Path Planning

Writeup

Path Planning Project

In this project the goal is to safely navigate around a virtual highway with other traffic that is driving +-10 MPH of the 50 MPH speed limit. The car's localization and sensor fusion data is provided and 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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Compilation

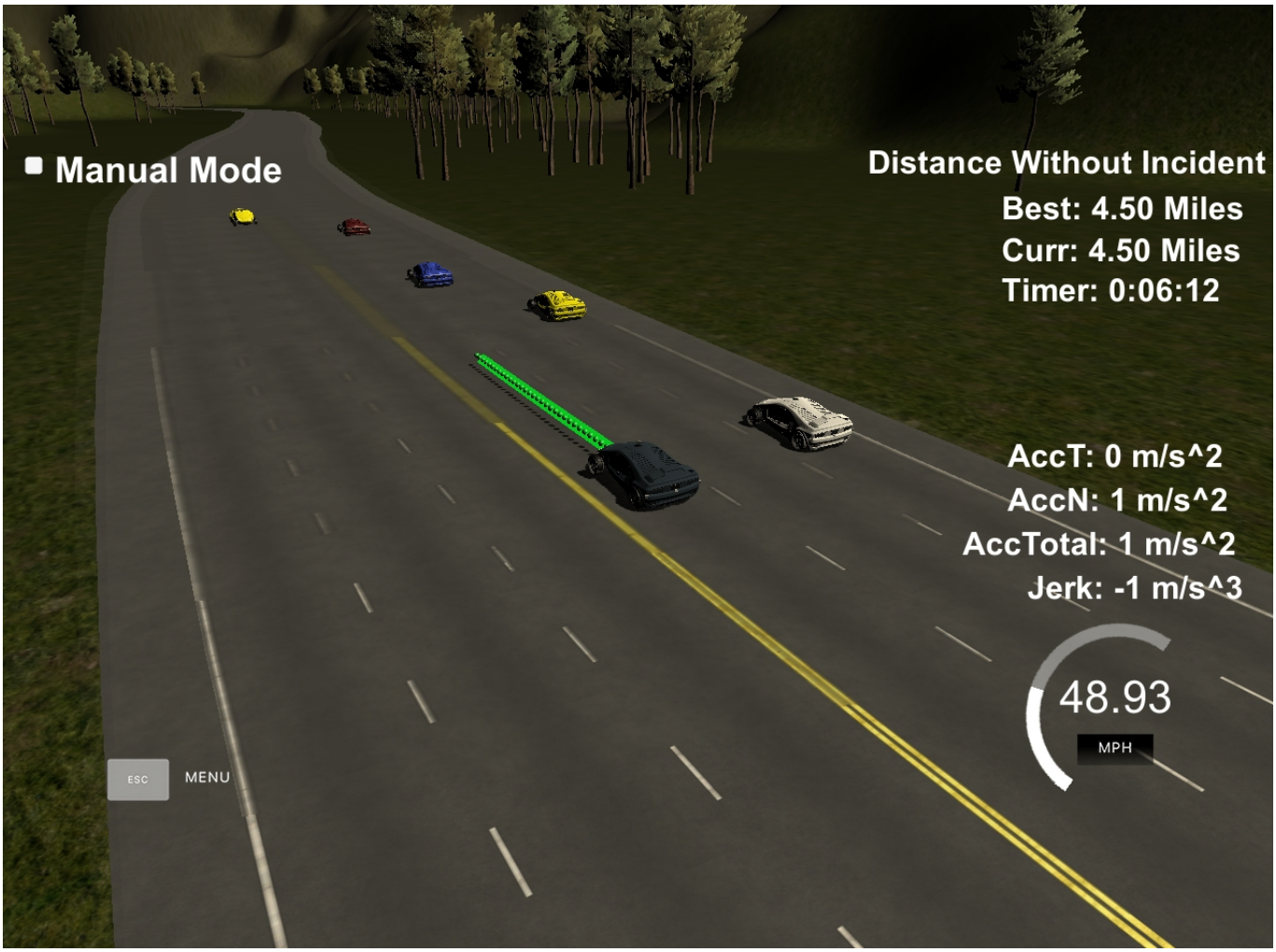
1. The code compiles correctly.

The project can be compiled using both cmake and Visual Studio. For more information, check out the readme.md.

Valid Trajectories

1. The car is able to drive at least 4.32 miles without incident.

The car is able to drive around the track without incident.



2. The car drives according to the speed limit.

The car has been set to drive at exactly 49 MPH. When encountering traffic, the car will reduce its speed to avoid collisions and change lanes easier.

3. Max Acceleration and Jerk are not Exceeded.

The car's acceleration and decceleration has been limited to 10 m/s^2. To avoid jerk, the paths are smoothed using tk's spline library. More precisely, each trajectory is computed from 5 anchor points, 2 at the beginning and 3 spread out across 90 meters. A spline is then fitted on these anchor points and sampled according to the car's velocity. A total of three trajectories have been implemented (see files *./src/Trajectory.cpp* and *./src/Trajectory.h*): KeepLaneTrajectory, PrepLaneChangeTrajectory and LaneChangeTrajectory.

4. Car does not have collisions.

To avoid collisions, two strategies have been implemented. First, the car actively slows down if another car occupies its lane at a distance of 20 meters or less. The magnitude of the deceleration is inversely proportional to the distance.

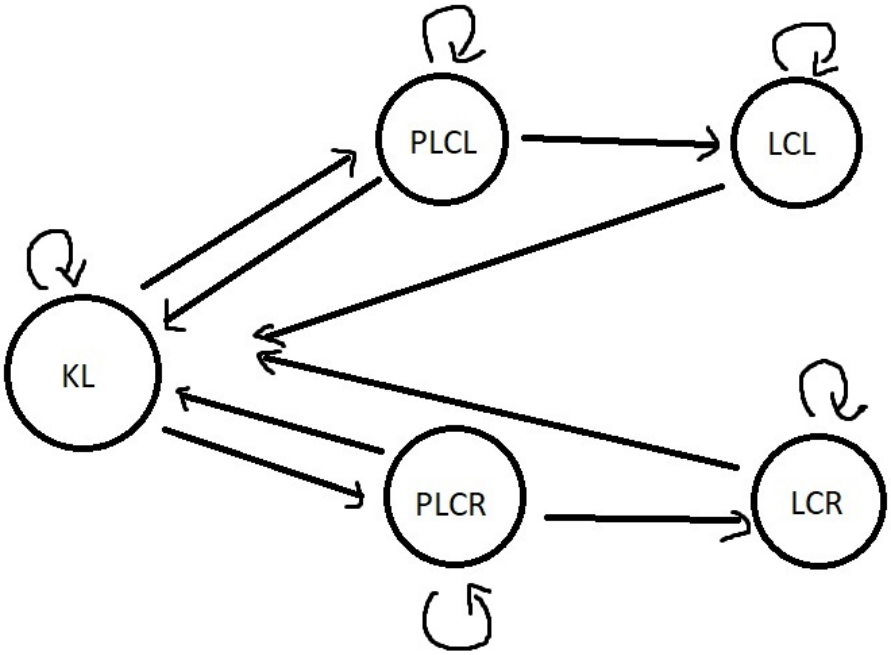
Second, before changing lanes, a cost function tests whether the target lane is free or not. Only in the case of a free lane, a lane change is carried out.

5. The car stays in its lane, except for the time between changing lanes.

A finite state machine controls when the car changes lanes. The default state is KeepLane (KL), during which the car will use the frenet coordinates and the map's checkpoints to stay in its lane. The state machine is implemented in the *./src/Behavior.cpp* and *./src/Behavior.h* files.

5. The car is able to change lanes

To switch lanes, the car must transition to one of the lane change states (LaneChaneLeft - LCL, LaneChangeRight- LCR). Prior to any lane change, the car muse first transition to a preparation state (PrepareLaneChaneLeft - PLCL, PrepareLaneChangeRight- PLCR). In this preparation state, the care will attempt to match the speed of the target lane, as well as reduce the speed in an effort to find a free spot to initiate a change.



All state transitions are controlled by the following cost functions (see files *./src/Cost.cpp* and *./src/Cost.h*):

- Lane Change: penalizes lane change if there is no room on the other lane
- Distance: penalizes states which have the care move on a lane blocked by other vehicles. The magnitude of the penalty is proportional to the nearest car.
- Laziness: generally penalizes state changes, discouraging rapid transitions

Reflection

1. There is a reflection on how to generate paths.

You are reading it 😃