Model Predictive Controller

The Model:

In this project the model was a simplification of dynamic model that ignores gravitational forces, mass and frictional forces of tire. The model used here is a **kinematic model.**

The objective is to find the best suitable values of acceleration and steering of the car to complete the given track without swirling away from the road.

The model calculates the current state of the car based off the actuation values from the previous time step. The equations below are the constraints used to build up the model.

All the other constraints based on the vehicle model:

$$x_{t+1} = x_t + v_t cos(\psi_t) * dt$$

$$y_{t+1} = y_t + v_t sin(\psi_t) * dt$$

$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f}\delta_t * dt$$

$$v_{t+1} = v_t + a_t * dt$$

$$cte_{t+1} = f(x_t) - y_t + (v_t sin(e\psi_t)dt)$$

$$e\psi_{t+1} = \psi_t - \psi des_t + (\frac{v_t}{L_f}\delta_t dt)$$

Where : x and y are Car's Position
psi is the Car's Orientation
v is the Car's Velocity
cte is Cross-Track Error
epsi is the Orientation error
a is the acceleration
delta is the steering angle.

Timestep Length and Elapsed Duration (N & dt):

The time step N is 10 and dt = 0.1 was used. After multiple trials this turned out to be the best fit any other values turned out to be erroneous. The values were tried in the range from N = 10 to N = 20 and dt was varied from 0.2 to 0.01. Looks like Udacity suggested the best value for us to start from.

Polynomial Fitting and MPC Preprocessing:

The waypoints provided are not in the car's coordinate system so they have to be transformed and line 99 - 108 do that and the transformation is as shown below.

```
for(int i = 0; i <ptsx.size(); i++){

    //subtract all the points from current position this making x & y 0
    double shift_x = ptsx[i]-px;
    double shift_y = ptsy[i]-py;

    //making psi zero by rotating our points
    ptsx[i] = (shift_x*cos(-psi)-shift_y*sin(-psi));
    ptsy[i] = (shift_x*sin(-psi)+shift_y*cos(-psi));
}
```

Where the number of waypoints are given by pts.size() and then transformation of the points to car's coordinate system is done for all the waypoints. These values will be used later on to calculate the values of Cross-Track Error and Orientation error.

Model Predictive Control with Latency:

To handle latency of 100 ms the values are initialized and a prediction for the state is done by the code as shown below:

```
118             double Lf = 2.67;
119     //          double cte = polyeval(coeffs,0);
120             double steer_value = j[1]["steering_angle"];
121             double throttle_value = j[1]["throttle"];
122
123             // predict state in 100ms
124             double latency = 0.1;
125             px = v*latency;
126             py = 0;
127             psi = -v*steer_value*latency/Lf;
128             double epsi = -atan(coeffs[1]);
129             double cte = polyeval(coeffs,0)+v*sin(epsi)*latency;
130             v+= throttle_value*latency;
131             Eigen::VectorXd state(6);
132             state << px, py, psi, v, cte, epsi;
```

This helps in the computation of the state vector. The state vector is then passed to MPC::Solve. The way this latency is handled is it predicts future position of the car based of the estimations in the current state and expects the car to be in the estimated state for a brief period till the actuations happen. This helped me with steep corners where without this logic the car would steer off the driving track in steep turns.

Using helped me get through the corners easily.

```
fg[0] += CppAD::pow(vars[delta_start + t] * vars[v_start+t], 2);
```

Final Video:

The final video of the car can be found here (https://github.com/rakshithkeegadi/CarND-MPC-Project/blob/master/final_video_MPC.flv)