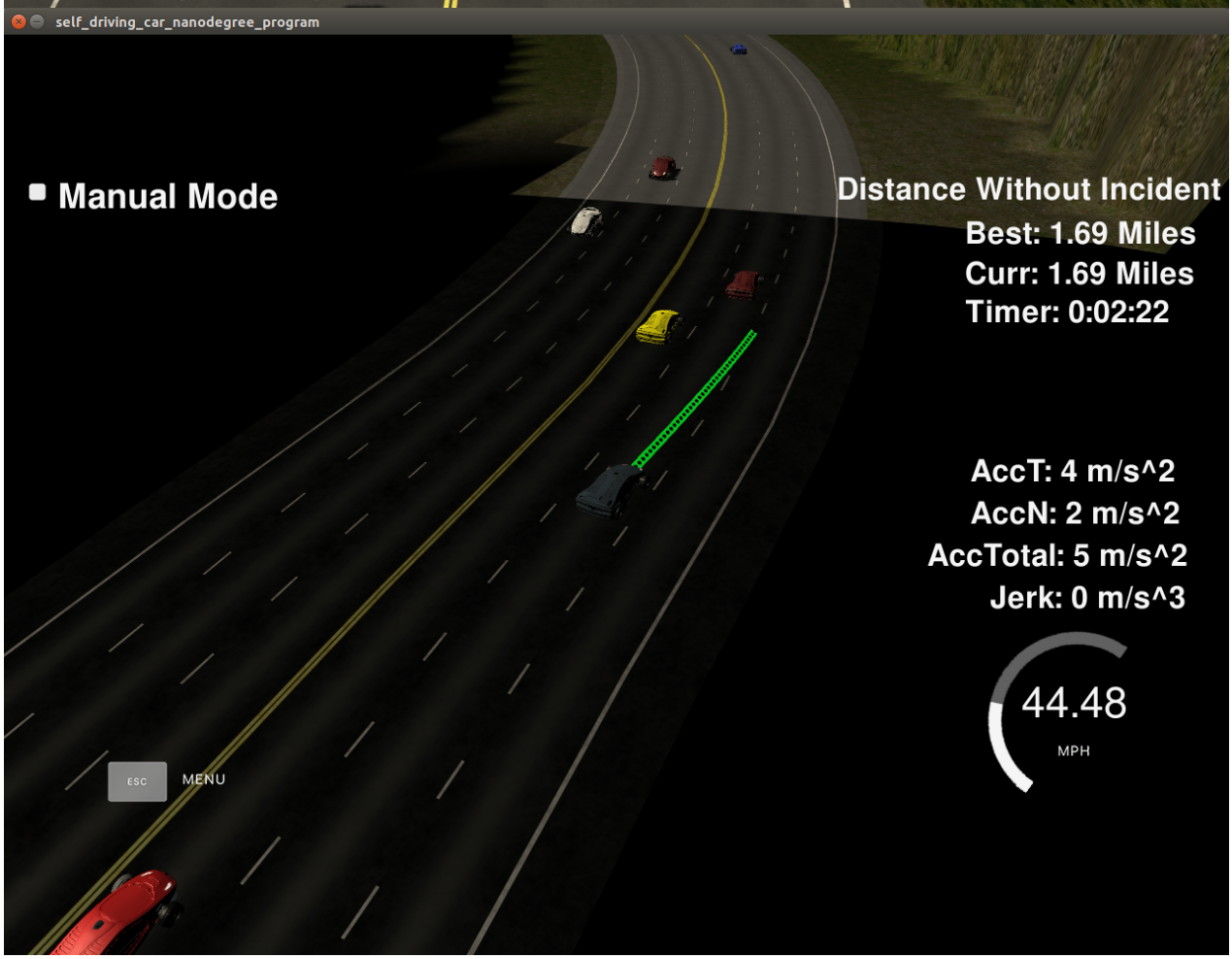
The car doesn't spend more than a 3 second length out side the lane lanes during changing lanes, and every other time the car stays inside one of the 3 lanes on the right hand side of the road.

The vehicle is very swift when changing lanes and does not exceed the 3s limit on the lane lines while performing lane change. The vehicle does not also spend 3s outside the lane when going through the track. Good work!

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.

Lane change is quite smooth and occurs without exceeding jerk and maximum acceleration limits. Very good work!


self_driving_car_nanodegree_program
Manual Mode
Distance Without Incident
Best: 0.29 Miles
Curr: 0.29 Miles
Timer: 0:00:23
AccT: 2 m/s^2
AccN: 4 m/s^2
AccTotal: 4 m/s^2
Jerk: 0 m/s^3
49.34
MPH
ESC
MENU


self_driving_car_nanodegree_program
Manual Mode
Distance Without Incident
Best: 1.69 Miles
Curr: 1.69 Miles
Timer: 0:02:22
AccT: 4 m/s^2
AccN: 2 m/s^2
AccTotal: 5 m/s^2
Jerk: 0 m/s^3
44.48
MPH
ESC
MENU

Reflection

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

A comprehensive writeup was added to the project which explains, in details, the various techniques used to plan a safe path for the ego vehicle. :+thumbsup:

Model Documentation

> Note: All Path Planning logic can be found in its own PathPlanner class including the already given helper functions such as: `getXY`, `distance`, `ClosestWaypoint`, and `NextWaypoint`.

- The model starts by initiaing it in `main.cpp` line 65, where `map_waypoints` are read from input data file `../data/highway_map.csv` and passed as a paramter to the `PathPlanner` class constructor.
- Then reference velocity and starting lane are set on lines 66, and 67 respectively.
- Then the following code runs on the event of recieveing a message from the simulator.
  - On lines 88-126, read the following inputs from the simulator in corresponding variables and pass it to the `path_planner` instant:
    - car x, y, s, d, yaw, and speed
    - previous_path x, and y
    - end_path s, and d
    - scan of other (12) cars positions on the road in `sensor_fusion` vector
  - On line 128, the `calc_next_xy_vals()` method of the path_planner object is called and which includes all the path planning logic and explained in more details below
  - On lines 131, and 132, returns `next_x_vals`, and `next_y_vals` vectors calculetd by `calc_next_xy_vals()` in the previous step respectively.

`calc_next_xy_vals()`

Starting from lines 144-151 in `PathPlanner.cpp` file. Initialize path_planner state to default values.

Lines 154-170, checks whether a previous path info can be found or not, in case it is found, the model make use of it.

Lines 172-210, the model tries to find a suitable reference velocity by scanning through the current lane the car found at, and check whether there are any others car are ahead or not (in the range of 30 m ahead). The model not only checks the cars distance from our car only but their velocities as well. if there is a car in the range of 20-30 m, match the car ahead speed, and prepare to change lanes. if the car is closer than 20 m, match the car ahead speed - 5 mph.

Lines 212-278 describes change lanes logic, where the model starts by checking whether it is safe to change lane to the left first, if not safe, then check change lane to the right, taking into consideration whether we are at the left most lane or right most one.

Lines 281-380, are the lines where the model generates `next_x_vals`, and `next_y_vals` by transforming corrdinates to the car local frame, pick some anchor points, which are used later on with the spline `set_points` to fill other points inbetween.

> Note: This is the model demonstrated by David Silver & Aaron Brown in the Project's Walkthrough And Q&A video. I tried hard to do something different but time run out and I am worry about my graduation

Suggestions

A cost-based implementation along with finite states could work quite well for the highway scenario. Each state could be accompanied by a cost which will be affected by factors like proximity to other vehicles, the speed of

leading and rear vehicles, lane number, previous lane changes and so on. The state with the least cost could then be chosen, and this makes the work scalable as more states could be added eventually.

DOWNLOAD PROJECT

1 CODE REVIEW COMMENTS ›

RETURN TO PATH

Student FAQ