

[◀ Return to "Self-Driving Car Engineer" in the classroom](#)[DISCUSS ON STUDENT HUB](#)

# Path Planning

## REVIEW

## CODE REVIEW

## HISTORY

### Meets Specifications

Hello Student,

Congratulations 🎉🏆

The working project is very outstanding and this is a clear reflection of the handwork and commitment made on the project. Keep it up on your next projects . I really enjoyed it 🙌

Courage and good luck moving forward from here . Stay audacious 🦊

Extra learning on the project

Here are a few resources you might find useful for more insight and further learning.

- [The path planning problem in depth](#)
- [A discussion on What is the difference between path planning and motion planning?](#)
- [Introduction to Robotics #4: Path-Planning](#)
- [Introduction to robot motion: Robot Motion Planning](#)
- [Introduction to robot motion: Path Planning and Collision Avoidance](#)

### Extra Resources for learning C++

- [A Complete Guide to Programming in C++](#)
- [C++ overview](#). You can also click on [Tutorials](#) to learn any other programming language apart from C++.
- [C++ Tutorials](#)

## 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.

Well done. Starting nicely as code compiles with `cmake .. && make` without any error 

```
ath-Planning-Project/build
Scanning dependencies of target path_planning
[ 50%] Building CXX object CMakeFiles/path_planning.dir/src/main.cpp.o
[100%] Linking CXX executable path_planning
[100%] Built target path_planning
```

## Additional Knowledge on Cmake:

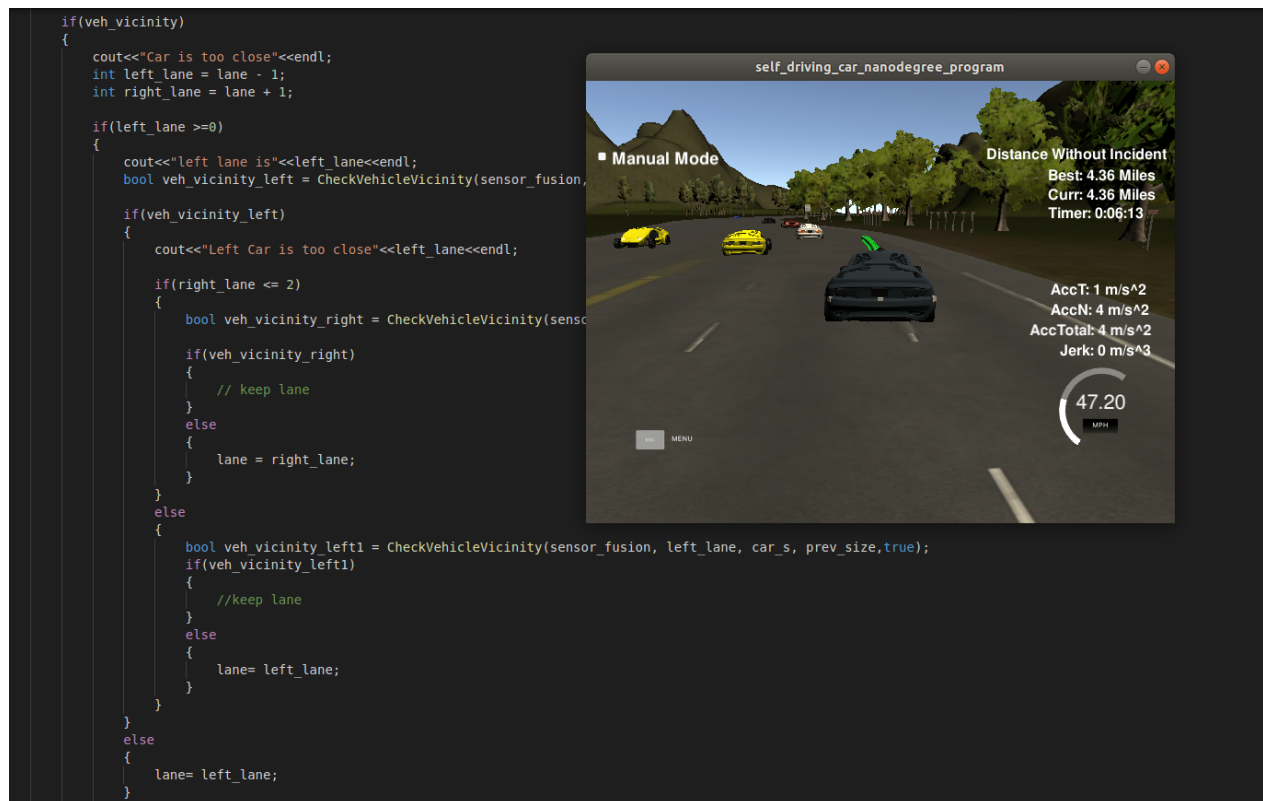
If it interest you, then you can learn more about it from the following ;

- [cmake FAQs](#)
- [cmake tutorial guide](#)
- [Introduction to CMake by Example](#)
- [How to Build a CMake-Based Project](#)

## 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.

Excellently done. The ego car has driven for over 4.32 miles without any of the incidents like exceeding acceleration/jerk/speed, collision or driving outside of the lanes 🙌



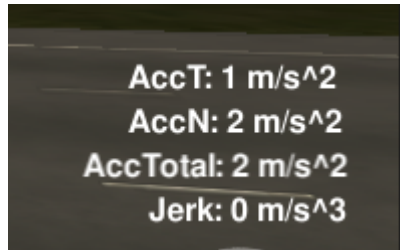
The car doesn't drive faster than the speed limit. Also the car isn't driving much slower than speed limit unless obstructed by traffic.

Great job done! The speed of the car has been wisely adjusted based on the traffic (how close the ego car is to a slow moving car that is in front of it). You did very well by controlling the speed of the car so that it does not exceed the 50MPH speed limit and avoiding collisions by slowing down dense traffic when a lane change isn't feasible( as in line 186 of main.cpp) 🙌




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

Commendable work and results! Throughout the entire simulation, total acceleration and jerk were maintained below specific values. The values were adequately controlled by wisely adding and subtracting .224 to adjust and maintain the vehicle's acceleration and jerk within specified limits. 🙌




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

The car didn't get in contact with other cars during the simulation. Good job reducing speed of the car in scenarios of dense traffic where a lane change is not feasible. This indicates that this section of your implementation works perfectly. Nice work! 


Here are a few links with useful information efficient path planning and collision avoidance.

- [The path planning problem in depth](#)
- [A discussion on What is the difference between path planning and motion planning?](#)
- [Introduction to Robotics #4: Path-Planning](#)
- [Introduction to robot motion: Robot Motion Planning](#)
- [Introduction to robot motion: Path Planning and Collision Avoidance](#)

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.

Coupled with some other prior logic in `CheckVehicleVicinity()` function, Lane changing has been effectively implemented between lines 123-184 of main.cpp and indeed, the car was always in one of the three lanes and it didn't spend more than the recommended time (3 seconds) when crossing to a different lane. This is even more reflected in the fact that there wasn't a red flag for driving out of lane during the simulation. 

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.

Well done here. The ego car could smartly sense it's surrounding environment to decide if a lane change is possible or not and smoothly do a lane change only when surrounding traffic is clear else the ego vehicle's speed is reduce to match speed of the preceding vehicle 

## 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".

Beautiful description as on the `README_Project 9 - Path Planning.pdf` file and it's indeed inline with your code. Nice work and keep it up!

 [DOWNLOAD PROJECT](#)

[RETURN TO PATH](#)

Rate this review