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:

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

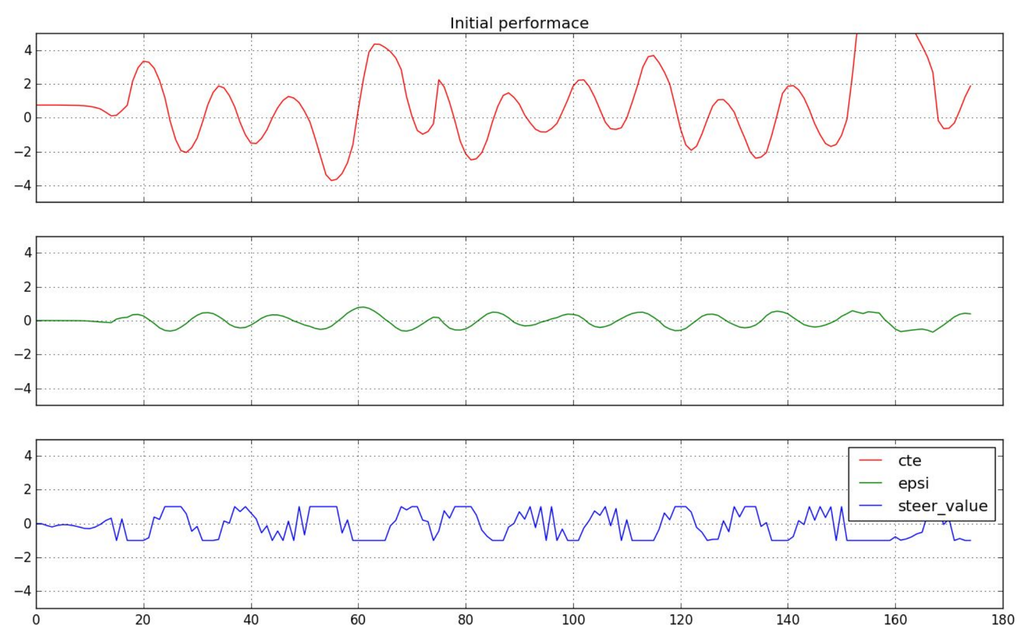
where constants c_{0-6} are used to tune the performance.

Timestep length and Elapsed Duration (N & dt)

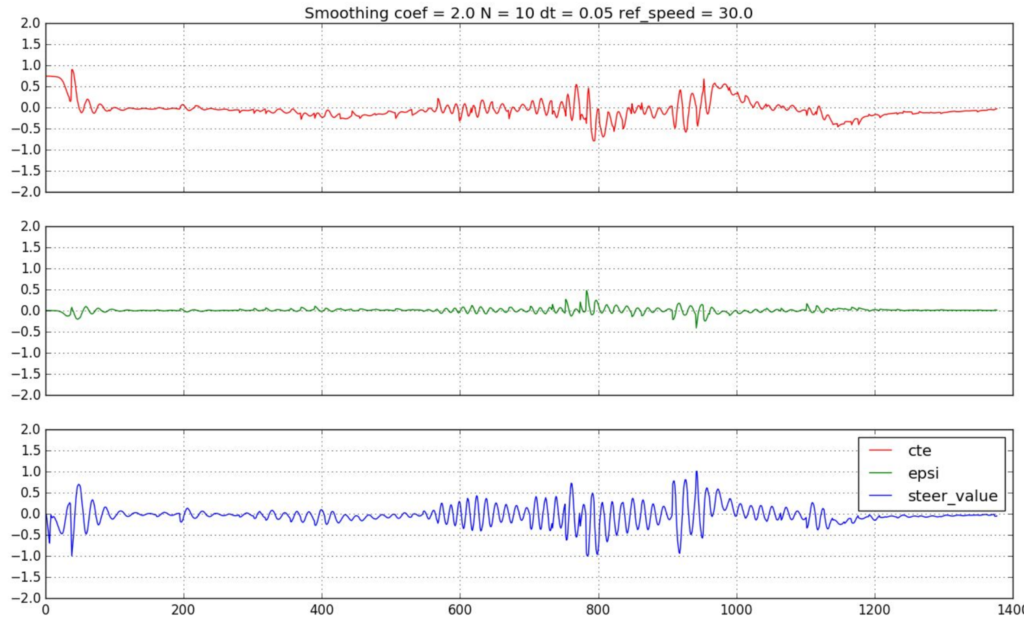
N is number of points in future where state of the car is calculated and fitted to waypoints. Greater *N* means larger time interval our model predicts, but it also linearly increases number of calculations. *dt* is time interval between these waypoints. Larger *dt* means coarser model, and therefore less accurate model, but also less calculations.
By experimenting with N and dt the best results are achieved with values 10 and 0.05s, as visible in the table below.

N	dt	Avg cte^2 after 250 activations
10	0.05	0.423208
12	0.05	Does not finish one lap
9	0.05	0.918081
10	0.03	Does not finish one lap
10	0.07	0.830302
10	0.06	Does not finish one lap
10	0.04	1.9032
20	0.025	Does not finish one lap
10	0.1	1.57281

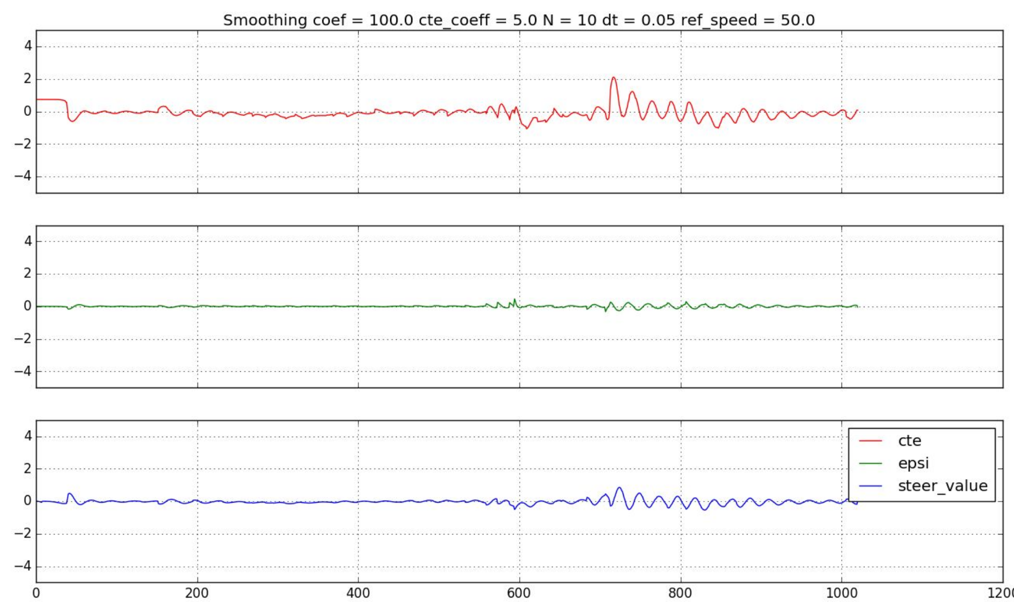
Other parameters of the model (those used in cost function) are also determined using a lot of experimentation.
At first, all the constants ($c_{0..6}$) are set to 1.0 but the car steers off the road.



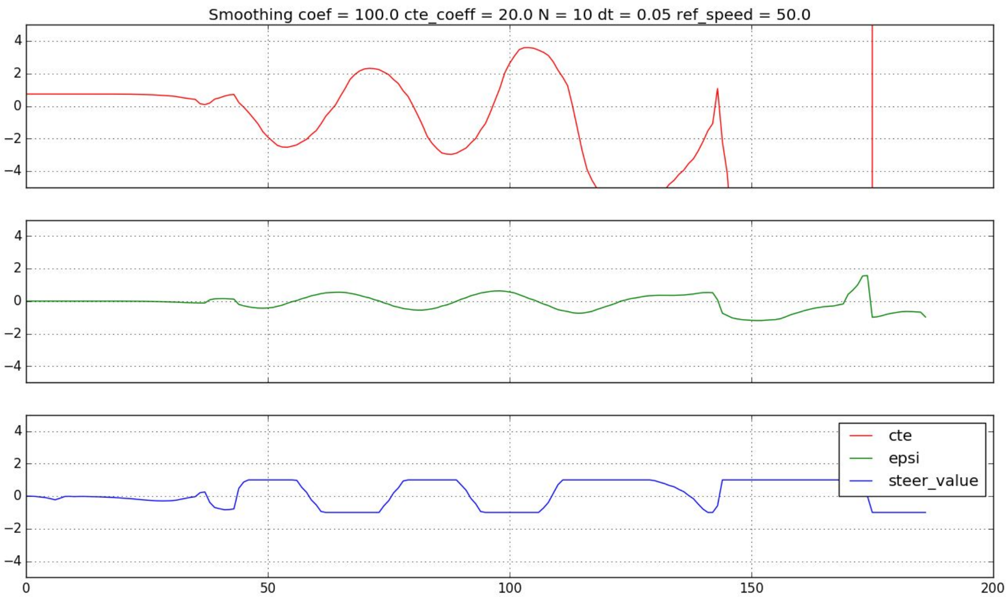
When coefficient linked with cte is increased to 2.0 model does not perform better as seen on the picture bellow.



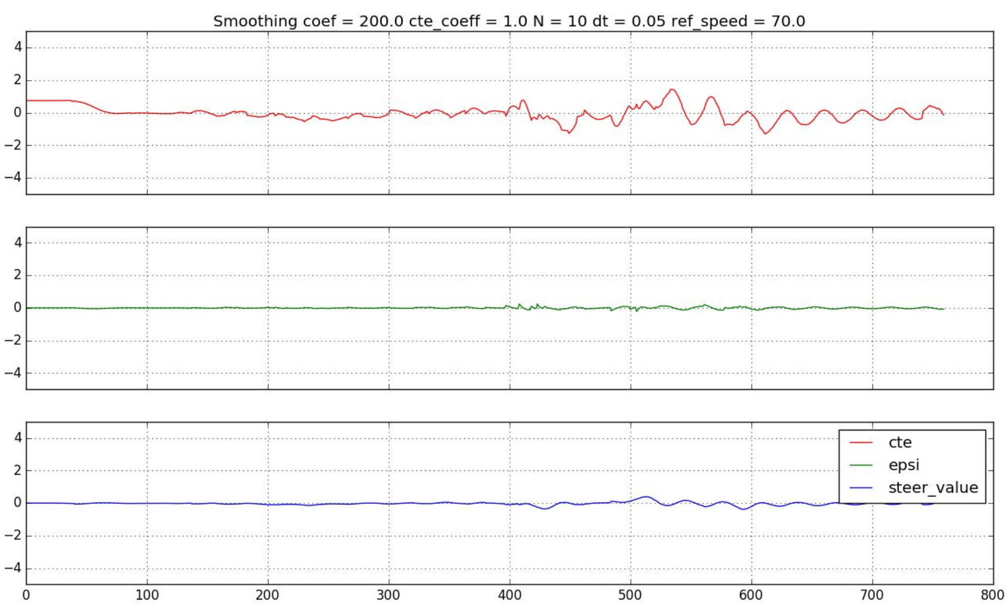
What improves performance is to increase coefficients linked with delta values and difference in delta values. When these coefficients are increased to 100s model behaves much better.



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. 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.

Next step is to fine tune individual coefficients. All the experiments are done with 78.0 as referent speed and N = 10, dt = 0.05. Delta_diff coefficient does not affect speed much but it can improve cte by a small fraction.

cte_coeff	epsi_coeff	v_coeff	delta_coeff	delta_diff	Avg cte	Avg speed
1.2	1.2	0.3	10*v	2.5*v	0.428346	63.689
1.2	1.2	0.3	10*v	3.0*v	0.445552	63.6372
1.2	1.2	0.3	10*v	2.0*v	0.502449	63.4979
1.2	1.2	0.3	10*v	2.75*v	0.416164	63.7419
1.2	1.2	0.3	10*v	2.625*v	0.421364	63.6447
1.2	1.2	0.3	10*v	2.6875*v	0.465285	63.7781

Fine tuning delta_diff coefficient

Delta_diff has more impact on the performance:

cte_coeff	epsi_coeff	v_coeff	delta_coeff	delta_diff	Avg cte	Avg speed
1.2	1.2	0.3	10.5*v	2.75*v	0.518043	64.1856
1.2	1.2	0.3	9.5*v	2.75*v	0.405574	63.7962
1.2	1.2	0.3	9.0*v	2.75*v	0.442036	64.1916
1.2	1.2	0.3	9.75*v	2.75*v	0.407592	63.5107
1.2	1.2	0.3	9.625*v	2.75*v	0.421677	63.6758

Fine tuning cte_coeff also improves performance,

cte_coeff	epsi_coeff	v_coeff	delta_coeff	delta_diff	Avg cte	Avg speed
1.25	1.2	0.3	9.5*v	2.75*v	0.397649	63.7981
1.3	1.2	0.3	9.5*v	2.75*v	0.383036	63.7453
1.35	1.2	0.3	9.5*v	2.75*v	0.399658	63.6486
1.275	1.2	0.3	9.5*v	2.75*v	0.432678	63.7376
1.325	1.2	0.3	9.5*v	2.75*v	0.389581	63.4558

and epsi_coeff does not.

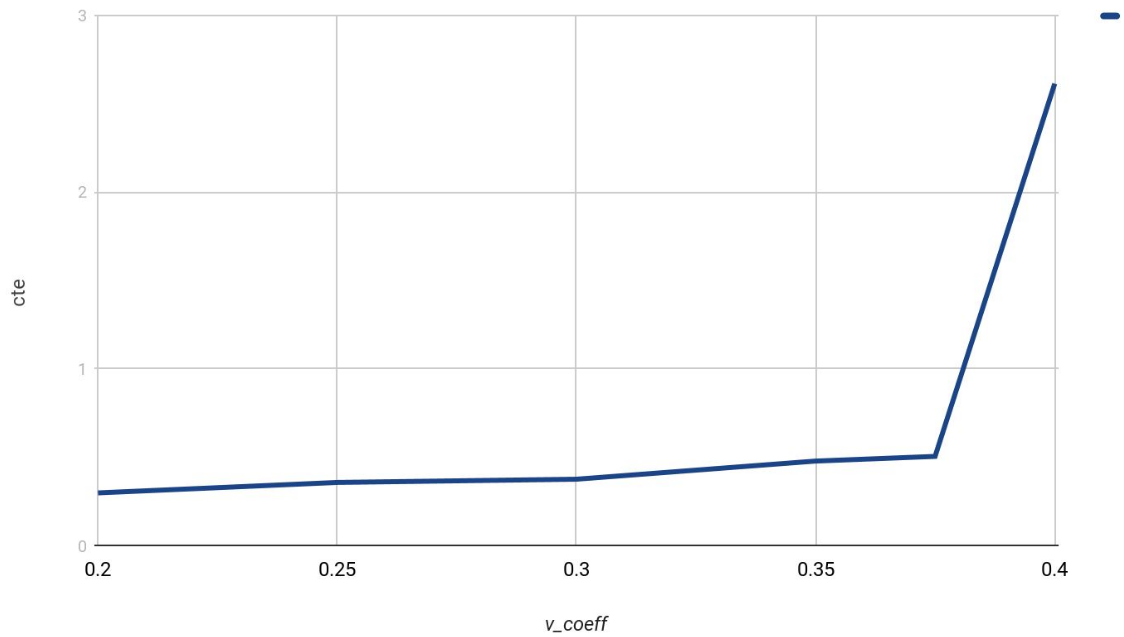
cte_coeff	epsi_coeff	v_coeff	delta_coeff	delta_diff	Avg cte	Avg speed
1.3	1.25	0.3	9.5*v	2.75*v	0.488583	64.0852
1.3	1.15	0.3	9.5*v	2.75*v	0.411887	63.5728
1.3	1.175	0.3	9.5*v	2.75*v	0.511228	63.5375

Tuning v_coeff is interesting as higher values increase speed of car but also increase cte. This makes sense as faster should produce less precise driving, and slower more precise driving. This is depicted on two graphs below.

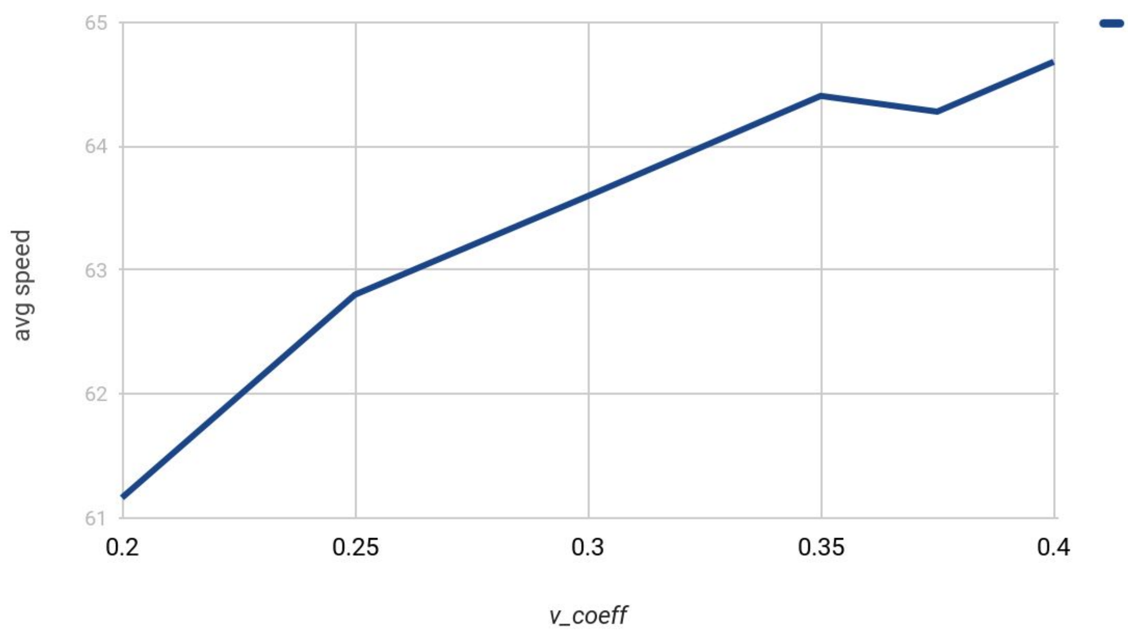
cte_coeff	epsi_coeff	v_coeff	delta_coeff	delta_diff	Avg cte	Avg speed
1.3	1.2	0.3	9.5*v	2.75*v	0.376888	63.5983
1.3	1.2	0.4	9.5*v	2.75*v	2.61289	64.6817[1]
1.3	1.2	0.2	9.5*v	2.75*v	0.299554	61.1637
1.3	1.2	0.25	9.5*v	2.75*v	0.3586	62.8022
1.3	1.2	0.35	9.5*v	2.75*v	0.479412	64.4066
1.3	1.2	0.375	9.5*v	2.75*v	0.505862	64.2795

[1] Car goes of the track

CTE



Average Speed



```
 for (int t = 0; t < N; t++) {
    fg[0] += 1.3*CppAD::pow(vars[cte_start + t], 2);
    fg[0] += 1.2*CppAD::pow(vars[epsi_start + t], 2);
    fg[0] += 0.35*CppAD::pow(vars[v_start + t] - ref_v, 2);
  }

  // Minimize the use of actuators.
  for (int t = 0; t < N - 1; t++) {
    fg[0] += 9.5*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] += 2.75*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);
  }
```

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;
```

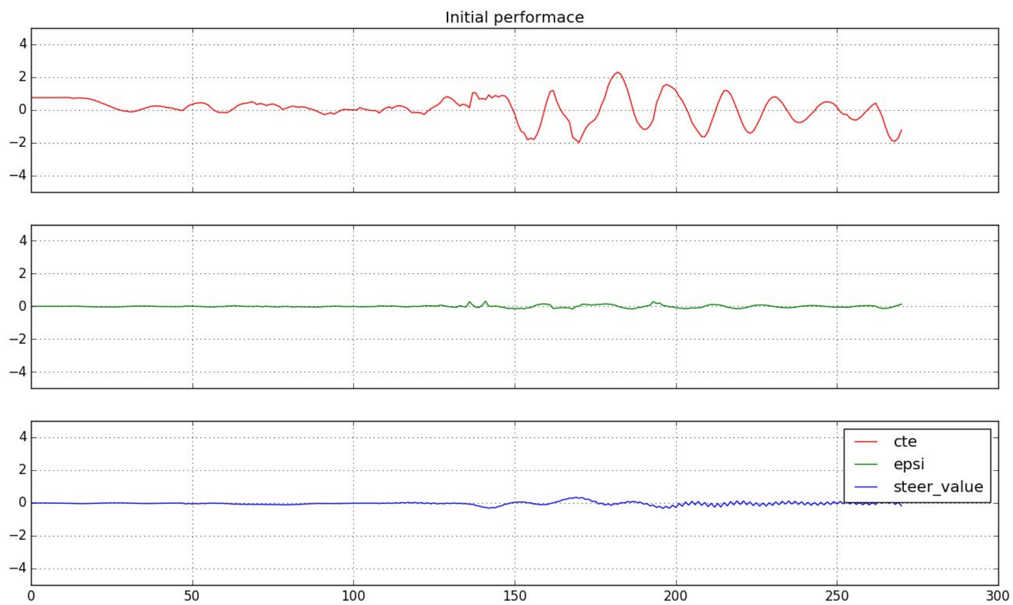
```
}

// 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:

avg cte = 0.547741
avg speed = 64.2907



There is still a lot of CTE error in the second part of the track. Steering is also suboptimal in this part of the track as it constantly shifts from small positive value to small negative value in zigzag fashion.