The Model

A standard bicycle model is used to model the vehicle motion. The bicycle model utilizes the vehicle position, velocity, and orientation and incorporates actuators that can introduce an orientation change (δ) and a velocity change (*a*) in the following motion update equations:

$$x_{t+1} = x_t + v_t \, cos(\psi_t) \, dt$$
$$y_{t+1} = y_t + v_t \, sin(\psi_t) \, dt$$
$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} \, \delta_t \, dt$$
$$v_{t+1} = v_t + a_t \, dt$$

where L_f is the distance from the center of gravity to the front-end of the vehicle.

The cross-track error (*cte*) and the orientation error (*epsi*) are based on the desired state of the vehicle in the immediate future. They are updated with the following equations:

$$cte_{t+1} = f(x_t) \; - \; y_t + v_t \, sin(e\psi_t) \, dt$$
$$e\psi_{t+1} = \psi_t - \psi des_t \, + \frac{v_t}{L_f} \, \delta_t \, dt$$

where the function *f(x)* is the polynomial fit to the waypoints along the center of the track.
The state of the vehicle is then defined to be a 6-dimensional vector:

$$[x, \; y, \; \psi, \; v, \; cte, \; e\psi]$$

Timestep Length and Elapsed Duration

The vehicle state and the target trajectory must both be discretized by the chosen timestep length, and the amount of points is defined by the chosen elapsed duration. For this project, a timestep length of *dt = 0.1* seconds was chosen and an elapsed duration of 1 second was used. This results in *N = 10* discrete steps used to predict the vehicle state as the model looks 1 second into the future.

At first, the values used were *N = 25* and *dt = 0.05* since these were used in the lessons. Because of other issues, I used the video walkthrough provided by the course and tried to use the instructors used, which are the ones above. These parameters seemed to be successful.

Next, I wanted to see if decreasing *dt* from 0.1 to 0.05 would increase performance, since this would mean the controller would be more granular. However, this resulted in horrible performance. My conclusion was that because the value of *dt* was less than the latency, the latency compensation was invalid. Therefore, I bumped *dt* back to 0.1. I also tried decreasing computation cost by increasing *dt* to 0.2, but this resulted in a more conservative controller that drove very slowly and really hugged the inside of corners.

Polynomial Fitting and MPC Preprocessing

In order to simplify the calculation of the cross-track error, the waypoints are transformed from the global reference frame to the reference frame of the vehicle. A 3rd degree polynomial is then fit to the transformed way points.

In the vehicle frame, the vehicle is located at *(0, 0)* and has an orientation of *0* radians. The *cte* is then simply equal to the value of polynomial at *x* = 0, and the *epsi* value is simply equal to the negative of the slope of the tangent to the polynomial at *x = 0*.

Model Predictive Control with Latency

The latency between decision and actuation was handled by taking the current vehicle state, simulating the state change over the duration of the latency, and then using this updated state as the initial state.

The bicycle motion model was used to move the vehicle into the future considering the current throttle value and the current steering angle. The vehicle state *in the global frame* was being considered here, so while the same update equations are valid, there are a couple small changes.

$$x_{t+latency} = x_t + v_t \, cos(\psi_{t,global}) \, \Delta$$
$$y_{t+latency} = y_t + v_t \, sin(\psi_{t,global}) \, \Delta$$
$$\psi_{t+latency} = \psi_t - \frac{\delta_t \, \Delta}{L_f}$$
$$v_{t+latency} = v_t + a_t \, \Delta$$

where Δ = latency duration, *a* = throttle value, δ = steering value, and $\psi_{t,global}$ = ψ_t - δ_t
Is the updated vehicle orientation in the vehicle reference frame (keeping in mind that the simulator uses a left-handed coordinate system, so that we must subtract the steering value from the orientation angle).

The limitation of this latency compensation is that if *dt* is shorter than the latency duration, then an actuation is applied *before* the latency duration ends and the simulation is expired and invalid. I suppose in some sense, it wouldn't make sense to try and figure out what actuator input is needed faster than the actuation can actually be applied, but it should be kept in mind and documented.