## The Model:

I have used kinematic model specified in the lectures. This model is implemented in MPC.cpp file; the state equations are as below

```
  fg[2 + x_start + t] = x1 - (x0 + v0 * CppAD::cos(psi0) * dt);
  fg[2 + y_start + t] = y1 - (y0 + v0 * CppAD::sin(psi0) * dt);
  fg[2 + psi_start + t] = psi1 - (psi0 + (v0 / Lf) * delta0 * dt);
  fg[2 + v_start + t] = v1 -(v0 + a0 * dt);
  fg[2 + cte_start + t] = cte1 -(f0 - y0 + v0 * CppAD::sin(psi0) * dt);
  fg[2 + epsi_start + t] = epsi1 - (psi0 - psides0 + (v0 / Lf) * delta0 * dt);
```

States include the x, y position of vehicle, orientation, velocity, crosstrack error and orientation error.

## Time step Length and Elapsed Duration (N & dt):

MPC predicts the N steps into future with a time difference of dt. I started with 20 steps and dt = 0.05, this setting had an issue to drive the car successfully at the curves, and vehicle keeps going off the track at the turns, then with a series of trail and errors I settled with dt = 0.1 and N = 10 which works fine with other parameters selected.

## Polynomial Fitting and MPC Preprocessing:

I have converted given reference lane points into vehicle coordinate system to be able to plot them in the simulator. And also used the same converted points to fit a 3rd order polynomial (which could best represent the curves in the road)

```
auto coeffs =  polyfit(ptsx_eigen, ptsy_eigen, 3);
```

In MPC, I'm using IPOPT to arrive at optimized path calculator which reduces the crosstrack error and orientation error. Different weights are applied to different costs, as seen below code:

```
 for (int t = 0; t < N; t++) {
   fg[0] += 300*CppAD::pow(vars[cte_start + t], 2);
   fg[0] += 300*CppAD::pow(vars[epsi_start + t], 2);
   //TODO: Check if we need to use velocity
   fg[0] += 0.1*CppAD::pow(vars[v_start + t] - ref_v, 2);
 }

 // Minimize the use of actuators.
 for (int t = 0; t < N - 1; t++) {
   fg[0] += CppAD::pow(vars[delta_start + t], 2);
   fg[0] += CppAD::pow(vars[a_start + t], 2);
 }

 // Minimize the value gap between sequential actuations.
 for (int t = 0; t < N - 2; t++) {
   fg[0] += 500 * CppAD::pow(vars[delta_start + t + 1] - vars[delta_start + t], 2);
   fg[0] += CppAD::pow(vars[a_start + t + 1] - vars[a_start + t], 2);
 }
```

a high weight of 300 applied to crosstrack error and heading error because we want to minimize this as much as possible so that vehicle can drive in the center. Also a weight of 500 is applied to make sure that there are no abrupt steering changes applied.

## Model Predictive Control with Latency:

Latency was incorporated into system by predicting the state of car including latency. Once this is incorporated, MPC will calculate the state after the latency. This calculation is done in the main.cpp as below:

```
  //Consider the latency
double dt = 0.1;
double px_car = v * dt;//dt = 0.1
psi = -v * steer_angle * dt / 2.67;
```

These values are included in state vector and are passed to MPC.