Model

The model was as given in the lessons from the Global Kinematic Model Lesson

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

$$y_{t+1} = y_t + v_t * sin(\psi_t) * dt \quad (2)$$

$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} * \delta * dt \quad (3)$$

$$v_{t+1} = v_t + a_t * dt \quad (4)$$

Figure 1. Equations for updating states

- State – States that were used were:
  - x and y -> the vehicle's position
  - Psi -> the vehicle's trajectory. In this case it was the angle between the local coordinate frame of the vehicle and the global coordinate system
  - V -> the vehicle's velocity in mph
- Actuators – Actuators are
  - a – acceleration that covers both the acceleration and braking. In this case the range of a is from -1 to 1. -1 to 0 denotes braking while values over 0 up to 1 represent acceleration
- Update Equations
  - The update equations are shown in the figure above. The position states are updated using the velocity and the trajectory multiplied by dt to update the current values of position, x and y. Psi is updated using the vehicles length from the center of gravity to the front of the vehicle, Lf multiplied by the steering input delta and the time step. The second term in the third equation denotes the resulting angular velocity as steering input is given. Finally velocity is updated with the acceleration of the vehicle at.

Timestep Length and Elapsed Duration

- N – I kept N at 25 steps as I was basing my decision of how to choose the duration on the length of the green line in the simulator and how it traced the waypoints. I felt 25 timesteps was an appropriate starting point and wanted to use the time step to adjust duration
- dt – I adjusted dt by using the green line projection as mentioned above, visually estimating what an appropriate time step should be. I knew that the total distance ahead of the vehicle through which the green line was projected was driven by the speed as well as the dt term. Therefore, as I increased the speed at which the vehicle would run, I would adjust dt to get the

appropriate duration or length ahead of the vehicle as shown by the green line. The green and yellow lines can be seen in the snip of the video below

Figure 2. Snip of video. Green shows approximate duration of the simulation while yellow ilustrates the way points

Polynomial Fitting and MPC PreProcessing

- Polynomial - As shown in the classroom notes, a third order polynomial was used to fit a curve through the waypoints. However, before fitting the curve, the waypoints were converted to local car coordinates as illustrated by the equations from a discussion forum on Udacity. The reason for the transformation is so that the cross-track error (cte) and heading error (espi) could more easily calculated. cte in local coordinate simply becomes the polynomial evaluated at x = 0, while epsi is simply the atan(first derivative of polynomial @ x=0). These calculations can be seen in lines 139 and 141 of main.cpp

```
x_car_space(i) = (ptsx[i] - px) * cos(psi) + (ptsy[i] - py) * sin(psi) ;
y_car_space(i) = (ptsy[i] - py) * cos(psi) - (ptsx[i] - px) * sin(psi) ;
```

-

Figure 3. Equations for transforming global to local coordinates

- MPC procedure. In MPC.cpp the MPC procedure is carried out in a class under the Solve method. This method was developed based on the starter code and the solution set provided in the course. MPC::Solve takes as inputs, the states and the fitted polynomial. MPC starts with assignment of states to appropriate variables, defining a vars vector that would hold the state values as required for the solver, and initialing the vars vector appropriately. The vars vector essentially holds the x, y, psi, v, cte, and epsi (states and actuators) for each time step defined by dt and N. bounds are set on the vars vector based on the indices of the elements. Finally, fg_eval is the object that computes the objective and constraints of the MPC and takes as input the polynomial coefficients. The MPC class is defined in MPC.cpp starting at line 138.
- Inside the FG_eval class is the cost function, whose result is stored in the first element of the vector fg. The remaining elements of fg contain the constraints used to solve for the optimal control outputs using the iopt solver.
- In tuning the MPC weights were included into the cost function for tuning the output behavior. This can be seen in lines 60 through 82 of MPC.cpp. The weights were tuned using trial and error. A few important weights that were tuned were for the cte and espi variables. These

determined how centered the car would be to the waypoints and how sensitive the vehicle would be to following the waypoints. The weights are on lines 62 and 64 respectively. The next weight that was of some importance was on line 71. This weight controlled the minimization of actuator inputs. As it turns out, I had to multiply the one for steer by a fraction in order to minimize the penalty on steering input in order to be able to turn sharp corners at higher speed. The last weight that I had to tune was on line 79. This controls how quickly the input for steering changed from one time step to another. I had to use a large value here so that I could use aggressive compensation for minimizing cte and epsi while making the driving smooth.

Model Predictive Control with Latency

The latency term was handled in main.cpp from lines 112 to 113. Although the latency was defined at 100ms, I actually used a value of 250ms in order to project vehicle a little further into the future. I noticed that increasing this value made it easier for me to tune the steering values at higher speeds. To handle the latency, I first chose to project the vehicle's future position in global coordinates. I wanted to do this because it was easier for me to visualize conceptually. Given the position x and y and the velocity I could predict the new global position of the vehicle. I chose not to include an updated psi angle as I was trying to use the vehicles steer angle in equation 3 of figure one but found that including the steer caused some instability in the prediction. I figured that the error by omitting the steer might be minimal given that for the most part the steer angle would be small to maintain the waypoint tracking.

After predicting the new x and y in global coordinates, I then transformed these coordinates into local vehicle coordinates (Lines 125 and 126). Here also, I chose not to use an update psi angle so I could rely on the more stable vehicle state, again figuring that the error in psi angle using this method might be minimized. This way, I didn't have to do any predictions in local coordinates and I could pass the states of 0, 0, 0 to x, y, and psi for the state (line 145). Also, cte and epsi were relatively easy to evaluate since the coefficients were in local coordinates vs. global.