MPC == > Model predictive control

The models used in MPC are generally intended to represent the behavior of complex dynamical systems.

MPC uses the current plant measurements, the current dynamic state of the process, the MPC models, and the process variable targets and limits to calculate future changes in the dependent variables. These changes are calculated to hold the dependent variables close to target while honoring constraints on both independent and dependent variables. The MPC typically sends out only the first change in each independent variable to be implemented, and repeats the calculation when the next change is required.

## Project Steps

- Fitting a line based on road waypoints and evaluating the current state based on that polynomial line.
- Implementing the MPC calculation, including setting variables and constraints
- Calculating actuator values from the MPC calc based on current state
- Accounting for latency
- Calculating steering angle & throttle/brake based on the actuator values
- Testing/tuning of above implementations on Udacity simulator

State vector :
x,y: Car position
psi: Car heading direction
v: Velocity
cte: cross track error
epsi: error orientation

### Polynomial Fitting & Preprocessing

The points from the simulator's global coordinates are transformed into the vehicle's coordinates.

A third-degree polynomial line is fit to these transformed waypoints, essentially drawing the path the vehicle should try to travel. Moving on further is where the transformations are critical - because we are operating from the vehicle's coordinates, from the vehicle's standpoint, it is the center of the coordinate system, and it is always pointing to a zero orientation. The cross-track error can then be calculated by evaluating the polynomial  at px (which in this case is now zero, so technically could also just be calculated as the first coefficient value - i.e. the one with a zero-order x). The psi error, or epsi, which is calculated from the derivative of polynomial fit line, is therefore simpler to calculate, as polynomials above the first order in the original equation are all eliminated through multiplication by zero (since x is zero). It is the negative arc tangent of the second coefficient (the first-order x was in the original polynomial).

The model combines the state and actuations from the previous timestep to calculate the state for the current timestep based on the equations below:

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

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

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

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

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

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

**Tuning Timesteps (N) and Timestep Duration (dt)**

Given the reference trajectory from polynomial fit of waypoints and the motion model of the car, MPC estimates the value of actuator inputs for current time step and few time steps later. This estimate is used to predict the actuator inputs to the car ahead of time. This process of estimate generation is tunable with the use of N and dt. Higher value of N ensures more number of estimates while higher value of dt ensures the estimates are closer in time. Different combinations of values of N and dt were tried and following were the observations:
I originally was using values of 15 for N and 0.2 for dt, because I thought 3 second (15 x 0.2 seconds) would be a good prediction span, and I also had thought I could account for latency in this way. I found the model seemed to slow down if N was higher, so I eventually settled on 10 for N, which meant that with 0.1 dt, I was only predicting for one second essentially.

**Latency Handling :**

There was a latency of 100 ms introduced , to handle this , the actual data were shifted 100 ms before passing it to the MPC module "state vector " to help reduce the effect of the latency.