Assignment 8

Trajectory Generation (Option 1)

1) In the frenet coordinate space, the *s* coordinates represent direction tangential to the vehicle on the road and *d* represents the normal direction to the vehicle. The lane markings are at d = 0, 4, 8, 12. By setting the target *d* of the next waypoints to the center of the lane (2/6/10), the car always follows the lane.
2) The spline function fits a smooth line over a set of sparse discrete waypoints. In order to generate a smooth trajectory, we always add the car's previous position, current position and 3 future positions (s+30/60/90) as the sparse way points for the smooth curve. We set the destination waypoint (distance horizon) to be s+30. The original heading is considered as x. We then calculate the y coordinates for n equally spaced along x. Each point corresponds to 0.02s of travel. Target distance *d* is calculated using ($target\_x^2 + target\_y^2$). The number of samples is computed using the target distance, *ref_vel* (reference velocity) and 0.02s. We can now calculate the y coordinates, rotate the points back to normal based of yaw and generate a smooth trajectory for the ego vehicle to follow.
3) We use the sensor fusion data to detect cars around the autonomous vehicle. We consider cars whose d position is within the current lane. After calculating the cars' future position (at the end of trajectory), we check if it is within 30m of the autonomous car's future position. If this is true, the points in the trajectory are generated closer together to slow down the vehicle (done by decreasing ref_vel).
4) In order to avoid a cold start, we do two things. We set the initial *ref_vel* to 0 and gradually increment the *ref_vel* by 0.224 (corresponds to ~5m/s^2 acceleration) if it is less than 49.5mph. This is done so the car does not instantaneously jump to 49.5mph. This is similar to slow down mechanism used to slow down the car when a collision is likely.

Behaviour Planning

In order to perform a lane change, we first need to determine if a lane change is required and if it is possible. This is done using the sensor fusion data of cars around the ego vehicle. If the ego vehicle is *too_close* (i.e. < 30m) from the vehicle in front of it. If this reduces 25m the car begins to slow down. Along with looking at the current lane, we also check the left adjoining lane and right adjoining lane. If there is car in the adjoining lane within *car_s*-8m < *car_s* < *car_s* + 35m, the lane is disqualified from being a potential lane to switch to. This is saved in the *too_close_<left/right>* variables. If the there's a slow car ahead of us, and at least one of the adjoining lanes are available to lane change, the ego vehicle decides to change lanes. We use a finite state machine with 2 states (STAY_LANE and TRANSITION_LANE). The state is changed to TRANSITION_LANE and the target *d* value is set to the center of the target lane. The spline function then generates a smooth trajectory to make the lane transition. The state is changed back to STAY_LANE when the *d* value of the ego car is within +/-1 of the center of the target lane. In order to keep the acceleration and jerk low, the car does not attempt to transition to a new lane when it is currently changing lanes. In the case where there are two available lanes to switch between, the ego car switches to the lane which is emptier (i.e. the nearest car is farther away). Additionally, in order to avoid getting stuck behind and beside a slow vehicle, the ego has an affinity to the center lane where it has two lane choices to switch to. If the car switches to either lane 0 or 2, it will attempt to switch back to lane 1 as soon as possible. To minimize the number of lane switches, it first checks if there is no car within a 100m of the ego car in the center lane and only then switches back. I also implemented a more gradual braking profile which allows the ego car to sneak closer to the vehicle in front it in order to perform a lane change. The ego takes around 2m20s to complete 2 miles with lane transitions.