MODEL PREDICTIVE CONTROL
project report

Goal: Successfully drive the car around the track in the simulator. No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

Implementation Details:

Vehicle Model: This project uses the kinematic model; external forces are not taken into account. Below are the equations used in the model.

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

In the above equations, x and y indicate the position of the car, psi is the heading direction, v denotes velocity, cte denotes cross track error and epsi is the orientation error. Lf is the length from front to CoG (centre of gravity) of the car.

Timestep Length and Elapsed Duration (N && dt): After repeated trial and error experiments, N = 12 and dt = 0.08 worked out well. I have tried out different N values ranging between 6 and 25. When N = 6 and dt = 0.05, the car was very responsive, but then it suffered from short sightedness and couldn't adapt well with curvy roads. For N = 25 and dt = 0.1, there was noticeable lag and therefore increased latency because of increased computations.

Polynomial Fitting: I have transformed from global coordinates to vehicle's coordinates using these equations.

*x = deltax * cos(-psi) - deltay * sin(-psi)*
*y = deltax * sin(-psi) + deltay * cos(-psi)*

State after transformation: state << 0, 0, 0, v, cte, epsi;

I have used a 3-degree polynomial (ax^3 + bx^2+ cx + d) to fit the waypoints.

Latency: To deal with latency, the future position is estimated based on t-2nd position. This is implemented in the following lines:

```
    if(t > 2){
  a0 = vars[a_start + t - 3];
  delta0 = vars[delta_start + t - 3];
}
```