Get $\boldsymbol{A_d}$ $\boldsymbol{B_d}$ $\boldsymbol{g_d}$

Convert linear model to discrete-time using forward Euler method

$$\dot{\boldsymbol{x}} = \boldsymbol{A_c x} + \boldsymbol{B_c u} + \boldsymbol{g_c}$$

Linearization and discretization the model equation:

$$\begin{cases} \dot{p_x} = v\cos(\phi) \\ \dot{p_y} = v\sin(\phi) \\ \dot{\phi} = \frac{v}{L}\tan(\delta) \\ \dot{v} = a \end{cases} \qquad \begin{cases} \dot{p_x} = \bar{v}\cos(\bar{\phi}) + \cos(\bar{\phi})(v-\bar{v}) - v\sin(\bar{\phi})(\phi-\bar{\phi}) \\ \dot{p_y} = \bar{v}\sin(\bar{\phi}) + \sin(\bar{\phi})(v-\bar{v}) + v\cos(\bar{\phi})(\phi-\bar{\phi}) \\ \dot{\phi} = \frac{\bar{v}}{L}\tan(\delta) + \frac{\tan\bar{\delta}}{L}(v-\bar{v}) + \frac{\bar{v}}{L}\frac{1}{\cos^2\bar{\delta}}(\delta-\bar{\delta}) \\ \dot{v} = a \end{cases}$$

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{v} \\ \dot{\phi} \end{bmatrix} = \underbrace{\begin{pmatrix} 0 & 0 & -\bar{v}\sin\bar{\phi} & \cos\bar{\phi} \\ 0 & 0 & \bar{v}\cos\bar{\phi} & \sin\bar{\phi} \\ 0 & 0 & 0 & \frac{\tan\bar{\delta}}{L} \\ 0 & 0 & 0 & 0 \end{pmatrix}}_{\boldsymbol{A_c}} \underbrace{\begin{bmatrix} p_x \\ p_y \\ v \\ \phi \end{bmatrix}}_{\boldsymbol{x}} + \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \frac{\bar{v}}{L}\frac{1}{\cos^2\bar{\delta}} \\ 1 & 0 \end{pmatrix}}_{\boldsymbol{B_c}} \underbrace{\begin{bmatrix} a \\ \delta \end{bmatrix}}_{\boldsymbol{u}} + \underbrace{\begin{pmatrix} \bar{v}\bar{\phi}\sin\bar{\phi} \\ -\bar{v}\bar{\phi}\cos\bar{\phi} \\ 0 \\ -\frac{\bar{v}}{L}\frac{\bar{\delta}}{\cos^2\bar{\delta}} \end{pmatrix}}_{\boldsymbol{g_c}}$$

$$\frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{T_s} = \boldsymbol{A}_c\boldsymbol{x}_k + \boldsymbol{B}_c\boldsymbol{u}_k + \boldsymbol{g}_c$$

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} + T_s\boldsymbol{A}_c)\boldsymbol{x}_k + T_s\boldsymbol{B}_c\boldsymbol{u}_k + T_s\boldsymbol{g}_c$$

$$\boldsymbol{A_d} = \boldsymbol{I} + \boldsymbol{T_s A_c} \quad \boldsymbol{B_d} = \boldsymbol{T_s B_c} \quad \boldsymbol{g_d} = \boldsymbol{T_s g_c}$$

Set cost function

The position error and polar angle error of the car are required to be as small as possible in the current position and trajectory:

$$J = (x - x_{ref})^2 + (y - y_{ref})^2 + \rho(\phi - \phi_{ref})^2$$

State maxtrix $x = \begin{pmatrix} p_x \\ p_y \\ \phi \\ v \end{pmatrix}$, and $Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

So the cost funtion become:

$$J_i = (x - x_{ref})^T \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} (x - x_{ref})$$

Note that when i=horizon:

$$J_N = (x - x_{ref})^T \begin{pmatrix} \rho_N & 0 & 0 & 0 \\ 0 & \rho_N & 0 & 0 \\ 0 & 0 & \rho_N * \rho & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} (x - x_{ref})$$

Set the constrain

$$-v_{max} \leqslant v \leqslant v_{max}$$

$$-a_{max} \leqslant a \leqslant a_{max}$$

$$-\delta_{max} \leqslant \delta \leqslant \delta_{max}$$

Note that the constrain of $d\delta$:

$$-d\delta_{max} \ * dt \leq d\delta = \delta_k - \delta_{k-1} \leq d\delta_{max} \ * dt$$

- **Solving quadratic programming problems**

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_{N-1} \\ \boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{AB} & \boldsymbol{B} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^{N-1}\boldsymbol{B} & \boldsymbol{A}^{N-2}\boldsymbol{B} & \cdots & \boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_{N-1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{A}^N \end{bmatrix} \boldsymbol{x}_0$$

Extended System State:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Extended A_0:

$$A_0 = \begin{bmatrix} A_0 \\ A_1 A_0 \\ \vdots \\ \prod_{k=0}^{N-1} A_k \end{bmatrix}$$

Extended B, which contains all the control matrices in discrete system equations, stacked:

$$B = \begin{bmatrix} B_0 & 0 & \cdots & 0 & 0 \\ A_1 B_0 & B_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \prod_{k=1}^{N-1} A_k B_0 & \prod_{k=2}^{N-1} A_k B_1 & \cdots & A_{N-1} B_{N-2} & B_{N-1} \end{bmatrix}$$

Extended G, which contains all the constant terms in discrete system equations, stacked:

$$G = \begin{bmatrix} g_0 \\ A_1 g_0 + g_1 \\ \vdots \\ \sum_{n=0}^{N-2} (\prod_{k=n+1}^{N-1} A_k) g_n + g_{N-1} \end{bmatrix}$$

$$\mathbf{q}_x = -\mathbf{Q}^T \mathbf{x}_{ref}$$

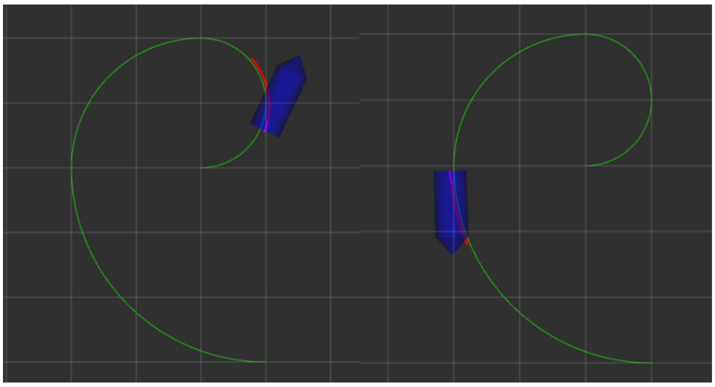
- **Get delay MPC** $\boldsymbol{new\_x_0}$

The initial state of the system can be defined as the amount of state after the current moment τ

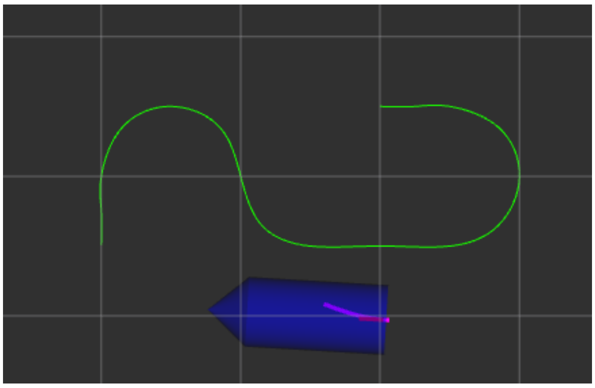
$$\overline{\boldsymbol{x}}_0 = \boldsymbol{x}(t+\tau) \approx \widehat{\boldsymbol{x}}(t+\tau) = \boldsymbol{A}^{\tau}\boldsymbol{x}(t) + \sum_{j=0}^{\tau-1} \boldsymbol{A}^j \boldsymbol{B} \underbrace{\boldsymbol{u}(t-1-j)}_{\text{Past inputs}}$$

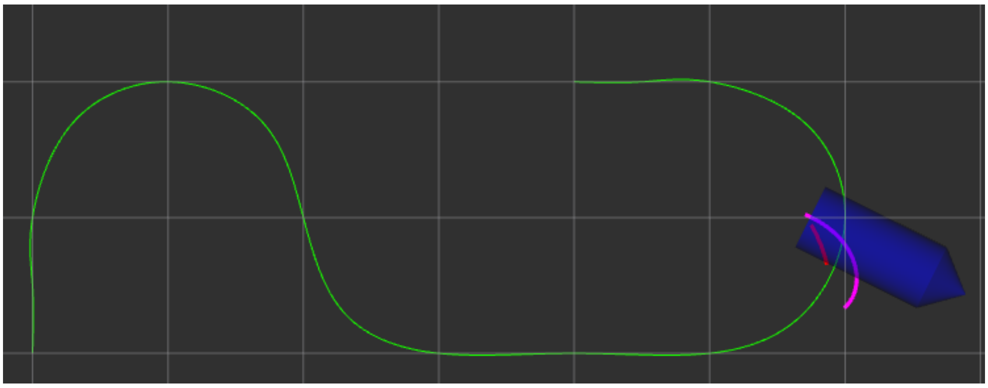
Let $N = t + \tau$ we get $new\_x_0 = x_N$, and according $X = BB * U + AA * x_0 + gg$, finally we can get new_x_0 from X

- ## Simulation result



MPC performance is affected by the initial position and yaw angle and horizon, so the other two paths perform not very well





- **Bugs:**

$$B = \begin{bmatrix} B_0 & 0 & \dots & 0 & 0 \\ A_1 B_0 & B_1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \prod_{k=1}^{N-1} A_k B_0 & \prod_{k=2}^{N-1} A_k B_1 & \dots & A_{N-1} B_{N-2} & B_{N-1} \end{bmatrix}$$

Multiplying A sequentially from right to left causes numerical instability when N_ is large. Dynamic programming from top to bottom multiplies one A at a time will not cause numerical instability