CarND-Path-Planning-Project

Goal: to safely navigate around a virtual highway with other traffic that is driving +-10 MPH of the 50 MPH speed limit

I create a C++ class Planner to generate vectors of (next_x_vals, next_y_vals) for simulator to drive the car. Below shows code in the main message handling routine from socket.

```
// Create vectors of (next_x_vals, next_y_vals)
vector<double> next_x_vals;
vector<double> next_y_vals;

 // Use planner to figure next points
Planner myPlanner(previous_path_x, previous_path_y, car_x, car_y, car_s, car_yaw);
myPlanner.get_next_points(map_waypoints_x, map_waypoints_y, map_waypoints_s,  end_path_s, sensor_fusion, next_x_vals, next_y_vals);

// Send (next_x_vals, next_y_vals) to simulator
msgJson["next_x"] = next_x_vals;
msgJson["next_y"] = next_y_vals;
```

C++ class Planner
Public Methods:
"**get_frenet_points**" is a test function using Frenet-coordinate and map waypoints to calculate next points in global coordinate. It takes map waypoints (x,y) and s as input parameters. (next_x_vals, next_y_vals) are input and output parameters in global coordinate. (next_x_vals, next_y_vals) are used by simulator to navigate the car.

void **get_frenet_points**(vector<double>& map_waypoints_x, vector<double>& map_waypoints_y, vector<double>& map_waypoints_s, vector<double>& next_x_vals, vector<double>& next_y_vals)

"**get_next_points**" is the primary function of the class to get planned next 50 points for simulator to drive the car. It has the same input and output parameters as get_frent_points(). It uses private functions to smooth navigation and change lane.

void **get_next_points**(vector<double>& map_waypoints_x, vector<double>& map_waypoints_y, vector<double>& map_waypoints_s, double end_path_s, vector<vector<double>>& sensor_fusion, vector<double>& next_x_vals, vector<double>& next_y_vals)

map_waypoints_x, map_waypoints_y, map_waypoints_s: map waypoints (x,y) and s, input only
end_path_s: s value of the end path, input only
sensor_fusion: data of all cars on the road close to our self-driving car, input only
next_x_vals, next_y_vals: (x,y) values in global coordinate, input and output

Private Methods:
"**add_prev_pts**" is a utility function to add previous points into (next_x_vals, next_y_vals). It also adds initial points to (x_vals, y_vals) from previous points to smooth the path and updates reference values of x, y, and yaw angle.

void **add_prev_pts**(vector<double>& x_vals, vector<double>& y_vals, double& x_ref, double& y_ref, double& yaw_ref, vector<double>& next_x_vals, vector<double>& next_y_vals)

x_vals, y_vals: vector of (x,y) for planned path, input and output
x_ref, y_ref, yaw_ref: reference values of (x,y) and yaw angle. Initially, the reference (x,y) is location of the car, input and output
next_x_vals, next_y_vals: vector of (x,y) for simulator to navigate the car, input and output

"**check_car_ahead"** is a utility to check whether there is a safe distance from the front car in the same lane ahead, returning true if too close.

bool **check_car_ahead**(double end_path_s, vector<vector<double>>& sensor_fusion, int sample_count)

end_path_s: s value of the car or the end of prior path, input
sensor_fusion: vector of vector of double, data if all cars in the road, input
sample_count: number of previous points to calculate distance ahead or behind, input

“**change_lane”** is a utility function to change lane if feasible. It uses **is_lane_safe** to decide a particular lane is safe to change to. Here I uses a simply algorithm by checking left lane first and then right lane. Normally, we want to pass a car by the left lane.

bool **change_lane**(double end_path_s, vector<vector<double>>& sensor_fusion, int sample_count)

end_path_s: s value of the car or the end of prior path, input
sensor_fusion: vector of vector of double, data if all cars in the road, input
sample_count: number of previous points to calculate distance ahead or behind, input

“**is_lane_safe”** is a utility function to change a particular lane is safe to change to for cars in front and behind.
bool **is_lane_safe**(int idx, double end_path_s, vector<vector<double>>& sensor_fusion, int sample_count)

idx: index of the lane to change to. It starts zero from the leftmost lane next to double yellow lines and increases one by one to right, input only.
end_path_s: s value of the car or the end of prior path, input only
sensor_fusion: vector of vector of double, data if all cars in the road, input
sample_count: number of previous points to calculate distance ahead or behind, input only

“**add_way_points”** is a utility function to add waypoints into (x_vals, y_vals).

void **add_way_points**(vector<double>& x_vals, vector<double>& y_vals, vector<double>& map_waypoints_x, vector<double>& map_waypoints_y, vector<double>& map_waypoints_s)

x_vals, y_vals: vector of (x,y) for planned path, input and output
map_waypoints_x, map_waypoints_y, map_waypoints_s: (x,y) and s of map waypoints, input only

“**add_addl_points_to_total**” is a utility function to add more points using spline to make total of 50 for simulator.

void **add_addl_points_to_total**(vector<double>& next_x_vals, vector<double>& next_y_vals, double x_ref, double y_ref, double yaw_ref, int count)

x_vals, y_vals: vector of (x,y) for planned path, input and output
x_ref, y_ref, yaw_ref: reference (x,y) and yaw to transform and translate to global coordinate, input only

"**transform_to_car_coord" is a** utility to perform translation and rotation from global coordinate to car coordinate.

void **transform_to_car_coord**(vector<double>& x_vals, vector<double>& y_vals, double x_org, double y_org, double yaw)

x_vals, y_vals: vector of (x,y) for planned path, input and output
x_org, y_org, yaw: reference (x,y) and yaw to transform and translate to car coordinate, input only

Private Variables:
There are two internal states, m_lane for lane number and m_ref_vel for velocity in miles/hour.

```
static int m_lane; // lane number starting from double yellow lines as zero
static double m_ref_vel; // reference velocity miles/hour
```

I also keep some states from simulator.

```
vector<double>& m_previous_path_x;
vector<double>& m_previous_path_y;
double m_car_x;
double m_car_y;
double m_car_s;
double m_car_yaw;

A state variable spline is used across private functions.
tk::spline m_spline;
```

Future Work

I use simple algorithm for lane pass and state management. It needs more sophisticate algorithm to handle corner cases with smoothness in real world. I also have some jerks during lane change, which can be improved.