Self-driving Car based on Model Predictive Control

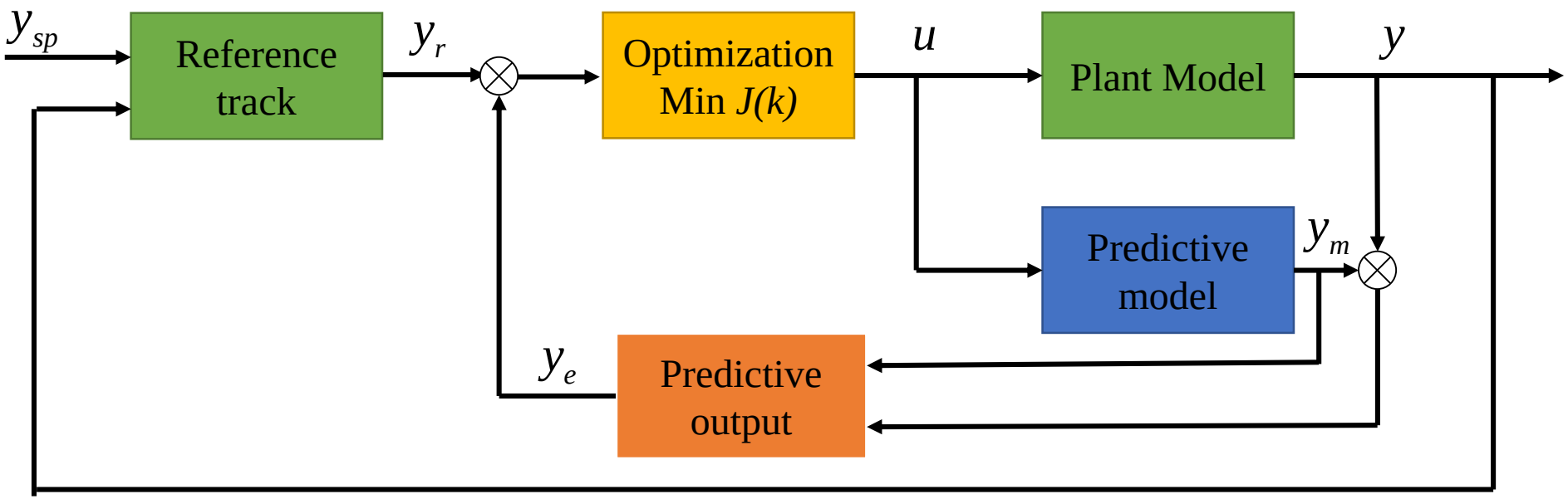
Benchun Zhou

School of Automation Science and Electronic Engineering
Beihang University

CONTENTS

- **Framework**

Model predictive controllers rely on dynamic models of the process. The main advantage of MPC is the fact that it allows the current timeslot to be optimized, while keeping future timeslots in account. This is achieved by optimizing a finite time-horizon, but only implementing the current timeslot. MPC has the ability to anticipate future events and can take control actions accordingly.



• Kinematic Bicycle Model

- The nonlinear continuous time equations that describe a kinematic bicycle model

$$\dot{x} = v\cos(\psi + \beta)$$

$$\dot{y} = v\sin(\psi + \beta)$$

$$\dot{\psi} = \frac{v}{l_r}\sin(\beta)$$

$$\dot{v} = a$$

$$\beta = \tan^{-1}\left(\frac{l_r}{l_r + l_f}\tan(\delta_f)\right)$$

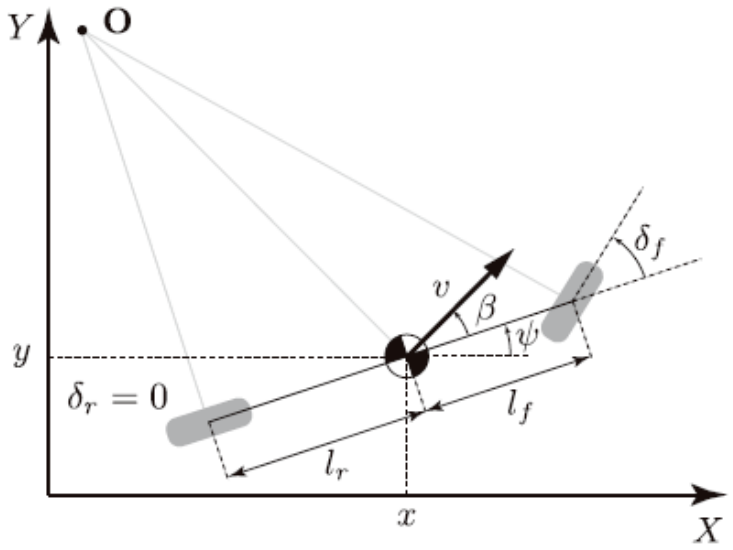


Fig. 1: Kinematic Bicycle Model

(x, y)	The current location in x-axis, y-axis
ψ	The current orientation / heading of the vehicle
v	The current velocity/speed of the vehicle
(l_f, l_r)	The distance from the center of mass to the front and rear
β	Angle of the current velocity with respect to the longitudinal ax
δ_f	The control inputs are the front and rear steering angles

• Kinematic Bicycle Model

- The nonlinear continuous time equations that describe a kinematic bicycle model

$$\dot{x} = v\cos(\psi + \beta)$$
$$\dot{y} = v\sin(\psi + \beta)$$
$$\dot{\psi} = \frac{v}{l_r}\sin(\beta)$$
$$\dot{v} = a$$
$$\beta = \tan^{-1}\left(\frac{l_r}{l_r + l_f}\tan(\delta_f)\right)$$

discretization

$$x_{t+1} = x_t + v_t\cos(\psi_t + \beta)\times dt$$
$$y_{t+1} = y_t + v_t\sin(\psi_t + \beta)\times dt$$
$$\psi_{t+1} = \psi_t + \frac{v_t}{l_r}\sin(\beta)\times dt$$
$$v_{t+1} = v_t + a_t \times dt$$
$$\beta = \tan^{-1}\left(\frac{l_r}{l_r + l_f}\tan(\delta_f)\right)$$

rewrite

$$\dot{X} = AX + BU$$
$$X = [x, y, \psi, v]$$
$$U = (a, \delta_f)$$

discretization

$$x(k+1) = A_{k,t}x(k) + B_{k,t}u(k)$$

- **Track information**

The track information is a reference track which the car want to follow

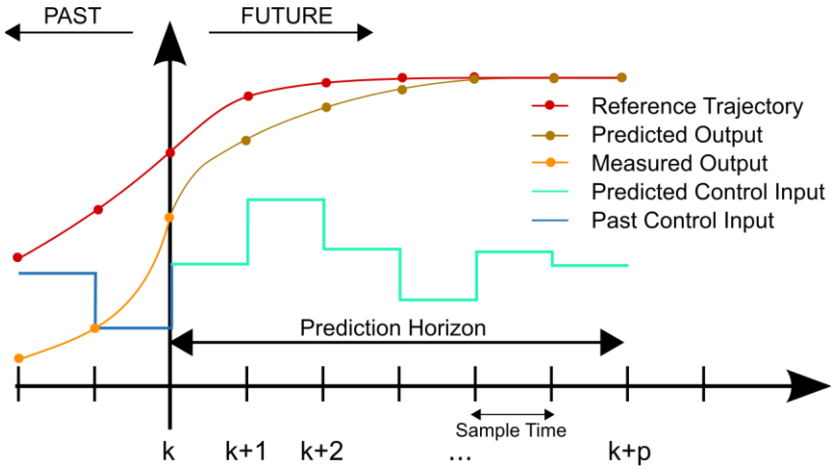
- If we know all the path, we can just discretize the path
- However, if we only know the some waypoint of the path, we would better use a function (such as a 3rd order polynomial function) to get the path

- **Prediction**

The state and output function can be described as:

$$x(k+1) = A_{k,t}x(k) + B_{k,t}u(k)$$
$$y(k+1) = Cx(k)$$

When assigned: $\xi(k|t) = \begin{bmatrix} x(k|t) \\ u(k-1|t) \end{bmatrix}$



The state and output function can changed into

$$\xi(k+1|t) = \tilde{A}_{k,t}\xi(k|t) + \tilde{B}_{k,t}\Delta u(k|t)$$
$$\eta(k|t) = \tilde{C}_{k,t}\xi(k|t)$$

$$\tilde{A}_{k,t} = \begin{bmatrix} A_{k,t} & B_{k,t} \\ 0 & \mathrm{I}_m \end{bmatrix} \quad \tilde{B}_{k,t} = \begin{bmatrix} B_{k,t} \\ \mathrm{I}_m \end{bmatrix} \quad \tilde{C}_{k,t} = \begin{bmatrix} C_{k,t} & 0 \end{bmatrix}$$

If the prediction horizon is Np, control horizon is Nc (sometimes, Np=Nc), the prediction of car can be describe as:

$$\xi(t+N_p|t) = \tilde{A}_t^{N_p}\xi(t|t) + \tilde{A}_t^{N_p-1}\tilde{B}_t\Delta u(t|t) + \ldots + \tilde{A}_t^{N_p-N_c-1}\tilde{B}_t\Delta u(t+N_c|t)$$
$$\eta(t+N_p|t) = \tilde{C}_{t,t}\tilde{A}_{t,t}^{N_p}\xi(t|t) + \tilde{C}_t\tilde{A}_t^{N_p-1}\tilde{B}_t\Delta u(t|t) + \ldots + \tilde{C}_t\tilde{A}_t^{N_p-N_c-1}\tilde{B}_t\Delta u(t+N_c|t)$$

- **Prediction**

The prediction of car can be also rewrite as:

$$X(t) = \Psi_t \xi(t|t) + \Theta_t \Delta U(t)$$

$$Y(t) = \tilde{C}_t X(t)$$

$$X(t) = \begin{bmatrix} \xi(t+1|t) \\ \xi(t+2|t) \\ \cdots \\ \xi(t+N_c|t) \\ \cdots \\ \xi(t+N_p|t) \end{bmatrix} \quad Y(t) = \begin{bmatrix} \eta(t+1|t) \\ \eta(t+2|t) \\ \cdots \\ \eta(t+N_c|t) \\ \cdots \\ \eta(t+N_p|t) \end{bmatrix} \quad \Psi_t = \begin{bmatrix} \tilde{A}_t \\ \tilde{A}_t^2 \\ \cdots \\ \tilde{A}_t^{N_c} \\ \cdots \\ \tilde{A}_t^{N_p} \end{bmatrix} \quad \Delta U(t) = \begin{bmatrix} \Delta u(t|t) \\ \Delta u(t+1|t) \\ \cdots \\ \Delta u(t+N_c|t) \end{bmatrix}$$

$$\Theta_t = \begin{bmatrix} \tilde{B}_t & 0 & 0 & 0 \\ \tilde{A}_t\tilde{B}_t & \tilde{B}_t & 0 & 0 \\ \cdots & \cdots & \ddots & \cdots \\ \tilde{A}_t^{N_c-1}\tilde{B}_t & \tilde{A}_t^{N_c-2}\tilde{B}_t & \cdots & \tilde{B}_t \\ \tilde{A}_t^{N_c}\tilde{B}_t & \tilde{A}_t^{N_c-1}\tilde{B}_t & \cdots & \tilde{A}_t\tilde{B}_t \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{A}_t^{N_p-1}\tilde{B}_t & \tilde{A}_t^{N_p-2}\tilde{B}_t & \cdots & \tilde{A}_t^{N_p-N_c}\tilde{B}_t \end{bmatrix}$$

Thus, we can get the predictive state and predictive output using current state $\xi(t|t)$ and control command $\Delta U(t)$

- **Optimization**

the object of MPC is to minimize a cost function as follows

$$J(k)=\sum_{j=1}^{N}\tilde{\chi}^{T}(k+j|k)Q\tilde{\chi}(k+j|k)+\tilde{u}^{T}(k+j-1)Q\tilde{u}(k+j-1)$$

Final state　　Control command

To make the command smooth, we take delta control command as the state of cost function

$$J(\xi(t),u(t-1),\Delta U(t))=\sum_{t=1}^{N_p}\left\|\eta(t+i|t)-\eta_{desire}(t+i|t)\right\|_Q^2+\sum_{t=1}^{N_c}\left\|\Delta u(t+i|t)\right\|_R^2$$

With control constraints

$$u_{\min}(t+k)\le u(t+k)\le u_{\max}(t+k)\qquad k=0,1,\ldots,N_c-1$$

control delta constraints

$$\Delta u_{\min}(t+k)\le \Delta u(t+k)\le \Delta u_{\max}(t+k)\qquad k=0,1,\ldots,N_c-1$$

output constraints

$$y_{\min}(t+k)\le y(t+k)\le y_{\max}(t+k)\qquad k=0,1,\ldots,N_c-1$$

When derived with kinematic, the cost function can be rewrite as

$$J(\xi(t),u(t-1),\Delta U(t))=\Delta U(t)H_t\Delta U^{T}(t)+G_t\Delta U^{T}(t)+P_t$$

Where $H_t=\Theta_t^{T}Q_e\Theta_t+R_e$　$G_t=2E^{T}(t)Q_e\Theta_t$　$P_t=E^{T}(t)Q_eE(t)$ and Pt is constant　$E(t)=\Psi_t\xi(t)-y_{desire}(t)$

- **Feedback Control**

At each control step, we can get a set of control command as following

$$\Delta U_t^* = \left[\Delta u_t^*, \Delta u_{t+1}^*, ..., \Delta u_{t+N_c-1}^* \right]$$

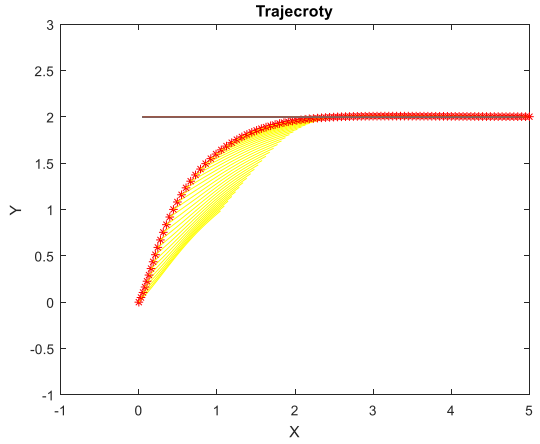
According to MPC controller, we use the first element as the real command

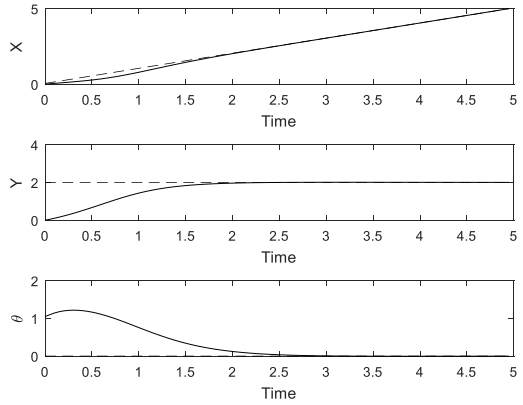
$$u(t) = u(t-1) + \Delta u_t^*$$

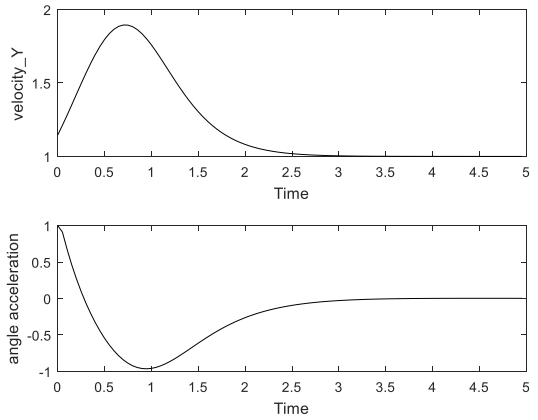
The system will output new state, and we can predict new state and calculate new command

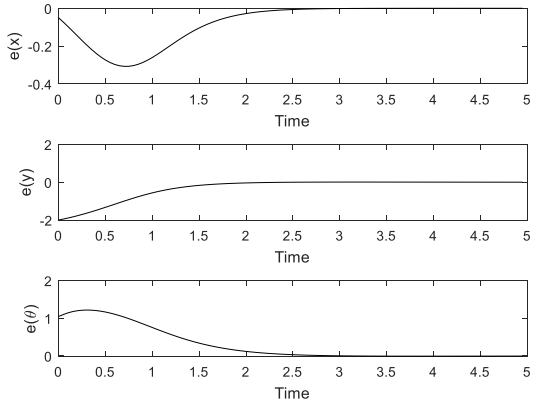
- **Simple simulation in Matlab**

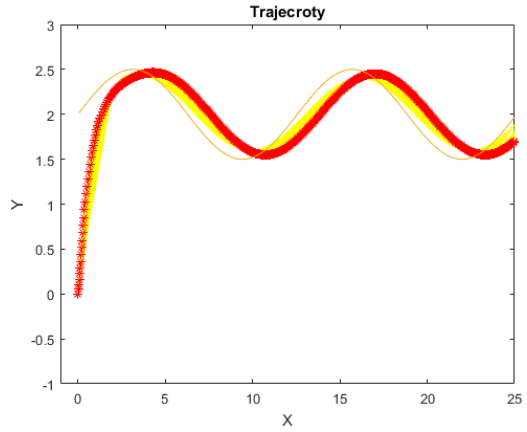
In Matlab, the reference track is a known information, and we aim to simply follow the track

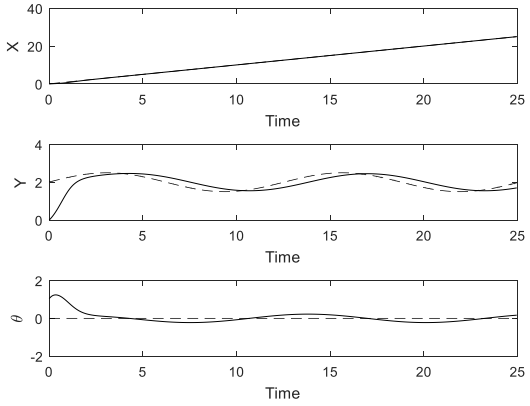


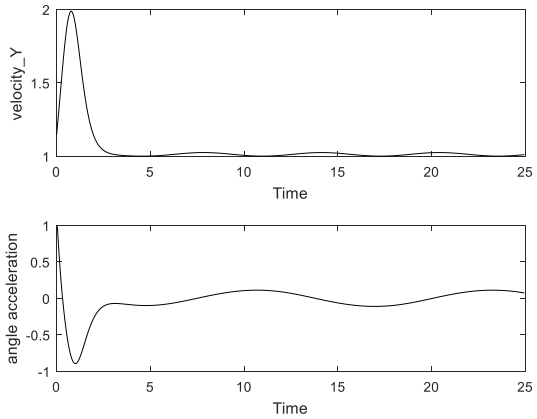


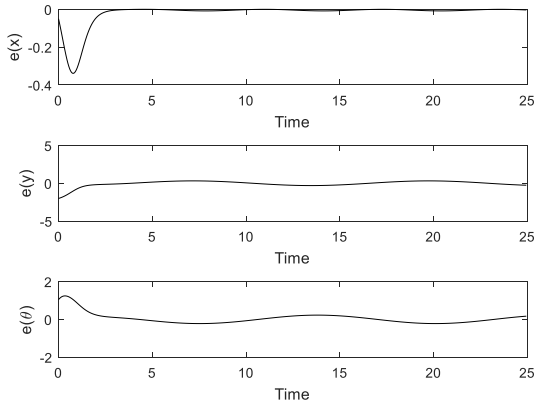












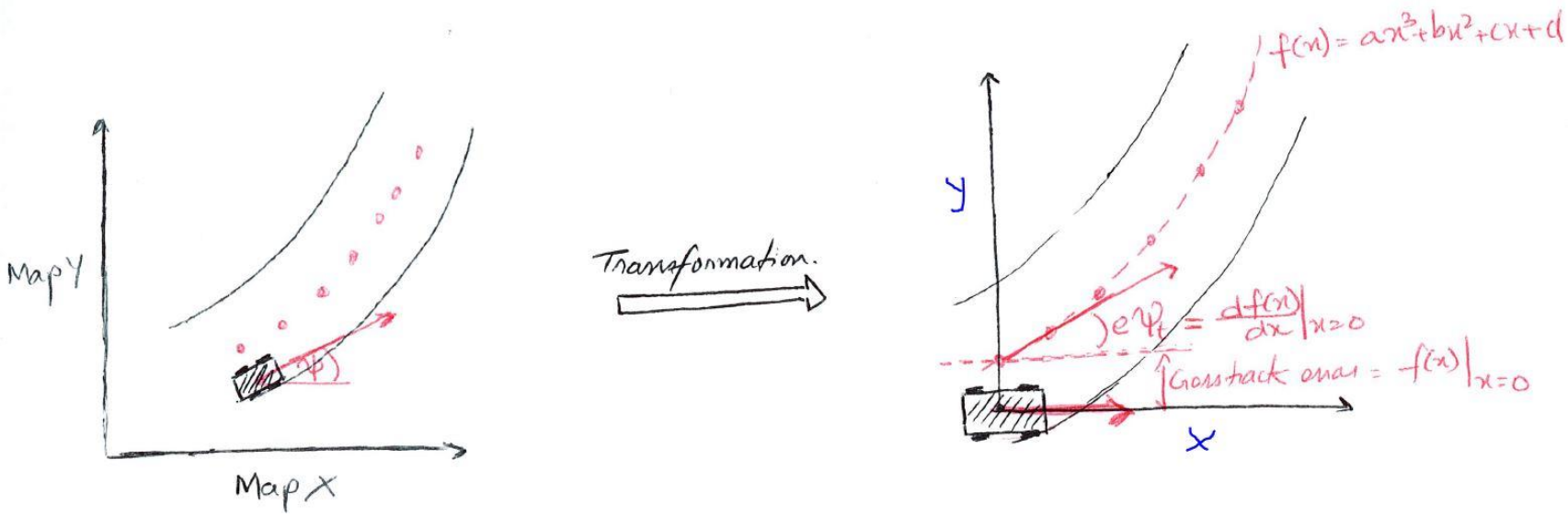
- **Simulation in Unity**

The aim of the project is to drive a car in a simulator by taking current state of car and new few way points and obtaining ideal throttle and steering values such that car is able to navigate without going off road. The idea is to use method of Model Predictive Car Control. In this we use the cars turning radius and create a model of car behavior for given set of actuations and then using this model obtain ideal values of actuations by defining a cost function. The cost function incorporates the cross track error, orientation error, target speed and values and changes in these values in actuation.

- **Simulation in Unity**

Track information

- Transfer waypoint to track using polynomial fitting



- Firstly, transfer the waypoints from map coordinate system to car coordinate system
- Secondly, fit the waypoints with a 3rd order polynomial function f(x)

$$y_{desire} = f(x) = k_3 x^3 + k_2 x^2 + k_1 x^1 + k_0$$

$$\tan(\psi_{desire}) = dy/dx = 3k_3 x^2 + 2k_2 x^1 + k_1$$

- **Simulation in Unity**

Cost Function

Reference track info:

$$y_{desire} = f(x) = k_3 x^3 + k_2 x^2 + k_1 x^1 + k_0$$

$$\tan(\psi_{desire}) = dy/dx = 3k_3 x^2 + 2k_2 x^1 + k_1$$

$$y_{desire} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_{10} \end{bmatrix} \quad \psi_{desire} = \begin{bmatrix} \psi_1 \\ \psi_2 \\ \cdots \\ \psi_{10} \end{bmatrix}$$

$$cte_{t+1} = cte_t + v_t \sin(e\psi_{t+1}) \times dt$$

$$e\psi_{t+1} = e\psi_t + \frac{v_t}{l_r} \times \delta_t \times dt$$

$$cte_t = y_{desire} - y_t$$

$$e\psi_t = \psi_{desire} - \psi_t$$

cte_{t+1} the cross track error which is the difference between our desired position and actual position.

$e\psi_{t+1}$ The orientation error which is the difference between our desired heading and actual heading.

- **Simulation in Unity**

Cost Function

In the simulator, since delta rotation is opposite to the convention followed in the model equations, in the model equation mentioned above, delta is replaced with negative delta. The cost function was implemented as sum of squared values of following parameters

$$J = \sum_{t=1}^{N} A \times \|cte_t\|^2 + B \times \|e\psi_t\|^2 + C \times \|v - v_{\max}\|^2$$
$$+ \sum_{t=1}^{N-1} D \times \|\delta_t\|^2 + E \times \|a_t\|^2$$
$$+ \sum_{t=1}^{N-2} F \times \|\delta_t - \delta_{t-1}\|^2 + G \times \|a_t - a_{t-1}\|^2$$

1. Cross track error
2. Orientation error
3. Difference between current and target velocity
4. Values of actuators; steering
5. Values of actuators: acceleration.
6. Gap between sequential actuations;
7. change in steering and acceleration.

Weight: A=1500, B=1500, C=1, D=10, E=10, F=150, G=15

Constraints: $\delta \in \left[-25^\circ, 25^\circ\right], a \in [-1,1]$

- **Result**

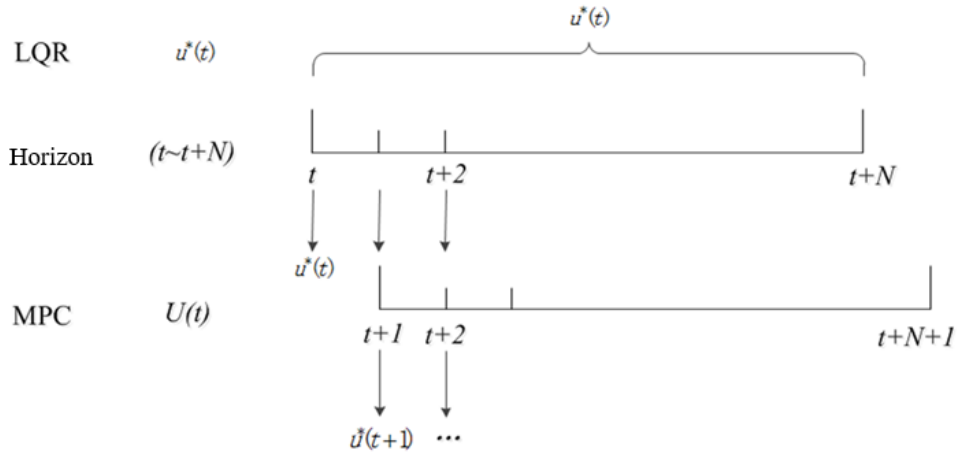
5. Comparison between MPC and LQR

- **Objective**

  LQR: linear system
  MPC: linear and nonlinear system
  Self-driving car: change from nonlinear system to linear system

- **Horizon**

  LQR: fix window, use the single solution for whole time horizon
  MPC: receding time window, use new solution every receding time



- **Cost function**

  LQR: $J = \frac{1}{2}\chi^T(t_f)Q_0\chi(t_f) + \frac{1}{2}\int_{t_0}^{t_f}\left[\chi^T Q\chi + u^T Ru\right]dt$

  MPC: $J = \chi(t+N)Q_0\chi(t+N) + \sum_{j=1}^{N}\chi^T(t+j|k)Q\chi(k+j|k) + u^T(k+j-1)Ru(k+j-1)$

- **Control**

  LQR: one control solution for the whole control horizon, optimal control
  MPC: select the first control solution for predictive horizon, suboptimal control

THANKS YOU!