In the Model the state is contained in a vector which has the car's *x* coordinate, *y* coordinate, orientation angle *psi,* velocity *v*, the cross track error *cte*, and the psi error *epsi*. The actuators are contained in a vector which has acceleration *a*, and steering angle *delta*. The model uses the actuations and state from the previous timestamp to calculate the current state according to the following equations:

- 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] + v[t] / Lf * delta[t] * dt
- v_[t+1] = v[t] + a[t] * dt
- cte[t+1] = f(x[t]) - y[t] + v[t] * sin(epsi[t]) * dt
- epsi[t+1] = psi[t] - psides[t] + v[t] * delta[t] / Lf * dt

For N I have choosen 10, and for dt 0.1 These are the same values as in the lessons. I tried many other values for the amount of steps (N), and for the duration of the timestep (0.1) but it seemed that the model worked best with these values (other values had the car swaying, and sometimes crashing).

Other values tried; N= 40, dt= .4; N = 30, dt =.3; N=20, dt =.2; N=5, dt =.05

I have set the way-points to the car's coordinate system in main.cpp (from line 108), so the vehicle would be at the origin, with an orientation angle of 0. This so it would be easier to fit a polynomial (of the third degree) to the waypoints.

To combat latency I used the actuations from two timesteps previous (one to comply with the equations, and one to combat latency) as the delay was equal to one timestep (100ms) (MPC.cpp line 94-96).