Model Predictive Control:

From the simulator we obtain the state of the car (px, py, psi, v, steering angle and throttle) and the planned path of the car (ptsx, ptsy). We model where the vehicle will be after accounting for the motion of the vehicle during the latency time period:
px = px + cos(psi) * latency,
py = py + v * sin(psi) * latency,
psi = psi - (v * latency * steering_angle / 2.67),
v = v + throttle * latency.
As you can see these are just simple motion model equations. We obtained the coefficients of the path polynomial we want the car to take from the car's perspective (by rotating and translating the planned path of the car to the car's perspective:
ptsx[i] = (ptsx[i] - px) * cos(-psi) - (ptsy[i] - py) * sin(-psi) and
ptsy[i] = (ptsx[i] - px) * sin(-psi) + (ptsy[i] - py) * cos(-psi))
where each i is a point on the path and then we feed this information into our optimizer. Additionally we tell the optimizer, how many steps, N, we want the optimizer to calculate for in order to get us to our planned path. We don't want to choose too far out, nor too few steps, we experiment with this number to get 1 sec. With a planned duration of .1 seconds for each one of the N steps. We chose .1 also because this is our latency duration. So each time our optimizer plans to actuate the steering angle and throttle in our vehicle it shouldn't be less than the latency period because you don't want to calculate that you can actuate the vehicle before your car is able to. Additionally you want to plan that you can actuate (turn or adjust the throttle) as quickly as you're able to, in order to optimize your calculated path to the planned trajectory. I tried to plan a little longer (N = 15) but this yielded poorer driving values (i.e. the car almost went off the road). We also tell the optimizer our initial state and we bound the values of the actuation of the vehicle (delta and acceleration) by taking into account what the car is capable of doing. We further bound the optimizer by telling it how the car can move (the motion model) given the actuation
constraints at each time interval, t:
0 = x[t + 1] - (x[t] + v[t]*cos(psi[t]) * dt)
0 = y[t + 1] - (y[t] + v[t]*sin(psi[t]) * dt)
0 = psi[t + 10 - (psi[t] + v[t] * delta[t] / Lf * dt)
0 = v[t + 1] - (v[t] + a[t] * dt)
We also tell the optimizer to minimize the cte and epsi errors. We adjust the motion of the vehicle by adding additional constraints to create a smooth ride. After the optimizer gets our optimized calculated trajectory with each of the actuation values at each of the projected time steps we return the initial delta and acceleration values to feed to the car's actuator. Additionally we get the rest of the calculated points to show in the simulator as a comparison to the planned path. And that's it. These steps are repeated each time the simulator returns a state of the vehicle and planned path.