**MPC Control:**

As I understand it based on the course and extra reading, MPC is a form of control where there is some comprehension of future events whereas the traditional PID control does not have this ability.

This is done by modeling the system a accurately as possible to predict the future states based on current states and solving for optimal control inputs to achieve the target and then doing it all over again in the next cycle. While optimizing the control inputs a set of constraints are applied that reflect the system limitations.

**Udacity Project outline:**

Aim of this project is to write a C++ program that can drive the simulated car around a track using telemetry data fed to the program via the simulation. One of the main aspects of the program is to consider the 100 ms latency that is seen in real life when there is a delay between accelerator input and corresponding vehicle response.

**Approach:**

Here I followed mostly what was taught in the class:

1. Simulator inputs:
    a. x, y global coordinates representing the waypoints ahead of the vehicle,
    b. current position of the vehicle in x, y
    c. current orientation of the vehicle in global coordinates
    d. current velocity of the vehicle
    e. delta which is the current steering angle
    f. current acceleration
2. Polynomial fitting:

    Here a 3$^{rd}$ degree polynomial is fitted using the provided polyfit function to create a function that best fits a curve to the set of points provided in 1a above. Before doing this, the coordinates are transformed to vehicle frame of reference by making vehicle position as origin.

    The value of this function at vehicle location (x,y are 0) is zero. This is the cross track error.

    The derivative of the function is its slope at the given position. So at origin the only the first order coefficient remains and it is the dpsi (psi error).

3. Latency:

    Assuming origin as the vehicle, a set of simple state prediction equations are used to predict where the vehicle will be after the delay period (100 ms). (Main.cpp lines 155-160)

    These state parameters are then passed to the MPC solver.

4. MPC Solver:
    a. First, we set all the state variable and constraint arrays by sizing them based on number of actuators, timesteps and state vector length. All independent variables are set to zero. The lower and upper bounds are set for actuators along with other variables. Here I used values provided in the Udacity classes. These constraints will later be passed to the ipopt solver.
    b. Also passed to the ipopt solver is the cost function for each variable (FG_eval class). One thing I spent some time tuning is assigning weights to each cost. Cte and epsi are weighted heavily in order to keep the vehicle centered on the road. But v is weighted less as maintaining the cte is more important. Similarly, the changes to delta and a are weighted heavily to enable smooth operation of the car.
    c. A set of constraints are calculated using the same state equations used in step 3.
    d. Finally the variables, upper, lower bounds, the cost functions are passed to the ipopt solver which provides us with a solution with calculated future states including the a(throttle). This value is used as the actuator value.

5. Tuning the Timesteps (N) and timestep duration:

    Initially I started with 15 for N and 0.2 for duration as for any given speed I thought 3 seconds of future prediction will be enough. But it turned out that the vehicle was very lurchy and started going off track immediately. At N=15, the model performed slow, so I reduced that to 10 and in order to improve the resolution I changed dt to 0.1 sec. This combination worked for me.