Valid Trajectories

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

My project was able to meet this space without any incidents. And even more below is a screenshoot for the model after 4.32 and after also 16 miles

The car drives according to the speed limit.

Due to the constraints added in the code the model does not exceed the MAX_SPEED. Also I have made the max_speed with 48.7 mph less than allowed max_speed . This to make sure if any deviation happened for the speed to not exceed the allowed speed. Actually, the idea is indirectly taken from the lessons but in in the lesson we penalize the increased speed linearly but here we just don't allow that in the code. The max_speed is defined in the constant.cpp file.

Max Acceleration and Jerk are not Exceeded.

To avoid longitudinal jerk I don't increase the value of the speed by more than 0.224 mph in 0.02 sec time frame and for the latitudinal jerks I use the spline library for finding the path.

Car does not have collisions.

I penalize the collision and I give the vehicle a certain safe space to avoid the collision, check cost.cpp.

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

To make sure that the car will not move to another lane unless there is a reason to do so I added change_lane_cost function. This function penalizes the lane changing, however, not much o if there's a car in front of our vehicle we should change the lane.

The car is able to change lanes

Using a Finite state machine we calculate the cost of the possible options and in case of there is a car in front of us we penalize this and get better cost for prepare change lane(Right/Left) after what would be better.

Refelection

We can compose our model to three main steps:

1. Update watching list
2. `Update our vehicle`
3. `Path generator`

1. Update watching list

To apply this step we use this function ***`PathPlanningManager::updatewatchinglist.`*** `In this function we use the sensor fusion data to generate the predictions. This is very important step as after it we can know the state of the vehicle around us.`

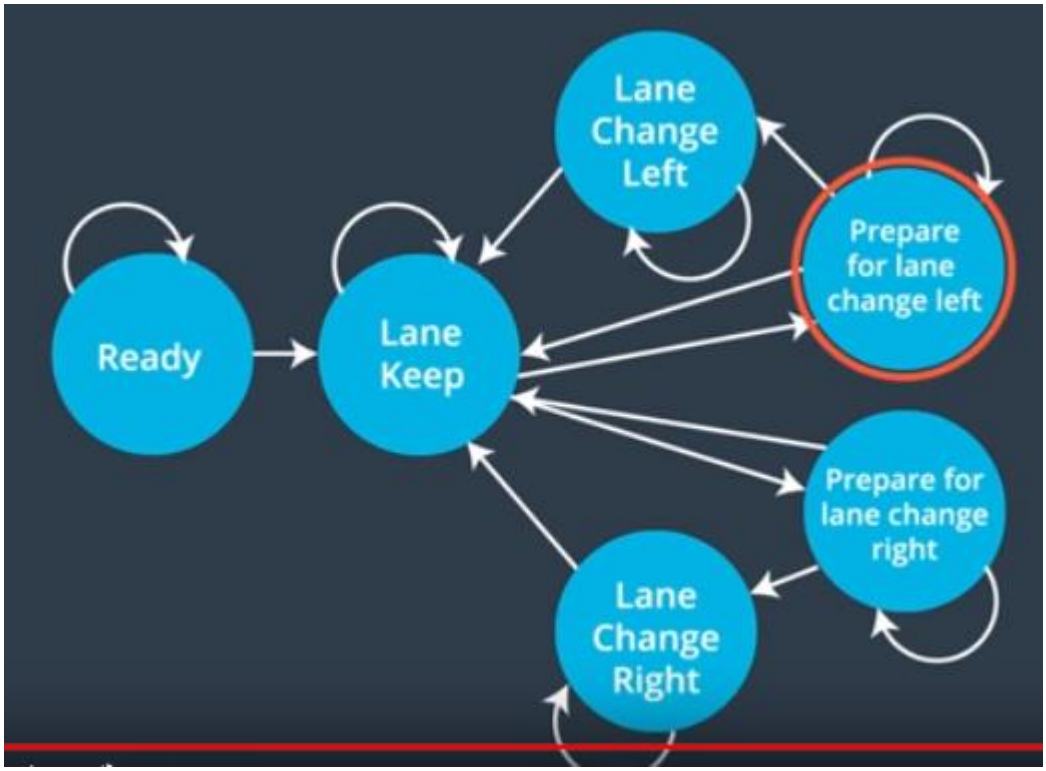
2. Update our vehicle

we mean by that to update the location of our vehicle. This step is a straight forward step, we use the function PathPlanningManage::update_auto_car_state to update the state of our car using the data received from the simulator.

3. Path Generator

`Based on the predictions and where the autonomous vehicle is allocated generate the new path. In our project we use the finite state machine explained in the lecture. So our model has six states(Ready, Keep lane, Prepare lane change right, prepare lane change left, change lane right, change lane next). Check PlannerFSM it contains all work related to FSM. We start with Ready state and then move in the next cycle to keep`

lane then at each cycle we check the cost of the possible states after the explained FSM in the lesson and we decide after the cost between all possible states.



To calculate the cost for each state we use the following cost functions:

1. Change lane cost: as mentioned before we should stay in same lane while there's no reason to change lane so we have added some cost here to avoid changing lane without real reason
2. Buffer cost: this cost will penalize the current lane if there's a vehicle in the same lane.
3. Inefficiency cost: It penalize the current lane it move slow so we prefer to move to another lane that move faster.
4. Free line cost: Penalize the trajectory that contains obstacles

Once we are able to know which state is better we realize the achieved state using PlannerFSM:realize_state and after that we generate the trajectory using trajectory::generate_trajectory.