Path Planning

The path planning describes the scenario for highways with 3 lanes. In comparison to urban traffic, the scenario is simpler, since on the 3 lanes, only other vehicles with different velocities operate. There are no crossing cars or intersections, which needs a prediction trajectory for other vehicles. The task is to accomplish around five kilometres of driving without any incident. Possible incidents are collisions with other cars, leaving the lane, too high velocity and acceleration. The problem of path finding can be split into two subproblems. One is **behaviour planning**, which steers the vehicle by changing lanes and changing the vehicle speed. The **trajectory calculation** translates those behavioural planning into an trajectory, which in the end our ego-vehicle will follow.

While trying to solve this challenge, I follow the proposed code from the Q&A. Especially for the trajectory calculation, I tried to come up with my own solution, but failed to reach the quality of the original proposal. In my tries, I constantly violated the acceleration requirements during lane changes by maximum allowed speed. I didn't add my trials in the submitted code, but can provide it, if requested.

1. Behaviour Planning

1.1 Implementation (code lines: 259 - 349)

My approach for behaviour planning is quite simple. I use basic hard-coded logic for the behavioural planning, instead of a machine learning model. This approach is still okay in this highway scenario, since the use-case is quite simple. For a more complex logic like driving in a crowded urban space, this hard-coded logic won't be maintainable, because of complexity.

In the highway scenario, there are only **four possible decisions** (change lane, accelerate, brake, do nothing), which the car has to decide. Those four decisions are implemented with six boolean flags:

-free_ride:	Ego-vehicle has a free lane
-left_lane_free:	Left lane is free
-right_lane_free:	Right lane is free
-center_lane_free:	Centre lane is free
-vehicle_in_front:	A vehicle is ahead of the ego-vehicle in any lane
-vehicle_inside_distance:	A vehicle is inside the defined security_distance in any lane

The logic is quite simple, if a car is too close, the speed will be reduced by 0.224 m/s. With a time increment of 0.2 seconds, this is an acceleration around 1 m/s*s. To make sure, the ego-vehicle reduces the speed as much as needed, it will adapt to the speed of the vehicle in front. Also, when the ego-vehicle has to break, it will check, if other lanes are free. If this is the case, it will, based on the own lane position, go to the left lane or to the right lane. Also the current implementation allows to change two lanes, if necessary. If the ego-vehicle has a free lane, it will accelerate around 1 m/s*s until it reaches the maximum speed limit.

1.2 Observed problems

The observed problems aren't critical. Sometimes the ego-vehicle changes to the dense lane, when it drives in the centre. This problem occurs, that the security distance is the only distance measurement to other vehicles. This issue can be solved with another distance, but also this will put a higher complexity to the code, which isn't beneficial in this task.

2. Trajectory Calculation

2.1 Implementation (code lines: 351 - 471)

From the behaviour planning, the program gets the current velocity of the ego_vehicle and the proposed lane. With this information, the trajectory had to be calculated. Since the velocity of the vehicle is depended on the distance of each point inside the trajectory, there is some difficulty in the accurate prediction of the future point coordinates. If the point coordinates are too far away, the velocity of the ego-vehicle will be higher than the speed limit. The point, where I gave up my solution, was the increased distance between points during lane changes. Here, I wasn't violating the speed limit, but I failed constantly to stay inside the acceleration limits. I tried to control it with some introduced penalty for the coordinates prediction, but it failed, when the car was changing two lanes at once. Here I followed the proposed solution from the Q&A, which is also not perfect, but it fails rarely and is better than my approach.

2.2 Spline Regression

A trajectory for the ego-vehicle consists of 50 points. If the ego-vehicle passes one of those "waypoints", it will be removed from the path and new predicted waypoints will be added. As a first step, the coordinates of the future trajectory points had to be predicted. Those waypoints will be predicted by an **unsupervised spline regression model** as proposed in the problem introduction. A Spline regression basically fits a smooth function on a given dataset. This smooth function is used to predict the curves on the highway and enables a smooth transition for lane changes. For the dataset, I use five data points, based on two historic waypoints and three predicted waypoints. Those predicted waypoints can be easily calculated with the Frenet coordinates and will be transformed to cartesian coordinates. The three predicted data point for model training will have a distance between each other by 30 m. My experience here is, that if the distance is higher than 30 m, the predicted y-coordinates will follow a wavy line, which isn't intended (also I'm not happy with 30 m either). If I use a smaller distance, then the gradient for changing lanes will be too steep, so that I violate the acceleration conditions. So the current 30 m distance was the best trade-off, which I could figure out. Also the model could be improved by more data points, which could be a future task.

2.3 Trajectory Calculation

With the fitted spline regression model, the y-coordinates for the trajectory points can be predicted. On the first step a total distance for the ego-vehicle horizon will be calculated based on the number of trajectory points, the time delta and the speed limit. Based on the total distance, a **distance increment** for each trajectory point will be calculated based on the current ego-vehicle speed. This makes sure, that the ego-vehicle won't excel the speed limit. For each trajectory point the coordinates can be calculated. Since most of the trajectory points were calculated before, this method updates the current trajectory to the amount of the defined points.

2.4. Observed problems

The Spline regression with five points is sometimes to coarse. There is a notion for wavy lines and straight lines, which push the ego-vehicle outside the lane for a short moment. This can be probably solved with more training data. Also the spline regression fails, if two values are equal. I changed the code inside the spline.h file to accept also equal values (code line 294).
Another problem lies in my current implementation, which is the proposed solution form the Q&A. The current implementation only works with 50 trajectory points and a time delta of 0.02 s. If those values are changed, the ego-vehicle will drive too slow or collides with other cars, since there is a strange influence on the security margin. This would afford more debugging from my site in the future.