Model Predictive Control for Autonomous Driving:

In this project a model predictive controller is designed for the car that will make the car able to autonomously drive along a path in the simulator.

For this project a global kinematic model is used to be able to predict the state of the car in a given future time. The state in this project is defined with 6 variables. The current x position of the car in the X-Y plane in time t, denoted as x_t. The current y position of the car at time t denoted as y_t. The speed of the car at time t denoted as v_t, the steering angle of the car with reference to the X axis at time t denoted as ψ_t and the 2 errors used in the project namely the cross track error at time t denoted as cte_t , which is the distance of the car to the middle of the lane line and also the orientation error at time t which is denoted as $e\psi_t$, which is the difference between the current steering angle of the car and the desired angle of steering for the car. This 6 variable constitute the states of the model. For actuation there are 2 actuators: the change in steering angle and the change in throttle. The first actuator is basically the wheel of the car and rotating the wheel to make the car co in different directions. The second actuator makes the car go faster or slower and is used to accelerate and decelerate the car for breaking. There are constraints for the actuators, the throttle value is between -1 and 1 and the steering angle can change only 25 degrees in both direction. This makes our model a non-holonomic model meaning that the actuators can't change as they want but have some limits imposed on them. For predicting the state t+1 based on the values of the state in time t, the update step of the model, the following equations are used:

$$x_{t+1}=x_t+v_t\cos(\psi_t)*d_t$$
$$y_{t+1}=y_t+v_t\sin(\psi_t)*d_t$$
$$\psi_{t+1}=\psi_t- v_tLf\delta_t*d_t$$
$$v_{t+1}=v_t+a_t*d_t$$
$$cte_{t+1}=y_t-f(x_t)+(v_t*\sin(e\psi_t)*d_t)$$
$$e\psi_{t+1}=e\psi_t+Lfv_t*\delta_t*d_t$$

In the previous equations, δ_t is the steering angle of the car at time t, a_t is the acceleration at time t and Lf is the measures the distance between the center of mass of the vehicle and it's front axle. The larger the vehicle, the slower the turn rate.
Also for the desired values, the desired cross track error and orientation error should be zero and the desired speed, should be as high as possible and I decided for the value 100 mph for the desired speed.

For choosing the N and d_t values, the first consideration was to decide about the prediction horizon value, called T. After that based on the formula T = N * d_t deciding for N and d_t will be much easier. I decided to use a prediction horizon of 1 second. This decision seems plausible for the speed of 100 mph, because the car will travel a huge distance if the horizon is higher and hence the prediction becomes more inaccurate. N is the number of steps in the in the prediction horizon and d_t is the duration of each step. The larger the value of N is the larger the computational load becomes and hence the longer it takes to get a response. For this reason we want to keep N as small as possible while also big enough that each step is reasonable. I tried some values above 10 and decided that N = 10 is a good value for this tradeoff and d_t becomes 0.1 seconds which is also a good value for each steps prediction.

After getting the waypoints from the simulator, a transformation is done on the points. This transformation is basically here to shift the car in the (0,0) point and also make ψ = 0 such that the calculation of the cross track error and orientation error becomes much easier. I have seen this transformation done in the Q and A video from Udacity and got the idea from there as well, but it really helped me to understand the problem better and make simplifications specifically for the calculation of latency it helped to reduce the formulas a lot. The code used here is as follows:

```
for (int i = 0; i < ptsx.size(); i++)
{
    double shift_x = ptsx[i] - px;
    double shift_y = ptsy[i] - py;

    ptsx[i] = (shift_x * cos(0-psi) - shift_y * sin(0-psi));
    ptsy[i] = (shift_x * sin(0-psi) + shift_y * cos(0-psi));
}
```

So we subtract each waypoint from the current position of the car and rotate the point psi degrees such to make the current psi of the car 0.

For handling the latency of the simulator what is done is that we can use the predictive model to predict where the car might be in the time after delay and then feed the model predictive controller with these new predicted values. The way this is done is that the update equations above are used with the value of 100 milliseconds for d_t and for the acceleration and steering angle values the values obtained from the simulator through the following code is used:

double steer_value = j[1]["steering_angle"];
double throttle_value = j[1]["throttle"];

After that the location of the car is updated for 100 milliseconds and then fed to the controller. This handles the latency of the simulator and the car can drive one lap, at least, in the simulator.