Kinematic model

The model used is a kinematic model that ignores tire forces, gravity, mass.

This model descripes the position in x,y , heading with respect to the map(world) , velocity , the cross-track error and psi error .

x[t] = x[t-1] + v[t-1] * cos(psi[t-1]) * dt
y[t] = y[t-1] + v[t-1] * sin(psi[t-1]) * dt
psi[t] = psi[t-1] + v[t-1] / Lf * delta[t-1] * dt
v[t] = v[t-1] + a[t-1] * dt
cte[t] = f(x[t-1]) - y[t-1] + v[t-1] * sin(epsi[t-1]) * dt
epsi[t] = psi[t] - psides[t-1] + v[t-1] * delta[t-1] / Lf * dt

This model use the following data:
-Lf : distance between Center of mass of the car and the front wheel
-a , delta : actuators update (accelration and steering angle).

Timestep Length and Elapsed Duration (N & dt):

Timestep length (N) is chosen to be 9, and timestep frequency (dt) 0.11sec.(100 millisecond latency between actuations commands and additional 10 milliseconds for connection latency)

Using much more N causes over processing that leads the car to carsh. I tried (N=16) in the video videos/16N.wmv

Using less N causes the car to drive unsafe and don't care to corrent itself. I tried (N=4) in the video videos/4N.mkv

Overestimating the delay (0.3 for example) make the driving not smooth (may crash) and in general the speed is very low.

Underestimating the delay (0.02 for example) make the car overshooting (It's no longer MPC .. It's a worse than PID control)

Polynomial Fitting , MPC Preprocessing Cost Function Parameters:

Transformed waypoints coordinates to the cars coordinates.
Used fit polynomial to the points - 3rd order.
Used the following costs for different parameters:

```
const int cteWeight = 3000;
const int epsiWeight = 1000;
const int vWeight = 1;
const int deltaWeight = 50;
const int aWeight = 20;
const int deltaChangeWeight = 50;
const int aChangeWeight = 100;
```

cteWeight , aWeight ,aChangeWeight were the most effective costs that enhanced my lap .