The Model

Model is non linear model of car movement described with six values:

- x coordinate (x_t)
- y coordinate (y_t)
- orientation angle (ψ_t)
- velocity (v_t)
- cross track error (cte_t)
- orientation angle error ($e\psi_t$).

Model incompases two actuators:

- steering_value (δ_t)
- and throttle_value (a_t).
-

Using previous state, L_f - slipping angle, $f(x)$ - desired path function, and ψdes - desired orientation, new state of model is calculated by these equations:

$$
\begin{aligned}
x_{t+1} &= x_t + v_t * cos(\psi_t) * dt \\
y_{t+1} &= y_t + v_t * sin(\psi_t) * dt \\
\psi_{t+1} &= \psi_t + \frac{v_t}{L_f} * \delta_t * dt \\
v_{t+1} &= v_t + a_t * dt \\
cte_{t+1} &= f(x_t) - y_t + v_t * sin(e\psi_t) * dt \\
e\psi_{t+1} &= \psi_t - \psi des_t + \frac{v_t}{L_f} * \delta_t * dt
\end{aligned}
$$

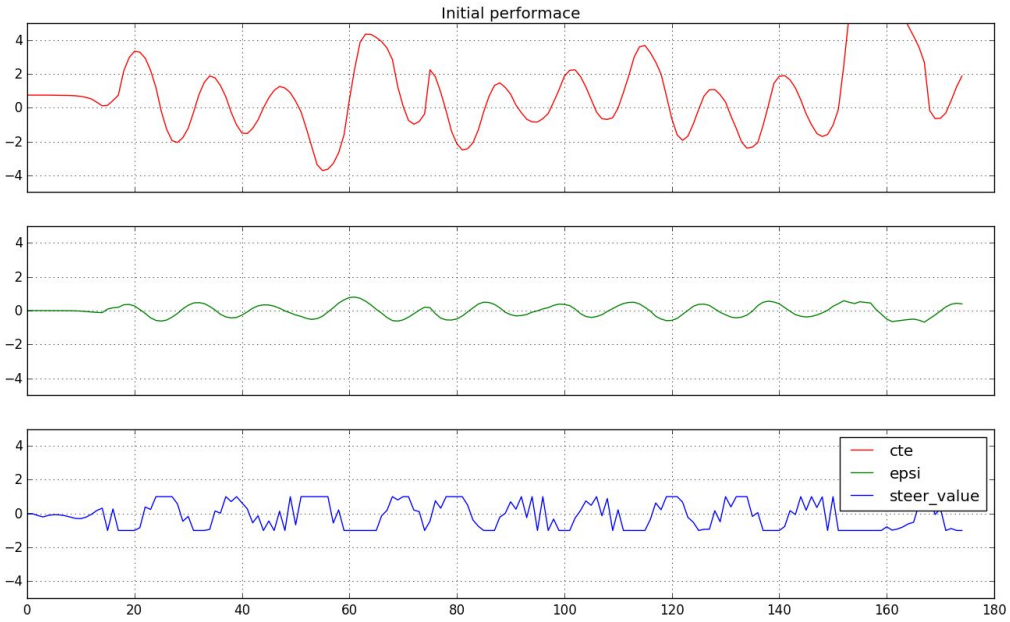
Cost is calculated as:

$$
\begin{aligned}
Cost &= c_0 * \Sigma_{t=0}^{N} cte_t^2 + c_1 * \Sigma_{t=0}^{N} e\psi_t^2 + c_2 * \Sigma_{t=0}^{N} (v_t - v_{ref})^2 \\
&+ c_3 * \Sigma_{t=0}^{N} \delta_t^2 + c_4 * \Sigma_{t=0}^{N} a_t^2 + c_5 * \Sigma_{t=1}^{N} (\delta_t - \delta_{t-1})^2 + c_6 * \Sigma_{t=1}^{N} (a_t - a_{t-1})^2
\end{aligned}
$$

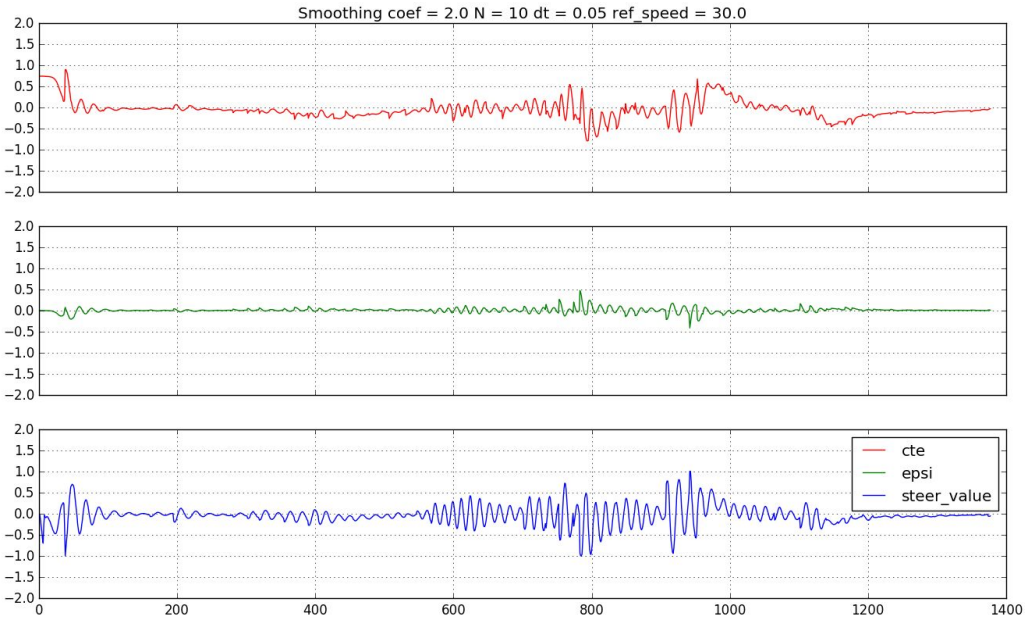
where constants c_{0-6} are used to tune the performance.

Timestep length and Elapsed Duration (N & dt)

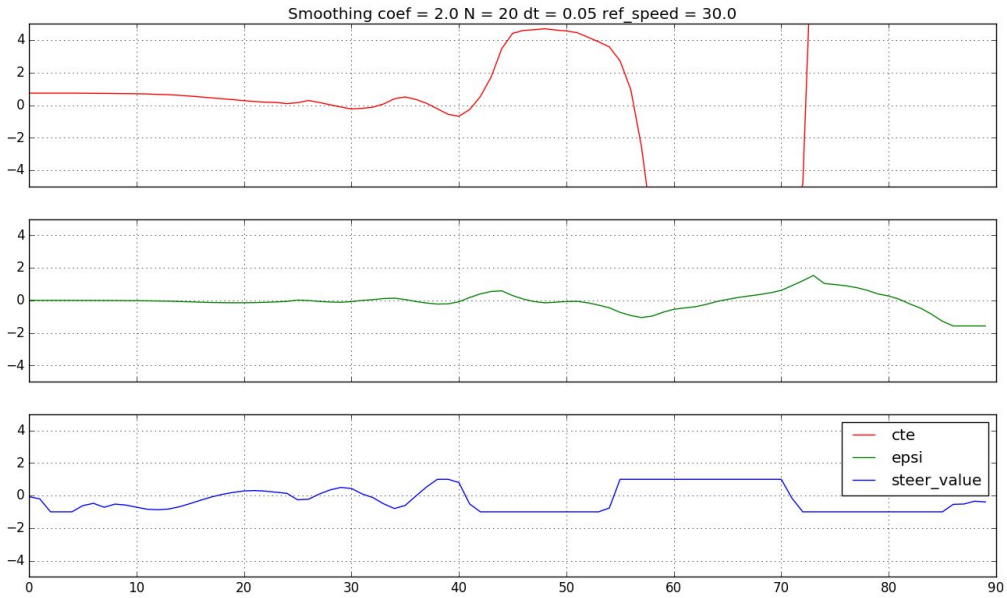
At first, all the constants ($c_{0..6}$) are set to 1.0 but the car steers off the road.



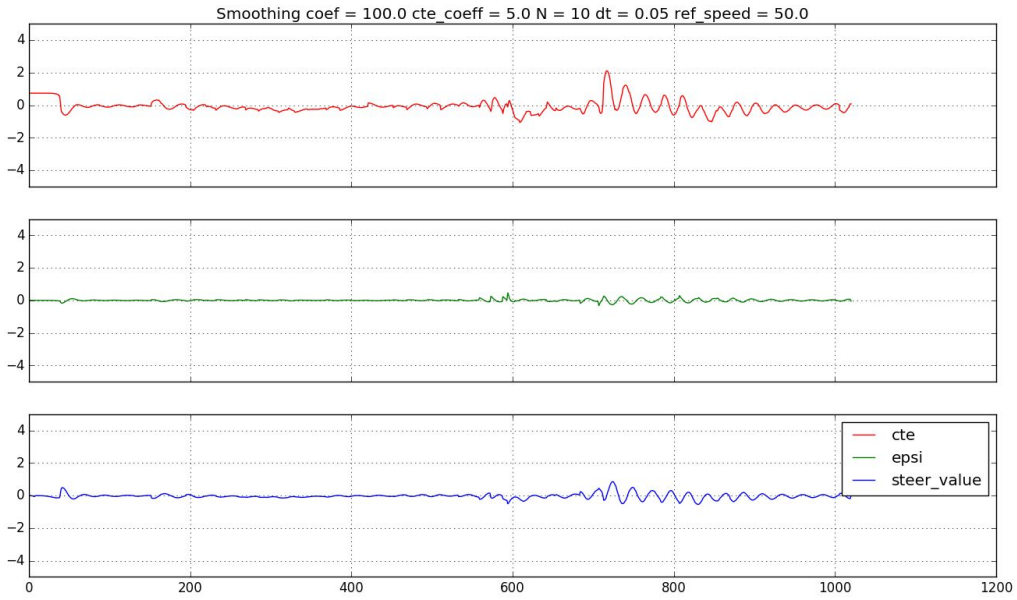
When coefficient linked with cte is increased to 2.0 model does not perform better as seen on the picture bellow.



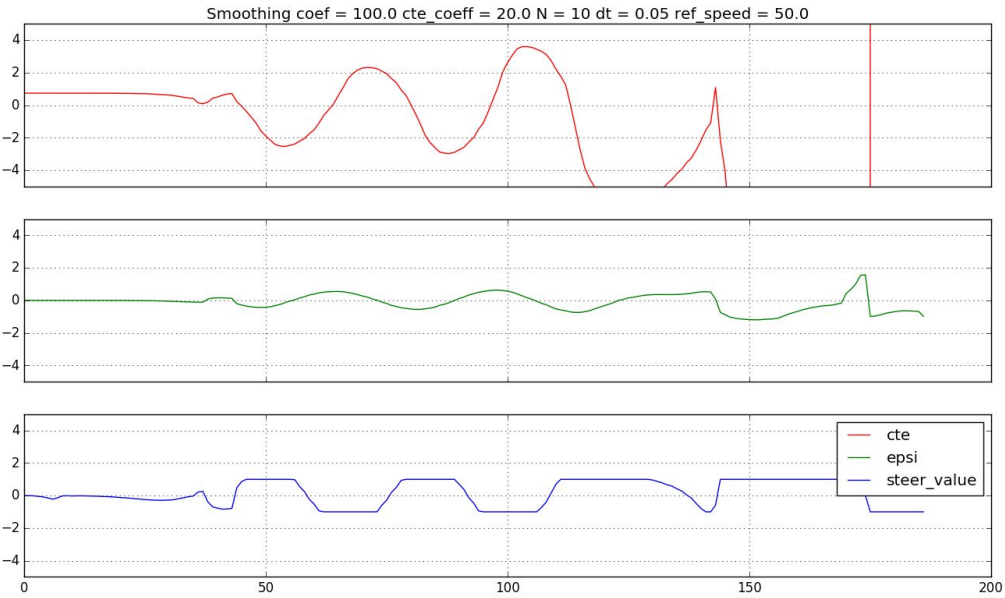
When N is increased to 20, car quickly sways of the track. This is really interesting.



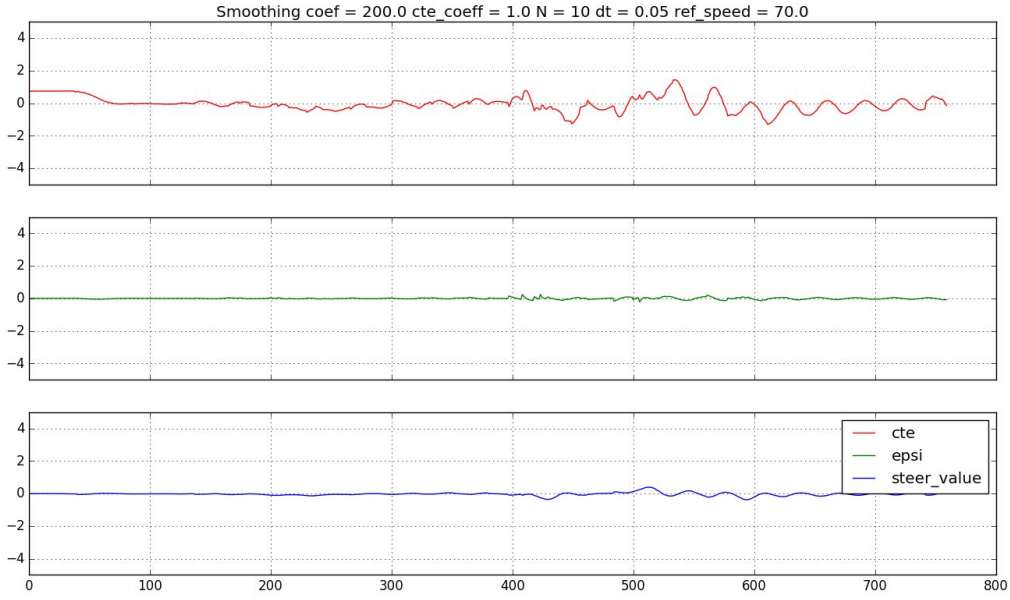
What improves performance is to increase coefficients linked with delta values and difference in delta values. When these coefficients are increased to 100s model behaves much better.



Noticeable is a wobbly ride in the middle of the track. Increasing cte_coeff seems to degrade car's performance:



That means that model with lower cte_coeff and epsi_coeff performs better.
Increasing speed to 70 is harder problem that is solved with a bit of experimentation. It turns out that only parameter that is important is smoothening of delta (coeff in the picture bellow).



Smoothing should be dependent on speed, as smoother ride is needed with higher speeds. Sharp turns should only be allowed with low speeds. Thus, I have set the smoothing coefficient as 10*current_speed for delta and 2.5*current_speed for deltas difference.

```
    for (int t = 0; t < N; t++) {
      fg[0] += 1.2*CppAD::pow(vars[cte_start + t], 2);
      fg[0] += 1.2*CppAD::pow(vars[epsi_start + t], 2);
      fg[0] += 0.3*CppAD::pow(vars[v_start + t] - ref_v, 2);
    }

    // Minimize the use of actuators.
    for (int t = 0; t < N - 1; t++) {
      fg[0] += 10*vars[v_start]*CppAD::pow(vars[delta_start + t], 2);
      fg[0] += CppAD::pow(vars[a_start + t], 2);
    }

    // Minimize the value gap between sequential actuations.
    for (int t = 0; t < N - 2; t++) {
      fg[0] += 2.5*vars[v_start]*CppAD::pow(vars[delta_start + t + 1] - vars[delta_start + t],
2);
      fg[0] += CppAD::pow(vars[a_start + t + 1] - vars[a_start + t], 2);
    }
```

In simulator drive seems to be smoother and more natural.

Polynomial Fitting and MPC Preprocessing

Waypoints are fitted with 3-rd degree polynomial. No preprocessing is used.

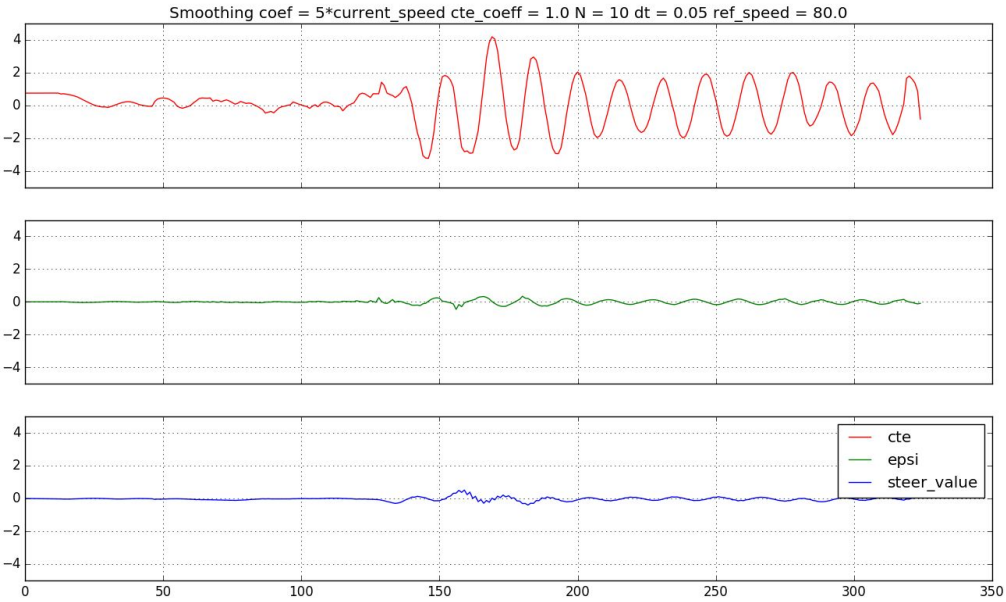
Model Predictive Control with Latency

To handle latency, actuators from the previous iteration of calculation are treated as constant for the duration of latency. As new actuations, actuators just after latency period are returned. Now, variable constraints look like this:

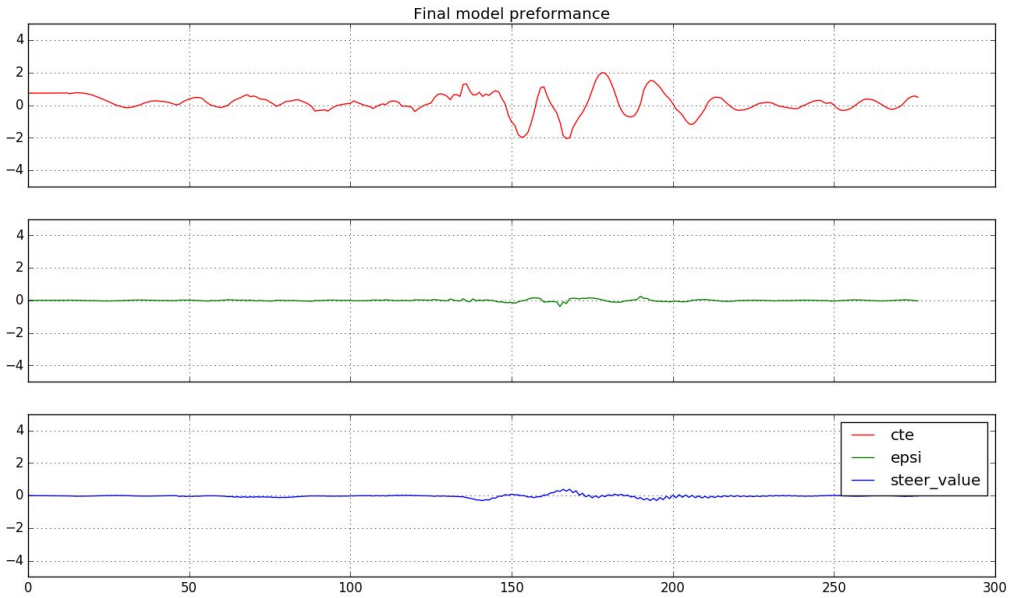
```
  for (int i = delta_start; i < delta_start + n_latency; i++) {
    vars_lowerbound[i] = previous_delta;
    vars_upperbound[i] = previous_delta;
  }

  // The upper and lower limits of delta are set to -25 and 25
  // degrees (values in radians).
  for (int i = delta_start + n_latency; i < a_start; i++) {
    vars_lowerbound[i] = -0.436332;
    vars_upperbound[i] = 0.436332;
```

```
}

// If there is a latency in actualizations
// actuators can not be changed during latency period
for (int i = a_start; i < a_start + n_latency; i++) {
  vars_lowerbound[i] = previous_a;
  vars_upperbound[i] = previous_a;
}

// Acceleration/deceleration upper and lower limits.
for (int i = a_start + n_latency; i < n_vars; i++) {
  vars_lowerbound[i] = -1.0;
  vars_upperbound[i] = 1.0;
}
```

Following performance is achieved:



By tuning cte_coeff = 1.2, epsi_coeff = 1.2 and v_coeff = 0.3 I get following performance:



Taken trajectory is a still not greatest in the middle section of the track.