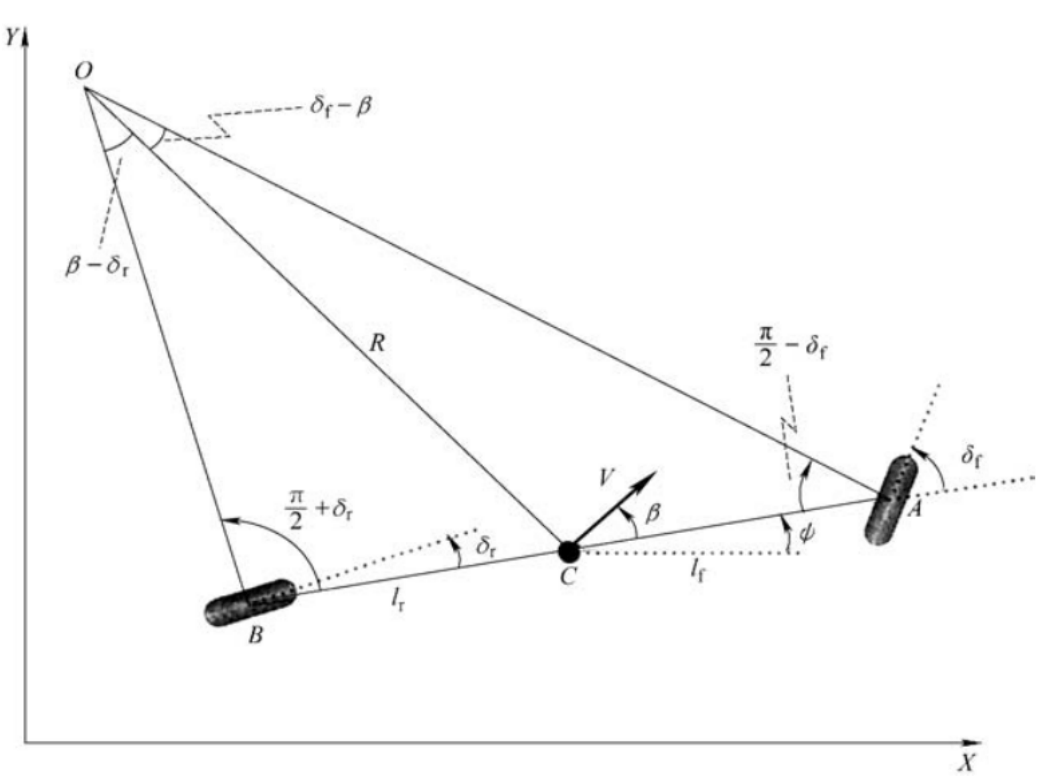
控制相关知识

基础知识

自行车模型的运动学模型推导



速时，轮胎产生的侧向力很小。为了在任意半径 R 的环形跑道上行驶，两个轮胎的侧向力之和为：

$$\frac{mv^2}{R}$$

点 O 为车辆的瞬时旋转中心，点 O 由垂直于两滚动轮方向的直线 AO 和 BO 的交点来确定。

车辆路径的半径 R 定义为连接质心 C 和瞬时旋转中心 O 的线段 OC 的长度。质心处的车速垂直于线 OC。质心处的速度方向与车辆纵轴的夹角称为车辆的侧偏角 β。

角 ψ 称为车辆的横摆角，车辆的方向角 γ 为：$\gamma=\psi+\beta$。

在三角形 OCA 上用正弦定理：

$$\frac{\sin(\delta_f-\beta)}{l_f}=\frac{\sin\left(\frac{\pi}{2}-\delta_f\right)}{R} \tag{2-1}$$

在三角形 OCB 上用正弦定理：

$$\frac{\sin(\beta-\delta_r)}{l_r}=\frac{\sin\left(\frac{\pi}{2}+\delta_r\right)}{R} \tag{2-2}$$

由式（2-1）得：

$$\frac{\sin(\delta_{f})\cos(\beta)-\sin(\beta)\cos(\delta_{f})}{l_{f}}=\frac{\cos(\delta_{f})}{R} \tag{2-3}$$

由式（2-2）得：

$$\frac{\cos(\delta_{r})\sin(\beta)-\cos(\beta)\sin(\delta_{r})}{l_{r}}=\frac{\cos(\delta_{r})}{R} \tag{2-4}$$

在式（2-3）两侧同时乘以$\frac{l_{f}}{\cos(\delta_{f})}$，可得：

$$\tan(\delta_{f})\cos(\beta)-\sin(\beta)=\frac{l_{f}}{R} \tag{2-5}$$

在式（2-4）两侧同时乘以$\frac{l_{r}}{\cos(\delta_{r})}$，可得：

$$\sin(\beta)-\tan(\delta_{r})\cos(\beta)=\frac{l_{r}}{R} \tag{2-6}$$

将式（2-5）和式（2-6）相加得：

$$\{\tan(\delta_{f})-\tan(\delta_{r})\}\cos(\beta)=\frac{l_{f}+l_{r}}{R} \tag{2-7}$$

假设车道半径由于低速而缓慢变化，则车辆方向的变化率（即：$\dot{\psi}$）必将等于车辆的角速度。由于车辆的角速度为$\frac{v}{R}$，因此有：

$$\dot{\psi}=\frac{v}{R} \tag{2-8}$$

方程（2-8）、（2-7）可整理为：

$$\dot{\psi}=\frac{v\cos(\beta)}{l_{f}+l_{r}}(\tan(\delta_{f})-\tan(\delta_{r})) \tag{2-9}$$

因此，运动的总方程为：

$$\dot{X}=v\cos(\psi+\beta) \tag{2-10}$$

$$\dot{Y}=v\sin(\psi+\beta) \tag{2-11}$$

$$\dot{\psi}=\frac{v\cos(\beta)}{l_{f}+l_{r}}(\tan(\delta_{f})-\tan(\delta_{r})) \tag{2-12}$$

在这一模型中有3个输入量：δ_{f}，δ_{r} 和 v。速度 v 为外部变量，可以假设它为时间的函数或是从纵向车辆模型中获得。

侧偏角 β 可由式（2-5）乘上 l_{r} 再减去式（2-6）乘上 l_{f} 得到：

$$\beta=\tan^{-1}\left(\frac{l_{f}\tan\delta_{r}+l_{r}\tan\delta_{f}}{l_{f}+l_{r}}\right) \tag{2-13}$$

自行车模型的动力学模型推导

考虑了二自由度车辆的“两轮”模型，如图 2-6 所示。两自由度用车辆侧向位置 y 和车辆方向角 ψ 表示。车辆的侧向位置可沿车辆横向轴到车辆旋转中心点 O 测量得到。车辆方向角 ψ 由与系统 X 轴的夹角测得。车辆在质心处的纵向速度用 v_x 表示。

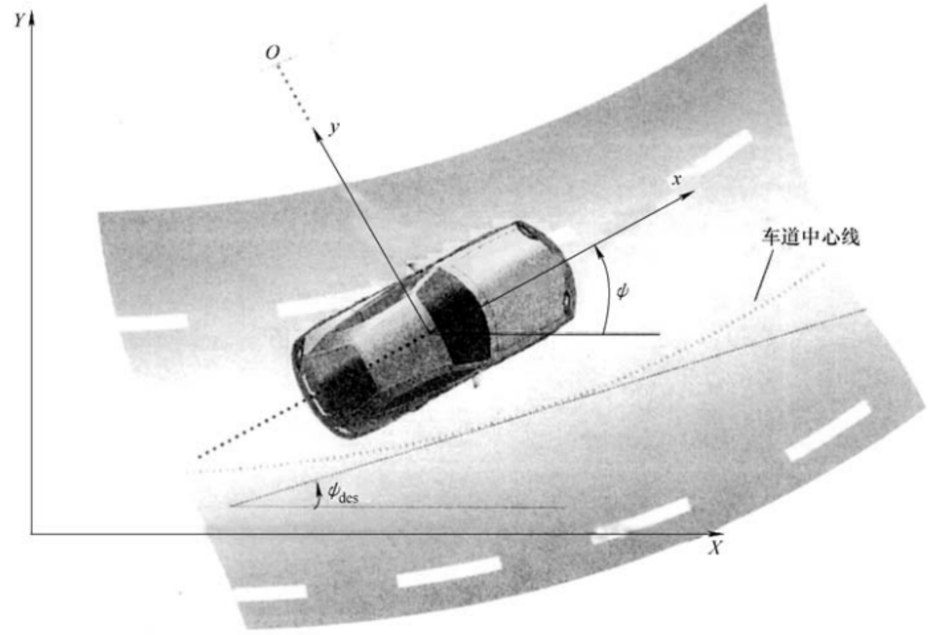


图 2-6　车辆侧向动力学

$$ma_y = F_{yf} + F_{yr} \tag{2-19}$$

式中　$a_y = \left(\dfrac{d^2y}{dt^2}\right)_{inertial}$，在 y 轴方向车辆质心处的惯性加速度；

F_{yf}和 F_{yr}——前、后轮的轮胎侧向力。

两个因素影响 a_y：沿 y 轴的运动加速度 $\ddot{y}$ 和向心加速度 $v_x\dot{\psi}$。因此：

$$a_y = \ddot{y} + v_x\dot{\psi} \tag{2-20}$$

将式（2-20）带入式（2-19），可得车辆侧向平移运动的方程为：

$$m(\ddot{y} + \dot{\psi}v_x) = F_{yf} + F_{yr} \tag{2-21}$$

绕 z 轴的转矩平衡可得到横摆动力学方程：

$$I_z\ddot{\psi} = l_fF_{yf} - l_rF_{yr} \tag{2-22}$$

式中　l_f 和 l_r——车辆质心到前轴和后轴的距离。

下一步骤是建立作用于车辆上的轮胎侧向力 F_{yf}和 F_{yr}模型。试验结果表明，当侧偏角较小时，轮胎的侧向力与“侧偏角”成正比。轮胎的侧偏角定义为轮胎平面方向和轮胎速度矢量方向之间的角度（图 2-7）。在图 2-7 中，前轮侧偏角为：

$$\alpha_f = \delta - \theta_{Vf} \tag{2-23}$$

式中　θ_{Vf}——车辆速度矢量和车辆纵轴之间的夹角；

δ——前轮转向角。

后轮侧偏角可近似表示为：

$$\alpha_r = -\theta_{Vr} \tag{2-24}$$

轮胎侧向力与侧偏角成正比的物理解释详见第 13 章（第 13.4 节）。

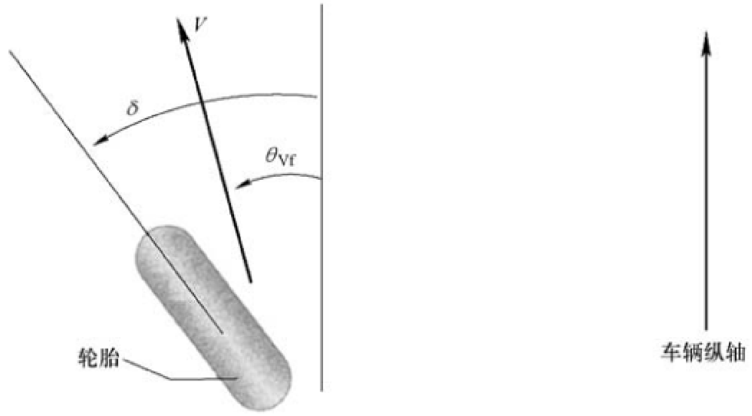


图 2-7　轮胎侧偏角

车辆的前轮侧向力可表示为：

$$F_{yf} = 2C_{\alpha f}(\delta - \theta_{Vf}) \tag{2-25}$$

式中　$C_{\alpha f}$——比例系数，指前轮的侧偏刚度；

δ——前轮转向角；

θ_{Vf}——前轮速度角。

系数 2 表示实际情况中有 2 个前轮。

同样，后轮的侧向力可表示为：

$$F_{yr} = 2C_{\alpha r}(-\theta_{Vr}) \tag{2-26}$$

式中　$C_{\alpha r}$——后轮的侧偏刚度；

θ_{Vr}——后轮速度角。

利用关系式（2-27）或式（2-28）可计算 θ_{Vf}和 θ_{Vr}：

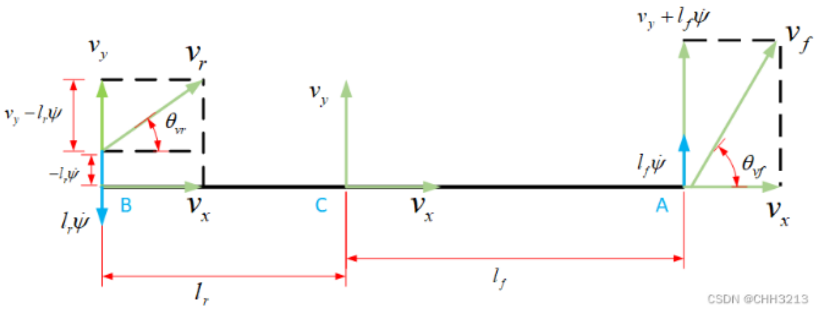
$$\tan(\theta_{Vf}) = \frac{v_y + l_f\dot{\psi}}{v_x} \tag{2-27}$$

$$\tan(\theta_{Vr}) = \frac{v_y - l_r\dot{\psi}}{v_x} \tag{2-28}$$

采用小角度的近似法及缩写 $v_y = \dot{y}$，则有：

$$\theta_{Vf} = \frac{\dot{y} + l_f\dot{\psi}}{v_x} \tag{2-29}$$

$$\theta_{Vr} = \frac{\dot{y} - l_r\dot{\psi}}{v_x} \tag{2-30}$$



点 C 代表车辆的重心， A 点和 B点到重心的距离分别用 l_f 和 l_r 表示，轴距表示为$L = l_f + l_r$。

车辆平动产生的速度分量 v_x 和 v_y ，以及绕点 C 转动产生的线速度 $l_f\dot{\psi}$ 和 $l_r\dot{\psi}$ (根据角速度与线速度的关系$\omega = \frac{v}{R}$ 得到)组成。根据上图得

$$\tan(\theta_{vf}) = \frac{v_y + l_f\dot{\psi}}{v_x} \tag{9}$$

$$\tan(\theta_{vr}) = \frac{v_y - l_r\dot{\psi}}{v_x} \tag{10}$$

v_x

将式（2-23）、式（2-24）、式（2-29）和式（2-30）代入式（2-21）和式（2-22）可得状态方程模型：

$$m\left(\ddot{y} + v_x\dot{\psi}\right) = 2C_{\alpha f}\left(\delta - \frac{\dot{y} + l_f\dot{\psi}}{v_x}\right) + 2C_{\alpha r}\left(-\frac{\dot{y} - l_r\dot{\psi}}{v_x}\right) \tag{13}$$

等式(13)左右两边同时除以 m，分别提取 $\ddot{y}$、$\dot{y}$、$\dot{\psi}$ 和 δ 项得

$$\ddot{y} = -\frac{2C_{\alpha f} + 2C_{\alpha r}}{mv_x}\dot{y} - \left(v_x + \frac{2C_{\alpha f}l_f - 2C_{\alpha r}l_r}{mv_x}\right)\dot{\psi} + \frac{2C_{\alpha f}}{m}\delta \tag{14}$$

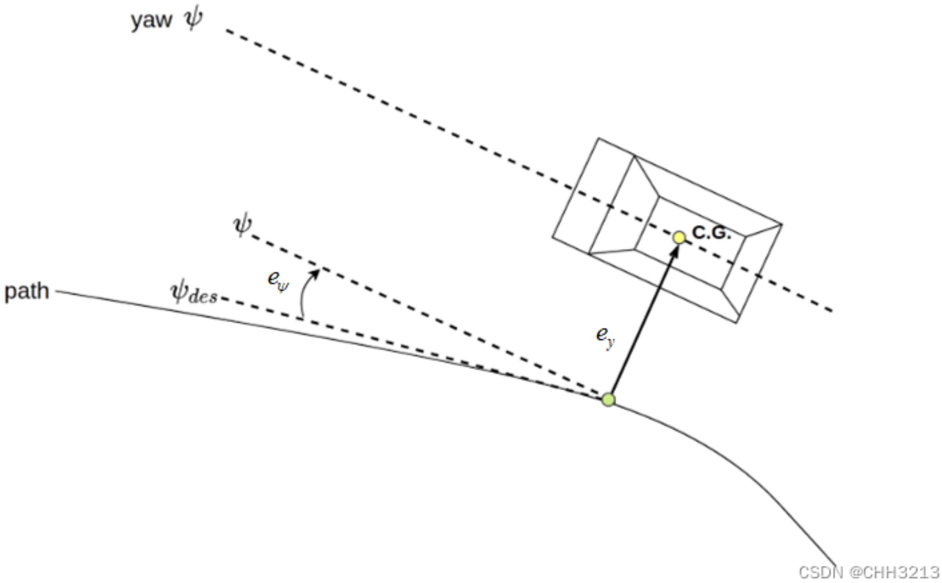
$$I_z\ddot{\psi} = 2l_fC_{\alpha f}\left(\delta - \frac{\dot{y} + l_f\dot{\psi}}{v_x}\right) - 2l_rC_{\alpha r}\left(-\frac{\dot{y} - l_r\dot{\psi}}{v_x}\right) \tag{16}$$

等式(16)左右两边同时除以 I_z，分别提取 $\dot{y}$、$\ddot{\psi}$、$\dot{\psi}$ 和 δ 项得

$$\ddot{\psi} = -\frac{2l_fC_{\alpha f} - 2l_rC_{\alpha r}}{I_zv_x}\dot{y} - \frac{2l_f{}^2C_{\alpha f} + 2l_r{}^2C_{\alpha r}}{I_zv_x}\dot{\psi} + \frac{2l_fC_{\alpha f}}{I_z}\delta \tag{17}$$

$$\begin{bmatrix} \dot{y} \\ \ddot{y} \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{mV_x} & 0 & -\left(V_x + \frac{2C_{\alpha f}l_f - 2C_{\alpha r}l_r}{mV_x}\right) \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2l_fC_{\alpha f} - 2l_rC_{\alpha r}}{I_zV_x} & 0 & -\frac{2l_f{}^2C_{\alpha f} + 2l_r{}^2C_{\alpha r}}{I_zV_x} \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2l_fC_{\alpha f}}{I_z} \end{bmatrix}\delta$$

误差模型



> 假设——
>
> - e_y :车辆重心距车道中心线的距离；
> - e_ψ :偏航角误差；
> - V_x :车辆纵向速度；
> - R：车辆转弯半径，其中转弯半径 R足够大，以满足上述章节的小角度近似假设。

定义——

- 车身转过**期望角度所需要的偏航角速度**为

$$\dot{\psi}_{des}=\frac{V_x}{R} \tag{1}$$

- 所需的**横向加速度**（即期望的向心加速度）为

$$a_{des}=\frac{V_x^2}{R}=V_x\dot{\psi}_{des} \tag{2}$$

- 车辆**偏航角误差**为

$$e_\psi=\psi-\psi_{des} \tag{3}$$

- 车辆**偏航角速度误差**为

$$\dot{e}_\psi=\dot{\psi}-\dot{\psi}_{\text{des}} \tag{4}$$

- 车辆**偏航角加速度误差**为

$$\ddot{e}_{\psi} = \ddot{\psi} - \ddot{\psi}_{\text{des}} \tag{5}$$

- 车辆 **横向加速度误差**为

$$\begin{aligned} \ddot{e}_y &= a_y - a_{des} \\ &= \left(\ddot{y} + V_x \dot{\psi}\right) - V_x \dot{\psi}_{\text{des}} \\ &= \ddot{y} + V_x \left(\dot{\psi} - \dot{\psi}_{des}\right) \end{aligned} \tag{6}$$

  其中，a_y 由两 种力共同作用产生：车辆延 y 轴产生的惯性加速度 $\ddot{y}$ 和车辆绕旋转中心 O 旋转产生的向心加速度 $a_c = \frac{{V_x}^2}{R} = V_x \dot{\psi}$。

- 车辆 **横向速度误差**为

  当车辆纵向速度V_x 恒定时， y 轴方向的速度误差可以表示为

$$\dot{e}_y = \int \ddot{e}_y \, \mathrm{d}t = \dot{y} + V_x \left(\psi - \psi_{des}\right) \tag{7}$$

误差动力学模型

将等式(6)(7)变换如下：

$$\begin{aligned} \ddot{y} &= \ddot{e}_y + V_x \dot{\psi}_{des} - V_x \dot{\psi} \\ \dot{y} &= \dot{e}_y - V_x e_{\psi} \end{aligned} \tag{9}$$

根据车辆动力学模型中的等式(14)

$$\ddot{y} = -\frac{2C_{\alpha f} + 2C_{\alpha r}}{mV_x} \dot{y} - \left(V_x + \frac{2C_{\alpha f} l_f - 2C_{\alpha r} l_r}{mV_x}\right) \dot{\psi} + \frac{2C_{\alpha f}}{m} \delta \tag{10}$$

将等式(4)(9)代入等式(10)得

$$\begin{aligned} \ddot{e}_y + V_x \dot{\psi}_{des} - V_x \left(\dot{e}_{\psi} + \dot{\psi}_{\text{des}}\right) = & -\frac{2C_{\alpha f} + 2C_{\alpha r}}{mV_x} \left(\dot{e}_y - V_x e_{\psi}\right) - \\ & \left(V_x + \frac{2C_{\alpha f} l_f - 2C_{\alpha r} l_r}{mV_x}\right) \left(\dot{e}_{\psi} + \dot{\psi}_{\text{des}}\right) + \frac{2C_{\alpha f}}{m} \delta \end{aligned} \tag{11}$$

对等式(11)提取 $\ddot{e}_y$、$\dot{e}_y$、e_y、e_{ψ}、$\dot{e}_{\psi}$、$\dot{\psi}_{des}$ 和δ项得

$$\begin{aligned} \ddot{e}_y = & \frac{-2C_{\alpha f} - 2C_{\alpha r}}{mV_x} \dot{e}_y + \frac{2C_{\alpha f} + 2C_{\alpha r}}{m} e_{\psi} + \frac{-2C_{\alpha f} l_f + 2C_{\alpha r} l_r}{mV_x} \dot{e}_{\psi} \\ & + \left(\frac{-2C_{\alpha f} l_f + 2C_{\alpha r} l_r}{mV_x} - V_x\right) \dot{\psi}_{\text{des}} + \frac{2C_{\alpha f}}{m} \delta \end{aligned} \tag{12}$$

整理成矩阵形式为

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} \dot{e}_y = & \left[\begin{array}{cccc} 0 & -\frac{2C_{\alpha f} + 2C_{\alpha r} r}{mV_x} & \frac{2C_{\alpha f} + 2C_{\alpha r}}{m} & \frac{-2C_{\alpha f} l_f + 2C_{\alpha r} l_r}{mV_x} \end{array}\right] \left[\begin{array}{c} e_y \\ \dot{e}_y \\ e_{\psi} \\ \dot{e}_{\psi} \end{array}\right] + \\ & \left(\frac{-2C_{\alpha f} l_f + 2C_{\alpha r} l_r}{mV_x} - V_x\right) \dot{\psi}_{des} + \frac{2C_{\alpha f}}{m} \delta \end{aligned} \tag{13}$$

同理可得，根据车辆动力学模型中的等式(17)

$$\ddot{\psi}=-\frac{2l_f C_{\alpha f}-2l_r C_{\alpha r}}{I_z v_x}\dot{y}-\frac{2{l_f}^2 C_{\alpha f}+2{l_r}^2 C_{\alpha r}}{I_z v_x}\dot{\psi}+\frac{2l_f C_{\alpha f}}{I_z}\delta \tag{14}$$

将等式(4)(5)(9)代入等式(14)，得

$$\ddot{e}_\psi+\ddot{\psi}_{\mathrm{des}}=-\frac{2l_f C_{\alpha f}-2l_r C_{\alpha r}}{I_z V_x}\left(\dot{e}_y-V_x e_\psi\right)-\frac{2{l_f}^2 C_{\alpha f}+2{l_r}^2 C_{\alpha r}}{I_z V_x}\left(\dot{e}_\psi+\dot{\psi}_{\mathrm{des}}\right)+\frac{2l_f C_{\alpha f}}{I_z}\delta \tag{15}$$

对等式(15)提取 $\dot{e}_y$、e_y、e_ψ、$\ddot{e}_\psi$、$\dot{e}_\psi$、$\dot{\psi}_{des}$和δ项得

$$\begin{gathered}\ddot{e}_\psi=-\frac{2l_f C_{\alpha f}-2l_r C_{\alpha r}}{I_z V_x}\dot{e}_y+\frac{2l_f C_{\alpha f}-2l_r C_{\alpha r}}{I_z}e_\psi\\-\frac{2{l_f}^2 C_{\alpha f}+2{l_r}^2 C_{\alpha r}}{I_z V_x}\dot{e}_\psi-\frac{2{l_f}^2 C_{\alpha f}+2{l_r}^2 C_{\alpha r}}{I_z V_x}\dot{\psi}_{des}+\frac{2l_f C_{\alpha f}}{I_z}\delta-\ddot{\psi}_{des}\end{gathered} \tag{16}$$

由于上述假设为线性时不变系统 $(LT1)$ $\left(\dot{V}_x=0\right)$，故 $\ddot{\psi}_{des}=\frac{\dot{V}_x}{R}=0$，将上述等式整理成矩阵形式得

$$\begin{gathered}\frac{\mathrm{d}}{\mathrm{d}t}\dot{e}_\psi=\left[\begin{array}{llll}0 & -\frac{2l_f C_{\alpha f}-2l_r C_{\alpha r}}{I_z V_x} & \frac{2l_f C_{\alpha f}-2l_r C_{\alpha r}}{I_z} & -\frac{2{l_f}^2 C_{\alpha f}+2{l_r}^2 C_{\alpha r}}{I_z V_x}\end{array}\right]\left[\begin{array}{c}e_y\\ \dot{e}_y\\ e_\psi\\ \dot{e}_\psi\end{array}\right]\\-\frac{2{l_f}^2 C_{\alpha f}+2{l_r}^2 C_{\alpha r}}{I_z V_x}\dot{\psi}_{des}+\frac{2l_f C_{\alpha f}}{I_z}\delta\end{gathered} \tag{17}$$

根据等式(13)和(17)，基于跟踪误差变量的状态空间模型表示为

$$\begin{gathered}\frac{\mathrm{d}}{\mathrm{d}t}\left[\begin{array}{c}e_y\\ \dot{e}_y\\ e_\psi\\ \dot{e}_\psi\end{array}\right]=\left[\begin{array}{cccc}0 & 1 & 0 & 0\\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{mV_x} & \frac{2C_{\alpha f}+2C_{\alpha r}}{m} & \frac{-2C_{\alpha f}l_f+2C_{\alpha r}l_r}{mV_x}\\ 0 & 0 & 0 & 1\\ 0 & -\frac{2l_f C_{\alpha f}-2l_r C_{\alpha r}}{I_z V_x} & \frac{2l_f C_{\alpha f}-2l_r C_{\alpha r}}{I_z} & -\frac{2l_f^2 C_{\alpha f}+2{l_r}^2 C_{\alpha r}}{I_z V_x}\end{array}\right]\left[\begin{array}{c}e_y\\ \dot{e}_y\\ e_\psi\\ \dot{e}_\psi\end{array}\right]\\+\left[\begin{array}{c}0\\ \frac{2C_{\alpha f}}{m}\\ 0\\ \frac{2l_f C_{\alpha f}}{I_z}\end{array}\right]\delta+\left[\begin{array}{c}0\\ \frac{-2C_{\alpha f}l_f+2C_{\alpha r}l_r}{mV_x}-V_x\\ 0\\ -\frac{2l_f^2 C_{\alpha f}+2{l_r}^2 C_{\alpha r}}{I_z V_x}\end{array}\right]\dot{\psi}_{des}\end{gathered} \tag{18}$$

写成一般形式，状态空间表达式如下:

$$\dot{X}=AX+B_1\delta+B_2\dot{\psi}_{\mathrm{des}} \tag{19}$$

在使用时，我们一般会忽略$B_2\dot{\psi}_{\mathrm{des}}$， 然后将状态方程离散化，便于实现。即:

$$X_t=X_{t-1}+AX_{t-1}\Delta t+B_1\delta\Delta t=(I+A\Delta t)X_{t-1}+B_1\delta\Delta t=\bar{A}X_{t-1}+\bar{B}\delta \tag{20}$$

其中

$$\begin{aligned}&\bar{A}=I+A\Delta t\quad(I\text{为单位矩阵})\\&\bar{B}=B_1\Delta t\end{aligned}$$

上面的离散化是用前向欧拉法进行的离散化，还可以用零阶保持法进行离散化

但因为需要使用计算机进行数值计算，推导截至到上面的连续模型还不够，还需要对上述模型进行近似离散化,近似离散化后的模型为：

$$x((k+1)T) = A_D x(kT) + B_D u(kT) + C_D$$

其中：

$$A_D = \frac{e^{\frac{AT}{2}}}{e^{-\frac{AT}{2}}} = \frac{I + \frac{AT}{2}}{I - \frac{AT}{2}} = (I - \frac{AT}{2})^{-1}(I + \frac{AT}{2})$$

$$B_D = \int_0^T e^{At} dt B = TB$$

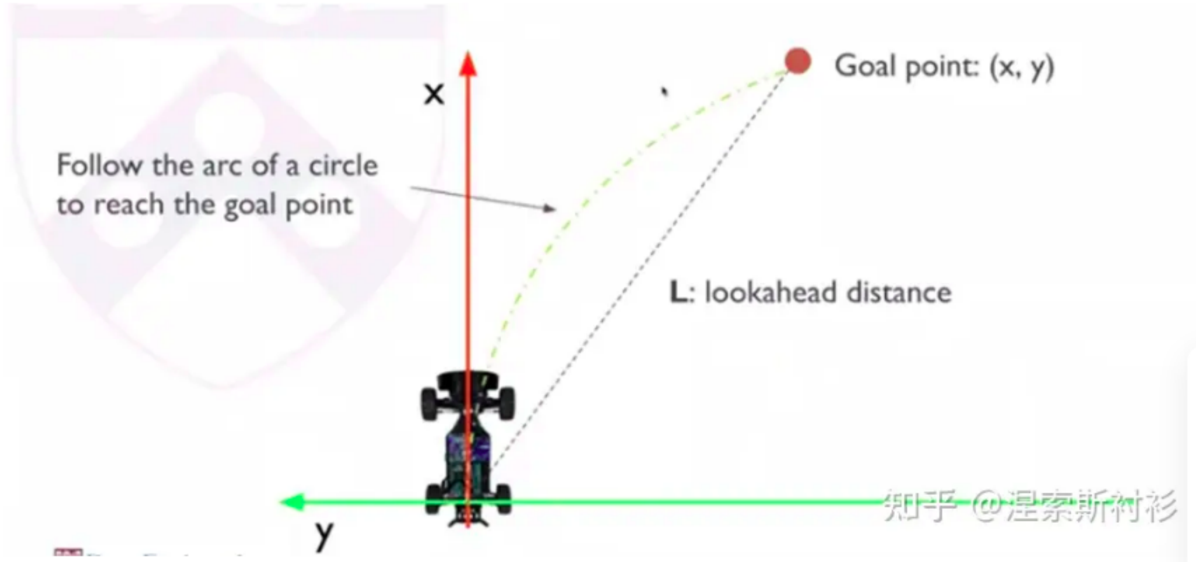
常用控制算法

Pure pursit(纯跟踪的PP控制算法)

算法理论

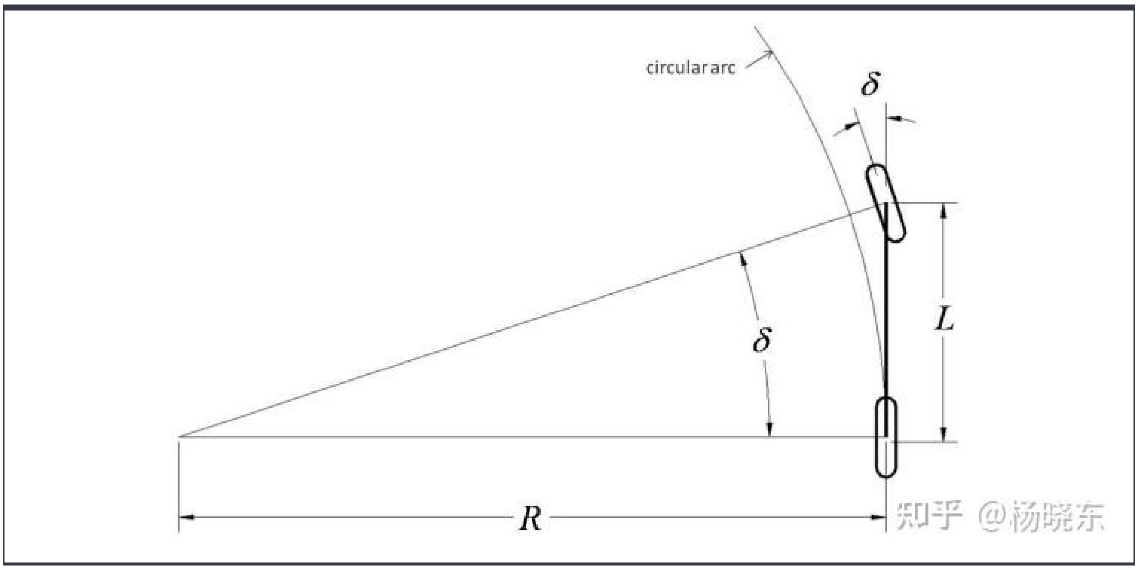
Pure Pursuit方法，顾名思义，纯轨迹跟踪。它是一种基于几何追踪的路径跟随算法，由Craig Coulter于1992年提出。因控制方法较为简单且直接，无须过多考虑车辆的运动学与动力学模型，且调节参数少，可适用于自动驾驶车辆或移动机器人中。

Pure Pursuit算法的基本思想是：参考人类驾驶的行为，通过计算车辆当前位置到预瞄点（goal point）的曲率，使车辆沿着经过预瞄点的圆弧行驶，从而实现轨迹跟踪如下图所示。因此，该算法的核心在于通过设计合理的预瞄距离，从而计算出轨迹跟踪的控制曲率。



公式推导

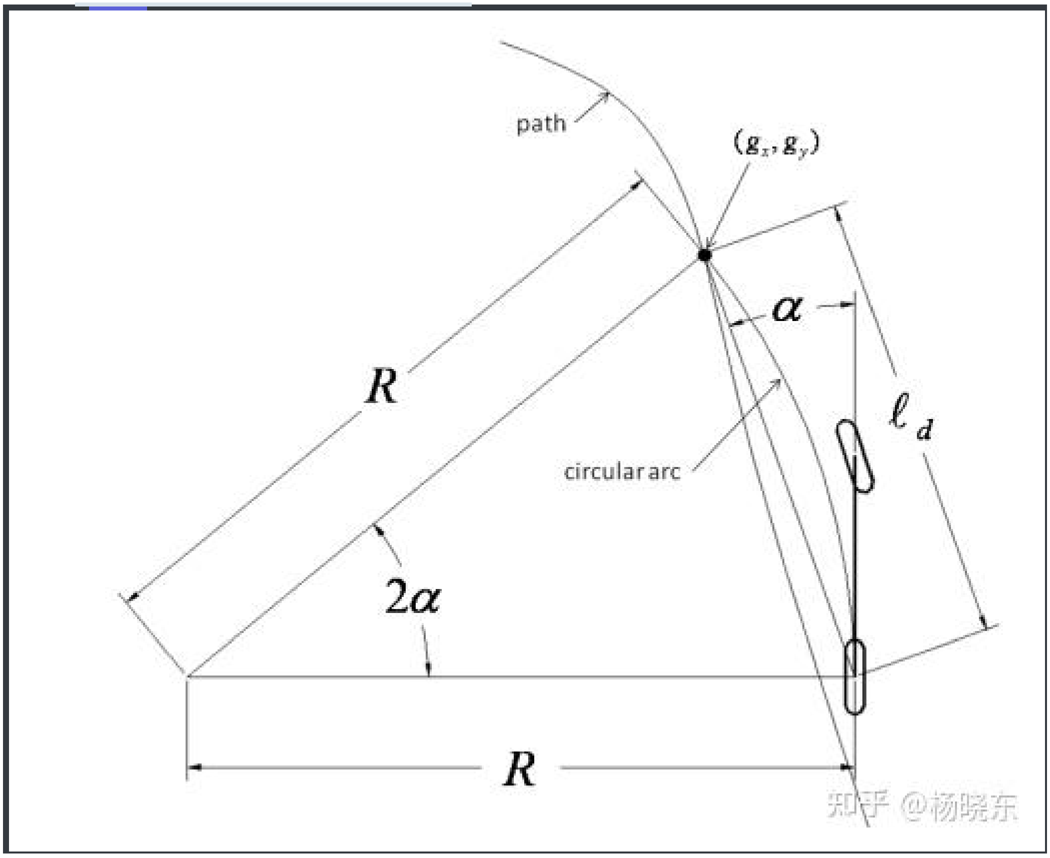
首先简单介绍下自行车控制模型，且假设汽车后轮无法转向，则形成下图单车模型

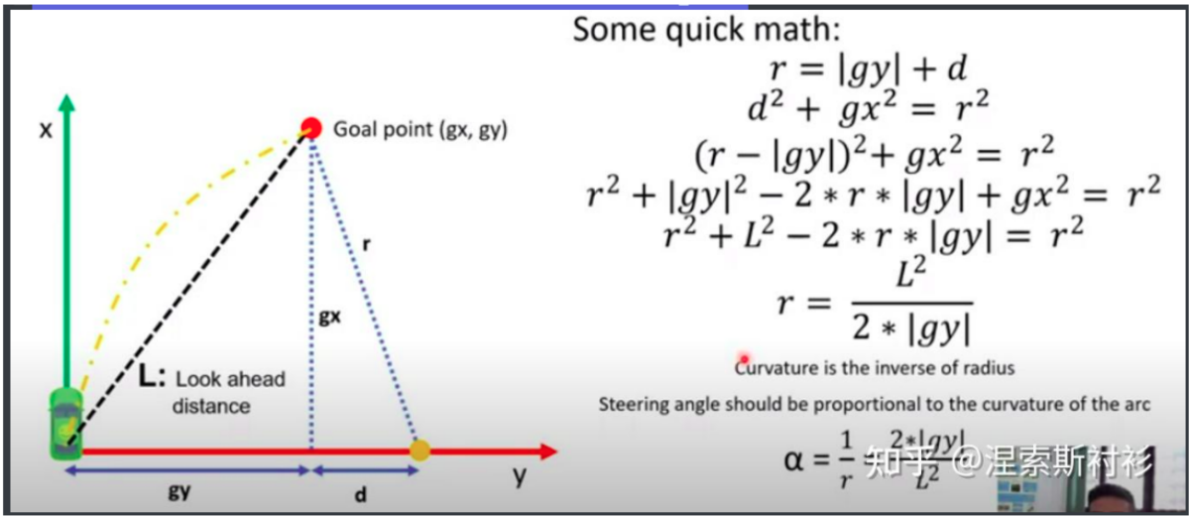


在上图中，有三角正弦定理可知(相关定理看数学文档中的笔记)

$$tan\delta = \frac{L}{R}$$

假设现在自车跟踪一条几何轨迹，如下图所示





$$\frac{sin2\alpha}{L_d} = \frac{sin(\frac{\pi}{2} - \alpha)}{R}$$

$$\frac{2sin\alpha cos\alpha}{L_d} = \frac{cos\alpha}{R}$$

$$\frac{2sin\alpha}{L_d} = \frac{1}{R}$$

将以上式子联系起来可以得到

$$\frac{2sin\alpha}{L_d} = \frac{1}{R} = \frac{tan\delta}{L}$$

$$tan\delta = \frac{2Lsin\alpha}{L_d}$$

$$sin\alpha = \frac{dy}{L_d}$$

$$tan\delta = \frac{2dy * L}{L_d^2}$$

$$\delta = arctan\frac{2dy * L}{L_d^2}$$

上式中的δ表示是前轮转角，但是前轮转角与方向盘转角之间还存在一个传动比，因此还需要乘一个传动比系数k,一般都是14.1

所以方向盘转角等于

$$steelangel = \delta * k = arctan\frac{2dy * L}{L_d^2} * k$$

demo代码

```c
#pragma section code "vLinkGenLinkerCodeSection_NullMax_Code_text"
#pragma section farrom "vLinkGenLinkerConstSection_NullMax_Const_rodata"
#include "pure_pursuit_hwp.h"
#include "NM_ControlCommom/util/angles.h"
#include "NM_ControlCommom/util/coordinate.h"
#include "NM_DrivingControl/control_common/context/control_context.h"
#include "NM_DrivingControl/control_common/type/ego_car.h"

static bool pure_pursuit_hwp_initialized = false;
static bool first_filter = false;
static float x_velocity_pp[15] = {0.f,  5.f,   10.f,  20.f,  30.f,
                                  40.f, 50.f,  60.f,  70.f,  80.f,
                                  90.f, 100.f, 110.f, 120.f, 130.f};  //速度
static float x_velocity_compensation[13] = {10.f,  20.f,  30.f, 40.f, 50.f,
                                            60.f,  70.f,  80.f, 90.f, 100.f,
                                            110.f, 120.f, 130.f};  //速度
static float x_velocity_compensation_high_curve[13] = {
    60.f, 60.f, 60.f, 60.f, 60.f, 60.f, 60.f,
    60.f, 60.f, 60.f, 60.f, 60.f, 60.f};  //速度
static float x_velocity_compensation_low_curve[13] = {
    60.f, 60.f,  60.f, 60.f, 60.f, 60.f, 60.f,
    60.f, 37.5f, 37.f, 37.f, 37.f, 37.f};  //速度
static float y_steer_angle_max_pp[15] = {
    460.f, 460.f, 460.f, 277.f, 127.f, 72.f, 46.f, 32.f,
    24.f,  18.f,  15.f,  12.f,  9.f,   7.f,  5.f};  //速度-方向盘角度
static float y_steer_rate_max_pp[15] = {
    400.f, 400.f, 350.f, 300.f, 270.f, 240.f, 210.f, 180.f,
    150.f, 120.f, 90.f,  60.f,  50.f,  40.f,  30.f};  //速度-方向盘最大角速度
const float KAckermannWheelbase = 2.8;
const float KSteeringRatio = 14.1;
const float kAngleRadianRatio = 57.3;
bool PurePursuitHwpFollowerInit(PurePursuitHwpFollower *follower) {
  //  static bool initialized = false;
  if (pure_pursuit_hwp_initialized) {
    return true;
  }
  follower->lss_scale = 2.5f;
  follower->angular_v_ratio = 1.f;
  follower->front_wheel_angle_directly_prev = false;
  follower->use_self_calibration_steer_angle = false;
  follower->front_wheel_angle_cmd = 0.f;
  FirstOrderLowPassFilterInitWithParam(
      &(follower->front_wheel_angle_cmd_filter), 0.02f, 0.1f);
  FirstOrderLowPassFilterInitWithParam(&(follower->baselink_y_filter), 0.02f,
                                       0.08f);
  FirstOrderLowPassFilterInitWithParam(&(follower->yaw_rate_cmd_filter), 0.02f,
                                       0.08f);
  // BiquadFilterInitWithParam(&(follower->yaw_rate_cmd_filter),
  //                           BIQUAD_PASS_TYPE_LOW_PASS, 0.1f, 0.11f);
  FirstOrderLowPassFilterInitWithParam(&(follower->linear_velocity_filter),
                                       0.02f, 0.005f);
  MeanFilterInit(&(follower->raw_rate_compentation_filter));
  MeanFilterInit(&(follower->steer_angle_bias_filter));
  PidControlInit(&(follower->steer_pid));
  PidControlSetGains(&(follower->steer_pid), 0.9, 0, 0);
```

```
  PidControlSetTimeInertia(&(follower->steer_pid), 0.05);
  pure_pursuit_hwp_initialized = true;
  return true;
}

bool PurePursuitHwpFollowerRunOnce(PurePursuitHwpFollower *follower) {
  ControlContext *control_context = ControlContextSingletonInstance();
  // while control reset then reset linear_velocity filter
  if (control_context->lat_controllor_reset) {
    FirstOrderLowPassFilterResetValue(
        &(follower->linear_velocity_filter),
        control_context->ego_car_state.linear_velocity);
    MeanFilterReset(&(follower->raw_rate_compentation_filter),
                    (control_context->ego_car_state.raw_angular_velocity -
                     control_context->angular_velocity_filted));
    MeanFilterReset(&(follower->steer_angle_bias_filter), 0.0f);
  }
  // for linear_vel add the filter
  follower->linear_velocity_filted = FirstOrderLowPassFilterProcessMeasurement(
      &(follower->linear_velocity_filter),
      control_context->ego_car_state.linear_velocity);
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .current_linear_velocity = follower->linear_velocity_filted;
  // and when icc active target point use icc points
  follower->target_position.x = control_context->lateral_control_point.wpa.x;
  follower->target_position.y = control_context->lateral_control_point.wpa.y;
  follower->target_position.z = control_context->lateral_control_point.wpa.r;
  follower->front_wheel_angle_directly = false;
  // cal kappa
  PurePursuitHwpFollowerCalcCurvature(follower);
  // yaw_rate = kappa * v;
  follower->yaw_rate_cmd = follower->kappa * follower->linear_velocity_filted;
  // when control use front_wheel_angle first then reset
  if (follower->front_wheel_angle_directly_prev !=
      follower->front_wheel_angle_directly) {
    first_filter = true;
  } else {
    first_filter = false;
  }
  // update prev directly
  follower->front_wheel_angle_directly_prev =
      follower->front_wheel_angle_directly;
  // cal front_wheel_angle_rate_max and front_wheel_angle_max
  PurePursuitHwpFollowerCmdLimit(follower);
  // cal yaw_cmd compentation
  PurePursuitHwpFollowerCalYawRateCompensate(follower);
  // cal yaw_rate_filter
  PurePursuitHwpFollowerFilterTwistCommand(follower);
  // cal front_wheel_angle_cmd_by_yawrate
  PurePursuitHwpFollowerSteeringControl(follower);
  // cal cmd_front_wheel_angle to control
  PurePursuitHwpFollowerSteeringPostProcess(follower);
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor
      .front_wheel_angle_directly = follower->front_wheel_angle_directly;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.yaw_rate_cmd =
      follower->yaw_rate_cmd;
```

```
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.target_position_x =
      follower->target_position.x;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.target_position_y =
      follower->target_position.y;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.target_position_z =
      follower->target_position.z;
  return true;
}

void PurePursuitHwpFollowerCmdLimit(PurePursuitHwpFollower *follower) {
  ControlContext *control_context = ControlContextSingletonInstance();
  // from the vel interpolate the max front_wheel_angle and the max
  // front_wheel_angle_rate
  follower->front_wheel_angle_max =
      Hwp_Interpolate(x_velocity_pp, y_steer_angle_max_pp,
                      follower->linear_velocity_filted * 3.6f) /
      KSteeringRatio / kAngleRadianRatio;
  follower->front_wheel_angle_rate_max =
      Hwp_Interpolate(x_velocity_pp, y_steer_rate_max_pp,
                      follower->linear_velocity_filted * 3.6f) /
      KSteeringRatio / kAngleRadianRatio / 50.f;
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .cmd_front_wheel_angle_curvature_max = follower->front_wheel_angle_max;
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .cmd_front_wheel_angle_lateral_errorP_max =
      follower->front_wheel_angle_rate_max;
}

void PurePursuitHwpFollowerCalcCurvature(PurePursuitHwpFollower *follower) {
  ControlContext *control_context = ControlContextSingletonInstance();
  Point3Vector predict_odom_point;
  // cmd send util run need time delay
  const float time_delay = control_context->control_calibration_parameter
                               .calibration_lateral.target_delay;
  // delay distance = v * time_delay
  const float distance_delay = follower->linear_velocity_filted * time_delay;
  // delay yaw = w * time_delay
  const float yaw_delay =
      control_context->ego_car_state.angular_velocity * time_delay;
  // base bicycle model to cal after time_delay the odom point
  predict_odom_point.x =
      control_context->odom_pose_point.x +
      distance_delay *
          cosf(control_context->odom_pose_point.z + 0.5f * yaw_delay);
  predict_odom_point.y =
      control_context->odom_pose_point.y +
      distance_delay *
          sinf(control_context->odom_pose_point.z + 0.5f * yaw_delay);
  predict_odom_point.z =
      normalize_angle(control_context->odom_pose_point.z + yaw_delay);
  // base bicycle model
  // kappa = (2 * y) / (ld * ld)
  follower->denominator =
      Distance2DSquare(follower->target_position.x, follower->target_position.y,
                       predict_odom_point.x, predict_odom_point.y);
  // convert to baselink coordinate to cal y
```

```
  Hwp_ConvertPointToRelativeCoordinate(&(follower->target_position),
                                       &predict_odom_point,
                                       &(follower->target_position_baselink));
  if (control_context->lat_controllor_reset) {
    FirstOrderLowPassFilterResetValue(&(follower->baselink_y_filter), 0.0);
  }
  follower->target_position_baselink.y =
      FirstOrderLowPassFilterProcessMeasurement(
          &(follower->baselink_y_filter), follower->target_position_baselink.y);
  follower->target_position_baselink.y =
      fmaxf(follower->target_position_baselink.y, -kMaxBaselinkY);
  follower->target_position_baselink.y =
      fminf(follower->target_position_baselink.y, kMaxBaselinkY);
  follower->numerator = 2.f * follower->target_position_baselink.y;
  // ld > sqrt(10m)
  if (follower->denominator > kEpsilonDistanceHwpPP) {
    float kp = 1.f;
    follower->kappa = kp * follower->numerator / follower->denominator;
    follower->kappa = fmaxf(follower->kappa, -0.02f);
    follower->kappa = fminf(follower->kappa, 0.02f);
  } else {
    follower->kappa = 0.f;
  }
  // monitor-predict odom
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .predict_point.x = predict_odom_point.x;
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .predict_point.y = predict_odom_point.y;
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .predict_point.z = predict_odom_point.z;
  // monitor-odom
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .current_point.x = control_context->odom_pose_point.x;
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .current_point.y = control_context->odom_pose_point.y;
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .current_point.z = control_context->odom_pose_point.z;
  // add runonce result to monitor
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor
      .target_position_baselink_x = follower->target_position_baselink.x;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor
      .target_position_baselink_y = follower->target_position_baselink.y;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.numerator =
      follower->numerator;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.denominator =
      follower->denominator;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.kappa =
      follower->kappa;
}
void PurePursuitHwpFollowerCalYawRateCompensate(
    PurePursuitHwpFollower *follower) {
  ControlContext *control_context = ControlContextSingletonInstance();
  follower->use_self_calibration_steer_angle = false;
  float angular_compensation_velocity = 1.0f;
  // if (follower->kappa >= 0.002f) {  // 500m
  //   angular_compensation_velocity = Hwp_Interpolate(
```

```
  //       x_velocity_compensation, x_velocity_compensation_high_curve,
  //       control_context->ego_car_state.display_velocity *
  //           3.6f);  //大曲率速度差值
  // } else {
  //   angular_compensation_velocity = Hwp_Interpolate(
  //       x_velocity_compensation, x_velocity_compensation_low_curve,
  //       control_context->ego_car_state.display_velocity *
  //           3.6f);  //小曲率速度差值
  // }
  angular_compensation_velocity = Hwp_Interpolate(
      x_velocity_compensation, x_velocity_compensation_low_curve,
      control_context->ego_car_state.display_velocity * 3.6f);  //曲率速度差值
  if (control_context->control_calibration_parameter.calibration_pp_lateral
          .angular_compensation_velocity <= 150.0f) {
    angular_compensation_velocity =
        control_context->control_calibration_parameter.calibration_pp_lateral
            .angular_compensation_velocity;
  }
  if (control_context->control_input.trick_vars.in_curvature) {
    follower->yaw_rate_cmd =
        follower->yaw_rate_cmd *
        (1.0f +
         (follower->linear_velocity_filted * follower->linear_velocity_filted) /
             (angular_compensation_velocity * 45.0f));
  }
  float raw_rate_compensate = 0.0f;
  if ((control_context->control_input.trick_vars.in_curvature ||
       control_context->control_input.trick_vars.changing_status ==
           LANE_CHANGING_STATUS_CHANGING ||
       control_context->control_input.trick_vars.changing_status ==
           LANE_CHANGING_STATUS_ABORT) &&
      control_context->is_calibration_success) {
    follower->use_self_calibration_steer_angle = true;
    raw_rate_compensate = 0.0f;
  } else {
    raw_rate_compensate = control_context->ego_car_state.raw_angular_velocity -
                          control_context->angular_velocity_filted;
  }
  raw_rate_compensate = MeanFilterProcessMeasurement(
      &(follower->raw_rate_compentation_filter), raw_rate_compensate);
  follower->yaw_rate_cmd_compensation =
      follower->yaw_rate_cmd + raw_rate_compensate;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.in_curvature =
      control_context->control_input.trick_vars.in_curvature;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor
      .yaw_rate_cmd_filted_ratio = follower->yaw_rate_cmd_compensation;
}

void PurePursuitHwpFollowerFilterTwistCommand(
    PurePursuitHwpFollower *follower) {
  ControlContext *control_context = ControlContextSingletonInstance();
  // reset angular v filter
  if (control_context->lat_controllor_reset) {
    FirstOrderLowPassFilterResetValue(
        &(follower->yaw_rate_cmd_filter),
        control_context->ego_car_state.raw_angular_velocity);
```

```
  }
  FirstOrderLowPassFilterSetParam(
      &(follower->yaw_rate_cmd_filter),
      control_context->control_calibration_parameter.calibration_lateral
          .cmd_front_wheel_angle_filt_coef);
  follower->yaw_rate_cmd_filted = FirstOrderLowPassFilterProcessMeasurement(
      &(follower->yaw_rate_cmd_filter), follower->yaw_rate_cmd_compensation);
  LimitYawRateCmd(follower);
}
void LimitYawRateCmd(PurePursuitHwpFollower *follower) {
  ControlContext *control_context = ControlContextSingletonInstance();
  // 限值yaw_rate_cmd_filted变化率
  static float lowpass_angular_z = 0.;
  static bool prev_pub_lat_cmd_pp = false;
  float maximum_delta_angular;
  if (!control_context->control_input.trick_vars.in_curvature &&
      follower->linear_velocity_filted < 5.0) {
    maximum_delta_angular = kMaxStepDeltaAngular;
  } else {
    maximum_delta_angular = 0.004;
  }
  if (control_context->pub_lat_cmd && false == prev_pub_lat_cmd_pp) {
    lowpass_angular_z = control_context->ego_car_state.raw_angular_velocity;
  }
  prev_pub_lat_cmd_pp = control_context->pub_lat_cmd;

  float angular = follower->yaw_rate_cmd_filted;
  if (angular > lowpass_angular_z) {
    angular = fminf(angular, lowpass_angular_z + maximum_delta_angular);
  } else if (angular < lowpass_angular_z) {
    angular = fmaxf(angular, lowpass_angular_z - maximum_delta_angular);
  }
  if (!control_context->pub_lat_cmd) {
    lowpass_angular_z = control_context->ego_car_state.raw_angular_velocity;
  } else {
    lowpass_angular_z = angular;
  }
  follower->yaw_rate_cmd_filted = angular;
  // 限制微小摆动
  if (control_context->control_input.trick_vars.in_curvature == false &&
      control_context->control_input.trick_vars.changing_status !=
          LANE_CHANGING_STATUS_CHANGING) {
    // low speed filter larger angular noise
    if (fabs(follower->yaw_rate_cmd_filted) <= kMaxYawRateCmdLimit) {
      follower->yaw_rate_cmd_filted = 0.;
    }
  } else if (fabs(follower->yaw_rate_cmd_filted) <= kMaxYawRateCmdLimit) {
    follower->yaw_rate_cmd_filted = 0.;
  }
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.yaw_rate_cmd_filted =
      follower->yaw_rate_cmd_filted;
}

void PurePursuitHwpFollowerSteeringControl(PurePursuitHwpFollower *follower) {
  ControlContext *control_context = ControlContextSingletonInstance();
  static float last_target_steering_angle = 0.0f;
```

```
  float target_steering_angle = 0.f;
  if (control_context->lat_controllor_reset || !control_context->pub_lat_cmd) {
    last_target_steering_angle = 0.0f;
  }
  if (follower->linear_velocity_filted > 1.2) {
    target_steering_angle =
        KSteeringRatio *
        atanf(KAckermannWheelbase * follower->yaw_rate_cmd_filted /
              follower->linear_velocity_filted);
  } else {
    target_steering_angle = last_target_steering_angle;
  }
  last_target_steering_angle = target_steering_angle;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor
      .target_steering_angle = target_steering_angle;

  float current_steering_angle = 0.f;
  float radius = 0.f;

  // cal current steer angle
  radius = control_context->ego_car_state.linear_velocity /
           control_context->angular_velocity_filted;
  if (fabsf(radius) > 0.001f) {
    current_steering_angle =
        KSteeringRatio * atan(KAckermannWheelbase / radius);
  }
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor
      .current_steering_angle = current_steering_angle;
  control_context->mcu_to_soc_ctrl_monitor.mcu_to_soc_lat_ctrl_monitor
      .current_steering_wheel_angle =
      control_context->ego_car_state.steering_wheel_angle;
  float steering_err = target_steering_angle - current_steering_angle;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor.steering_err =
      steering_err;
  follower->front_wheel_angle_cmd_by_yawrate =
      target_steering_angle / KSteeringRatio;
}

void PurePursuitHwpFollowerSteeringPostProcess(
    PurePursuitHwpFollower *follower) {
  ControlContext *control_context = ControlContextSingletonInstance();
  // from v to control
  if (true == follower->front_wheel_angle_directly) {
    follower->front_wheel_angle_cmd = follower->front_wheel_angle_cmd_directly;
  } else {
    follower->front_wheel_angle_cmd =
        follower->front_wheel_angle_cmd_by_yawrate;
  }
  if (control_context->lat_controllor_reset || first_filter) {
    FirstOrderLowPassFilterResetValue(
        &(follower->front_wheel_angle_cmd_filter),
        control_context->ego_car_state.steering_wheel_angle / KSteeringRatio);
  }
  follower->front_wheel_angle_cmd =
      fminf(follower->front_wheel_angle_cmd, follower->front_wheel_angle_max);
  follower->front_wheel_angle_cmd =
```

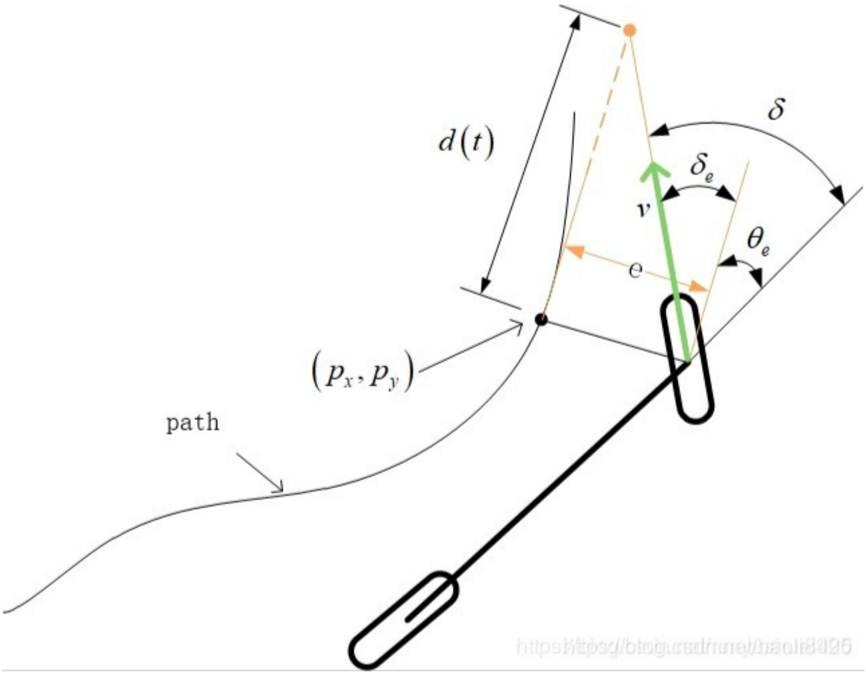
```
      fmaxf(follower->front_wheel_angle_cmd, -follower->front_wheel_angle_max);
  if (!control_context->pub_lat_cmd) {
    follower->front_wheel_angle_cmd =
        control_context->ego_car_state.steering_wheel_angle;
  }
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor
      .front_wheel_angle_cmd = follower->front_wheel_angle_cmd * KSteeringRatio;
  // cmd_front_angle filter
  // go in icc no large steer
  follower->front_wheel_angle_cmd_filter_coef +=
      control_context->control_calibration_parameter.calibration_pp_lateral
          .curve_scale;
  // 0.001f;
  follower->front_wheel_angle_cmd_filter_coef =
      fminf(follower->front_wheel_angle_cmd_filter_coef, 0.2f);
  FirstOrderLowPassFilterSetParam(&(follower->front_wheel_angle_cmd_filter),
                                  follower->front_wheel_angle_cmd_filter_coef);
  follower->front_wheel_angle_cmd_filted =
      FirstOrderLowPassFilterProcessMeasurementWithStepBound(
          &(follower->front_wheel_angle_cmd_filter),
          follower->front_wheel_angle_cmd,
          follower->front_wheel_angle_rate_max);
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor
      .front_wheel_angle_cmd_filted =
      follower->front_wheel_angle_cmd_filted * KSteeringRatio;
  float steer_angle_bias = 0.0f;
  if (follower->use_self_calibration_steer_angle) {
    steer_angle_bias = control_context->front_wheel_bias / KSteeringRatio;
  } else {
    steer_angle_bias = 0.0f;
  }
  steer_angle_bias = MeanFilterProcessMeasurement(
      &(follower->steer_angle_bias_filter), steer_angle_bias);
  control_context->cmd_front_wheel_angle =
      follower->front_wheel_angle_cmd_filted + steer_angle_bias;
  control_context->mcu_to_soc_ctrl_monitor.lat_pp_monitor
      .cmd_front_wheel_angle =
      control_context->cmd_front_wheel_angle * KSteeringRatio;
}

#pragma section code restore
#pragma section farrom restore
```

Purea pursit-Stanely

算法理论

Stanely控制算法是一种基于横向跟踪误差(cross-track error)的控制方法，能够实线横向跟踪误差指数收敛于0.



Stanely控制算法的主要思想就是根据自车位姿以及给定的几何路径，计算与自车最近的几何路径点，根据该路径点的朝向以及与自车之间的横向误差计算自车跟随轨迹所需要的航向角。

$$\delta(t) = \delta_e(t) + \delta_{\theta_e}(t)$$

式中$\delta(t)$表示前轮转角，$\delta_e(t)$表示根据横向误差计算出来的前轮转角，$\delta_{\theta_e}(t)$表示最近路径点对应的航向角

公式推导

首先在不考虑横向跟踪误差e的情况下，前轮偏角应该和给定路径切线方向是一致的，如上图所示。其中θ_e表示车辆航向与最近路径点切线方向之间的夹角，在没有任何横向误差的情况下，前轮的方向与所在的路径点的方向应该是相同的，因此

$$\delta_{\theta_t}(t) = \theta_e(t)$$

在不考虑航向跟踪误差θ_e的情况下，横向跟踪误差越大，前轮转向角越大，假设车辆预期轨迹在距离前轮$d(t)$处与给定路径上最近点切线相交，根据几何关系得出如下非线性比例函数：

$$\delta_e(t) = arctan\frac{e(t)}{d(t)} = arctan\frac{ke(t)}{v(t)}$$

其中$d(t)$与车速相关，最后用车速$v(t)$，增益系数k表示。随着横向误差的增加，$arctan$函数产生一个直接指向期望路径的前轮偏角，并且收敛速度受车速$v(t)$限制

因此Stanely控制算法的转角控制率如下：

$$\delta(t) = \theta_e(t) + arctan\frac{ke(t)}{v(t)}$$

误差推导

使用线性自行车运动模型，可以得到横向误差的变化率

$$\dot{e}(t) = -v(t)sin\delta_e(t)$$

其中

$$sin\delta_e(t) = \frac{e(t)}{\sqrt{d(t)^2 + e(t)^2}} = \frac{ke(t)}{\sqrt{v(t)^2 + (ke(t))^2}}$$

$$\dot{e}(t) = \frac{-v(t)ke(t)}{\sqrt{v(t)^2 + (ke(t))^2}} = \frac{-ke(t)}{\sqrt{1 + (\frac{ke(t)}{v(t)})^2}}$$

当横向跟踪误差$e(t)$很小时，$(\frac{ke(t)}{v(t)})^2 - > 0$

所以$\dot{e}(t) \approx -ke(t)$

通过积分上式：$e(t) = e(0)$\crosse$e^{-kt}$

因此横向误差指数收敛与e(t)=0,参数k决定了收敛速度，对于任意横向误差，微分方程都单调的收敛到0

PID控制算法

LQR控制算法

介绍

如果所研究的系统是线性的，且性能指标为状态变量和控制变量的二次型函数，则最优控制问题称为线性二次型问题。LQR，Linear Quadratic Regulator，即线性二次型调节器， 是求解线性二次型问题常用的求解方法。LQR，其对象是现代控制理论中以状态空间形式给出的线性系统，而目标函数为对象状态和控制输入的二次型函数。LQR最优设计是指设计出的状态反馈控制器 K要使二次型目标函数J 取最小值，而 K由权矩阵Q 与 R 唯一决定，故此 Q、 R 的选择尤为重要。LQR理论是现代控制理论中发展最早也最为成熟的一种状态空间设计法。特别可贵的是，LQR可得到状态线性反馈的最优控制规律，易于构成闭环最优控制。

> **基于动力学模型的LQR前提假设：**
>
> 1，小角度转向，故规划路径的曲率不能变化过快；
> 2，认为车速恒定，故加速度不能过大或过小，加速度尽可能小；
>
> **LQR算法的输入：**
>
> **1，整车参数：用于求解LQR模块所需的A,B矩阵**
>
> **2，车辆实时状态参数：速度，质心坐标，航向角，方向盘转角**
>
> **3，规划的离散轨迹点参数：坐标，航向角，曲率**

LQR算法的输出

$$\delta = -Kx + \delta_{ff}$$

LQR横向控制算法主要有3个关键模块:

1，LQR模块：求解反馈矩阵K;

2，横向误差计算模块：求解误差的状态变量Error;

3，前馈模块：补偿道路曲率对稳态误差的影响，消除稳态误差。

反馈矩阵K的求解

连续时间计算方式

我们求解的前提是假定系统处于稳定状态，此时的状态反馈为$u(t)=-Kx(t)$
将状态方程代入到代价函数中
状态方程：$\dot{x}(t)=(A-BK)x(t)$
代价函数：

$$
\begin{aligned}
J &= \int_0^{\infty}(x^TQx+u^TRu)dt \\
&= \int_0^{\infty}x^T(t)(Q+K^TRK)x(t)dt
\end{aligned}
$$

我们要找到这个积分的原函数，为了找到K，假设存在一个常量矩阵P，使得

$$\frac{d}{dt}(x(t)^TPx(t))=-x^T(t)(Q+K^TRK)x(t) \tag{8}$$

对式(8)两边取微分

$$\dot{x}^T(t)Px(t)+x^T(t)P\dot{x}(t)+x^T(t)Qx(t)+x^T(t)K^TRKx(t)=0 \tag{9}$$

将状态方程代入(9):

$$x^T(t)((A-BK)^TP+P(A-BK)+Q+K^TRK)x(t)=0$$

这个式子要成立的话，括号里的项必须为恒为零。

$$(A-BK)^TP+P(A-BK)+Q+K^TRK=0$$

进一步化简：

$$A^TP+PA+Q+K^TRK-K^TB^TP-PBK=0$$

进一步化简：

$$A^TP+PA+Q+K^TRK-K^TB^TP-PBK=0$$

取$K=R^{-1}B^TP$，代入上式得：

$$A^TP+PA+Q-PBR^{-1}B^TP=0 \tag{10}$$

K的二次项没有了，可K的取值和P有关，而P是我们假设的一个量，P只要使得的(10)式成立就行了。而(10)式在现代控制理论中极其重要，它就是有名的Riccati 方程。

3.3.3 计算$K=R^{-1}B^TP$

上个步骤中，可以利用工具箱求得矩阵P，最后根据公式计算出反馈矩阵K即可。

离散时间计算方式

计算原理与连续时间基本一致，目的是使目标函数极小

但是离散时间的计算方式需要按照离散时间去推导，推导方式为动态规划法：

求解过程

关于 LQR 的求解过程，可以采用动态规划算法，依据上述公式 (18) 的递归关系，反向递推，求出满足一定条件的最小代价值。

1. 确定迭代范围 N
2. 设置迭代初始值 $P_N = Q_f$
3. 循环迭代，$t = N, \ldots, 1$

$$P_{t-1} = Q + A^T P_{t+1} A - A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$$

4. 则反馈系数 $K_t = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$，对于时间 $t = 0, \ldots, N-1$
5. 优化的控制量 $u_t^{lqr} = K_t x_t$

推导过程：

推导教程链接

值函数

首先定义一个值函数 $V_t : R^n \to R$, 其中 $t = (0, \ldots, N)$：

$$V_t(z) = \min_{u_t, \ldots, u_{N-1}} \left(\sum_{\tau=t}^{N-1} (x_\tau^T Q x_\tau + u_\tau^t R u_\tau) + x_N^T Q_f x_N \right) \tag{8}$$

如果设置 $x_t = z$，根据公式 (1) 的关系，$x_{\tau+1} = A x_\tau + B u_\tau$, 并且 $\tau = t, \ldots, N$。

- $V_t(z)$ 可以表示在 t 时刻，从状态 z 开始的 LQR 最小代价值
- $V_0(x_0)$ 表示在 0 时刻，从状态 x_0 开始的 LQR 最小代价值

V_t 可以表示为二次型的形式，即 $V_T(z) = z^T P_t z$, 其中 $P_t = P_t^T \geq 0$。当 $t = N$ 时，代价值函数为：

$$V_N(z) = z^T Q_f z \tag{9}$$

因此 $P_N = Q_f$。

根据动态规划原理，等式 (8) 可以写成如下递归关系式：

$$V_t(z) = \min_w \left(z^T Q z + w^T R w + V_{t+1}(Az + Bw) \right) \tag{10}$$

其中，

- $z^T Q z + w^T R w$：如果 $u_t = w$, 则代表 t 时刻产生的代价值；
- $V_{t+1}(Az + Bw)$：代表从 $t+1$ 时刻开始，引起的最小代价值；

提取等式 (10) 中与 w 无关的选项得

$$V_t(z) = z^T Q z + \min_w \left(w^T R w + V_{t+1}(Az + Bw) \right) \tag{11}$$

等式 (11) 描述了 $V_t(z)$ 与 $V_{t+1}(z)$ 之间的递归关系。

最优控制率 u_t 可以表示为

$$u_t^{lqr} = arg \min_w \left(w^T R w + V_{t+1}(Az + Bw) \right)$$

求极值

假设 $V_{t+1} = z^T P_{t+1} z$, 并且 $P_{t+1} = P_{t+1}^T \geq 0$, 等式 (13) 可以进一步转化为 P_{t+1} 的形式:

$$V_t(z) = z^T Q z + \min_w \left(w^T R w + (Az + Bw)^T P_{t+1} (Az + Bw) \right) \tag{12}$$

为了求最小值，对 w 求导，导数为零的点即为最值点。

$$2w^T R + 2(Az + Bw)^T P_{t+1} B = 0 \tag{13}$$

推导等式 (13)，求取 w:

$$\begin{aligned}
w^T R + z^T A^T P_{t+1} B + w^T B^T P_{t+1} B &= 0 \\
w^T (R + B^T P_{t+1} B) &= -z^T A^T P_{t+1} B && \text{(合并同类项并移项)} \\
(R + B^T P_{t+1} B)^T w &= -B^T P_{t+1}^T A z && \text{(转置)} \\
(R + B^T P_{t+1} B) w &= -B^T P_{t+1} A z && (P_{t+1} = P_{t+1}^T, R = R^T) \\
w &= -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A z && \text{(矩阵求逆)}
\end{aligned} \tag{14}$$

由等式 (14) 可知，最优输入为

$$w^* = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A z \tag{15}$$

将等式 (15) 带入等式 (12) 得

$$V_t(z) = z^T Q z + w^{*T} R w^* + (Az + Bw^*)^T P_{t+1} (Az + Bw^*) \tag{16}$$

对等式 (16) 化简得

$$\begin{aligned}
V_t(z) &= z^T Q z + w^{*T} R w^* + (Az + Bw^*)^T P_{t+1} (Az + Bw^*) \\
&= z^T Q z + w^{*T} R w^* + z^T A^T P_{t+1} A z + 2 z^T A^T P_{t+1} B w^* + w^{*T} B^T P_{t+1} B w^* \\
&= z^T Q z + z^T A^T P_{t+1} A z + w^{*T} (R + B^T P_{t+1} B) w^* + 2 z^T A^T P_{t+1} B w^* \\
&= z^T Q z + z^T A^T P_{t+1} A z \\
&\quad + z^T A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} (R + B^T P_{t+1} B) (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A z \\
&\quad - 2 z^T A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A z \\
&= z^T Q z + z^T A^T P_{t+1} A z \\
&\quad + z^T A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A z \\
&\quad - 2 z^T A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A z \\
&= z^T Q z + z^T A^T P_{t+1} A z - z^T A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A z \\
&= z^T (Q + A^T P_{t+1} A - A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A) z \\
&= z^T P_t z
\end{aligned} \tag{17}$$

上述公式化简过程中，由于 $P_{t+1} = P_{t+1}^T, R = R^T$，所以 $\left((R + B^T P_{t+1} B)^{-1}\right)^T = (R + B^T P_{t+1} B)^{-1}$。

由等式 (17) 可知

$$P_t = Q + A^T P_{t+1} A - A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A \tag{18}$$

横向误差计算

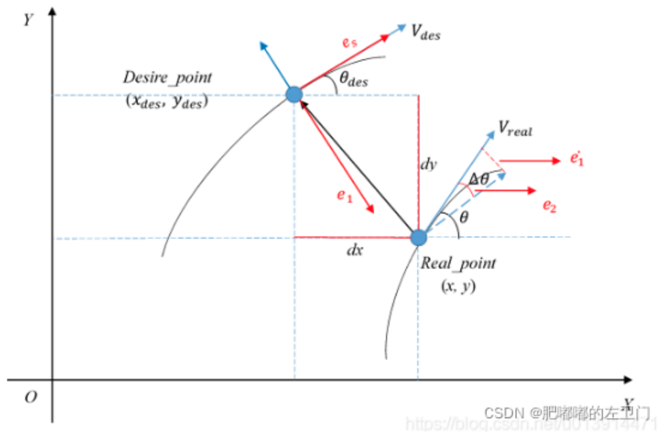
二，横向误差计算模块：求解误差状态变量Error;

$$x = \begin{bmatrix} cte \\ \dot{cte} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

4个元素分别为：
1，横向位置误差lateral_error（重点及难点）；
2，横向位置误差变化率lateral_error_rate；
3，航向误差heading_error；
3，航向误差变化率heading_error_rate；

横向位置偏差

横纵向位置偏差跟期望目标点的方向有关，故工程应用中均是在Frenet坐标系下求取横纵向位置偏差的值，故涉及笛卡尔坐标系与Frenet坐标的转换



首先，在笛卡尔坐标系中，参考点与真实点间在X、Y轴方向的误差：

$$dx = x - x_{des}, dy = y - y_{des}$$

由上述二维坐标转换的公式可知：

$$lateral\_error = dy * \cos\theta_{des} - dx * \sin\theta_{des}$$

2,横向位置偏差变化率，航向角偏差，航向角偏差变化率的求解方法解析

航向角偏差 = 车辆实际航向角 - 目标点航向角；
横向位置偏差变化率 = 线速度 × sin(航向角偏差);
航向角偏差变化率 = 车辆实际角速度 - 目标点角速度。

```
double heading_error =
    common::angles::normalize_angle(theta - model_state.reference_yaw);
double lateral_error_dot = linear_v * std::sin(heading_error);
double target_heading_rate =
    target_waypoint.curvature() * target_waypoint.velocity();
double heading_error_rate =  angular_v - target_heading_rate;

```

前馈模块

如前所述，在状态反馈下用于闭环侧向控制系统的状态空间模型可表示为：

$$\dot{x} = (A - B_1 K)x + B_2 \dot{\psi}_{des}$$

由于 $B_2\dot{\psi}_{des}$ 项的存在，当车辆在弯道行驶时，即使矩阵（$A - B_1 K$）趋于稳定，跟踪误差将不会完全收敛到 0。

本节我们将研究除状态反馈以外，使用前馈是否可确保弯道行驶的零稳态误差。假设转向盘控制器可由状态反馈加上前馈得到，以试图补偿道路曲率：

$$\delta = -Kx + \delta_{ff} \tag{3-4}$$

如果选择前馈转向角为以下值，稳态侧向位置误差可以为 0。

$$\delta_{ff} = \frac{m v_x^2}{RL}\left[\frac{l_r}{2C_{\alpha f}} - \frac{l_f}{2C_{\alpha r}} + \frac{l_f}{2C_{\alpha r}}k_3\right] + \frac{L}{R} - \frac{l_r}{R}k_3 \tag{3-11}$$

可近似看作：

$$\delta_{ff} = \frac{L}{R} + K_V a_y - k_3\left[\frac{l_r}{R} - \frac{l_f}{2C_{\alpha r}} \times \frac{m v_x^2}{Rl}\right] \tag{3-12}$$

式中，$K_V = \dfrac{l_r m}{2C_{\alpha f}(l_f + l_r)} - \dfrac{l_f m}{2C_{\alpha r}(l_f + l_r)}$为不足转向斜率，并且 $a_y = \dfrac{v_x^{\ 2}}{R}$。

3.3.1 QR矩阵选取

矩阵Q,R的选取，一般来说，Q值选得大意味着，要使得J小，那$x(t)$需要更小，也就是意味着闭环系统的矩阵(A-BK)的特征值处于S平面左边更远的地方，这样状态$x(t)$就以更快的速度衰减到0。另一方面，大的R表示更加关注输入变量$u(t)$,$u(t)$的减小，意味着状态衰减将变慢。

demo代码

离线生产lqr k 矩阵的代码

```
void GetOfflineLqrKMatrix() {
  NM_WARN("=====GetOfflineLqrKMatrix");
  double v = 10;  // v>5
  // double v_step = 1;  // v_step = 1kph
  const int states = 4;
  const int controls = 1;
  const int max_iter = 10000;
  const double eps = 0.001;
  unsigned int iter = 0;
  double err = 0.0;
  nullmax::common::Interpolation lat_err_interpolation;
  nullmax::common::Interpolation heading_err_interpolation;
  nullmax::common::Interpolation::InterpolatData xy1, xy2;
  nullmax::control::KineticalModel kinetical_model;
  kinetical_model.InitStateParams();
  double lat_err_gain_15 = 0.8;
  double lat_err_gain_30 = 0.6;
  double lat_err_gain_60 = 0.2;
  double lat_err_gain_80 = 0.15;
  double lat_err_gain_100 = 0.1;
  double lat_err_gain_120 = 0.1;
  double lat_err_gain_135 = 0.1;
  xy1.emplace_back(std::make_pair(15., const_cast<double *>(&lat_err_gain_15)));
  xy1.emplace_back(std::make_pair(30., const_cast<double *>(&lat_err_gain_30)));
  xy1.emplace_back(std::make_pair(60., const_cast<double *>(&lat_err_gain_60)));
  xy1.emplace_back(std::make_pair(80., const_cast<double *>(&lat_err_gain_80)));
  xy1.emplace_back(
      std::make_pair(100., const_cast<double *>(&lat_err_gain_100)));
  xy1.emplace_back(
      std::make_pair(120., const_cast<double *>(&lat_err_gain_120)));
  xy1.emplace_back(
      std::make_pair(135., const_cast<double *>(&lat_err_gain_135)));
  lat_err_interpolation.Init(xy1);

  double heading_err_gain_4 = 0.8;
  double heading_err_gain_8 = 0.6;
  double heading_err_gain_12 = 0.4;
  double heading_err_gain_20 = 0.3;
  double heading_err_gain_25 = 0.2;
  xy2.emplace_back(
      std::make_pair(4., const_cast<double *>(&heading_err_gain_4)));
```

```
  xy2.emplace_back(
      std::make_pair(8., const_cast<double *>(&heading_err_gain_8)));
  xy2.emplace_back(
      std::make_pair(12., const_cast<double *>(&heading_err_gain_12)));
  xy2.emplace_back(
      std::make_pair(20., const_cast<double *>(&heading_err_gain_20)));
  xy2.emplace_back(
      std::make_pair(25., const_cast<double *>(&heading_err_gain_25)));
  heading_err_interpolation.Init(xy2);
  // double q0 = 0.12;
  // NM_WARN("q0:{}", q0);
  Eigen::MatrixXd Q = Eigen::MatrixXd::Zero(states, states);
  Eigen::MatrixXd R = Eigen::MatrixXd::Zero(controls, controls);
  Eigen::MatrixXd K = Eigen::MatrixXd::Zero(controls, states);
  Eigen::MatrixXd K_0 = Eigen::MatrixXd::Zero(1, 135);
  Eigen::MatrixXd K_1 = Eigen::MatrixXd::Zero(1, 135);
  Eigen::MatrixXd K_2 = Eigen::MatrixXd::Zero(1, 135);
  Eigen::MatrixXd K_3 = Eigen::MatrixXd::Zero(1, 135);
  for (size_t i = 0; i < 135; i++) {
    v = i / 3.6;
    Q(0, 0) = 0.5;
    Q(1, 1) = 0.0;
    // Q(2, 2) = 1.0 * heading_err_interpolation.Interpolate(v);
    Q(2, 2) = 1.0;
    Q(3, 3) = 0.0;
    R(0, 0) = 200.0;
    NM_WARN(
        "SolveLQRProblem Q, R:, v is {}, Q0:{}, Q1{}, "
        "Q2{}, Q3{},R{}",
        i, Q(0, 0), Q(1, 1), Q(2, 2), Q(3, 3), R(0, 0));
    // R(0, 0) = 250.0;
    kinetical_model.model_state_.current_velocity = v;
    kinetical_model.UpdateStateMatrix();
    auto start_time_lqr = std::chrono::system_clock::now();
    auto res = nullmax::common::lqr_solver::SolveLQRProblem(
        kinetical_model.GD_, kinetical_model.HD_, Q, R, eps, max_iter, K, iter,
        err);
    auto end_time_lqr = std::chrono::system_clock::now();
    std::chrono::duration<double> diff_lqr = end_time_lqr - start_time_lqr;
    if (res) {
      NM_WARN(
          "SolveLQRProblem success, v:{}km/h, LQR used time:{}ms, k:{}, {}, "
          "{}, {}",
          i, diff_lqr.count() * 1000, K(0), K(1), K(2), K(3));
      // NM_WARN_STREAM("K is :" << K);
      K_0(i) = K(0);
      K_1(i) = K(1);
      K_2(i) = K(2);
      K_3(i) = K(3);
    } else {
      NM_WARN("SolveLQRProblem failed, v:{}km/h, iteration times:{}, error:{}",
              i, iter, err);
    }
  }
  NM_WARN("Matrix K0 is \n");
  for (int k = 0; k < 135; ++k) {
```

```
    std::cout << " " << K_0(k) << ","
              << " ";
  }
  std::cout << " " << std::endl;
  NM_WARN("Matrix K1 is \n");
  for (int k = 0; k < 135; ++k) {
    std::cout << " " << K_1(k) << ","
              << " ";
  }
  std::cout << " " << std::endl;
  NM_WARN("Matrix K2 is \n");
  for (int k = 0; k < 135; ++k) {
    std::cout << " " << K_2(k) << ","
              << " ";
  }
  std::cout << " " << std::endl;
  NM_WARN("Matrix K3 is \n");
  for (int k = 0; k < 135; ++k) {
    std::cout << " " << K_3(k) << ","
              << " ";
  }
  std::cout << " " << std::endl;
}


bool SolveLQRProblem(const Eigen::MatrixXd &GD, const Eigen::MatrixXd &HD,
                     const Eigen::MatrixXd &Q, const Eigen::MatrixXd &R,
                     const double tolerance,
                     const unsigned int max_num_iteration,
                     Eigen::MatrixXd &ptr_K, unsigned int &out_iteration_num,
                     double &out_err) {
  Eigen::MatrixXd P = Q;  // init P=S=Q P(k)
  std::ostringstream ss;
  ss << "G Matrix\n" << GD << std::endl;
  ss << "H Matrix\n" << HD << std::endl;
  ss << "Q Matrix\n" << Q << std::endl;
  ss << "R Matrix\n" << R << std::endl;
  NM_DEBUG(ss.str());
  Eigen::MatrixXd GD_T = GD.transpose();
  Eigen::MatrixXd HD_T = HD.transpose();
  unsigned int iteration_num = 0;
  double err = std::numeric_limits<double>::max();
  clock_t t;
  t = clock();
  NM_DEBUG("lqr--before solve");
  while (err > tolerance && iteration_num < max_num_iteration) {
    iteration_num++;
    Eigen::MatrixXd P_MO =
        Q +
        GD_T * P * (GD - HD * (R + HD_T * P * HD).inverse() * HD_T * P * GD);
    err = fabs((P_MO - P).maxCoeff());
    P = P_MO;
  }
  NM_DEBUG("lqr--end solve {}", iteration_num);
  out_iteration_num = iteration_num;
  out_err = err;
```

```
  if (iteration_num >= max_num_iteration) {
    NM_ERROR("Over iterate ERR= {}", err);
    t = clock() - t;
    NM_DEBUG_STREAM("It took me " << (static_cast<double>(t) / CLOCKS_PER_SEC)
                              << " seconds");
    return false;

  } else {
    ptr_K = (R + HD.transpose() * P * HD).inverse() * HD.transpose() * P * GD;
    return true;
  }
}
```

lqr控制算法代码

```
#include "lateral_control/lateral_follower/lqr_cherry_follower.h"

#include <sys/time.h>

#include "config/global_config/global_config.h"
#include "control_common/control_config/control_config.h"
#include "lqr_solver/lqr_solver.h"

namespace nullmax {
namespace control {
//原始的
// double K_0_0_array[116] = {
//     0.0141217,  0.0141177,  0.0141137,  0.0141097,  0.0141057,  0.0141017,
//     0.0140977,  0.0140937,  0.0140896,  0.0140857,  0.0140083,  0.0138812,
//     0.0137532,  0.0136241,  0.0134939,  0.0133627,  0.0132303,  0.0130968,
//     0.0129621,  0.0128261,  0.0126888,  0.0125502,  0.0124103,  0.0122689,
//     0.0120979,  0.0118099,  0.011515,   0.0112126,  0.0109022,  0.010583,
//     0.0102541,  0.00991465, 0.00956343, 0.00919909, 0.00881998, 0.00842407,
//     0.00800886, 0.00757113, 0.00710666, 0.00698472, 0.00695289, 0.00692094,
//     0.00688888, 0.0068567,  0.0068244,  0.00679197, 0.00675942, 0.00672674,
//     0.00669393, 0.006661,   0.00662792, 0.00659471, 0.00656137, 0.00652788,
//     0.00649425, 0.00646048, 0.00642656, 0.00639248, 0.00635826, 0.00632388,
//     0.00628935, 0.00625465, 0.00621978, 0.00618476, 0.00614956, 0.00611419,
//     0.00607864, 0.00604292, 0.00598601, 0.00592861, 0.00587068, 0.00581222,
//     0.00575321, 0.00569363, 0.00563346, 0.00557268, 0.00551127, 0.00544921,
//     0.00538648, 0.00532305, 0.0052589,  0.00519399, 0.0051283,  0.00506179,
//     0.00499444, 0.00492621, 0.00492543, 0.00492465, 0.00492387, 0.0049231,
//     0.00492233, 0.00492156, 0.00492079, 0.00492002, 0.00491924, 0.00491848,
//     0.00491772, 0.00491696, 0.00491619, 0.00491543, 0.00491468, 0.00491392,
//     0.00491316, 0.00491242, 0.00491166, 0.00491092, 0.00491017, 0.00490942,
//     0.00490868, 0.00490793, 0.00490719, 0.00490645, 0.00490573, 0.00490498,
//     0.00490425, 0.00490352};
// double K_0_1_array[116] = {
//     0.000115414, 0.000138458, 0.000161488, 0.000184505, 0.000207507,
//     0.000230496, 0.00025347,  0.000276429, 0.000299373, 0.000322303,
//     0.000343418, 0.000362979, 0.000382095, 0.00040076,  0.000418967,
```

```
//     0.00043671,  0.000453984, 0.00047078,  0.000487091, 0.000502911,
//     0.000518232, 0.000533044, 0.00054734,  0.000561111, 0.000573016,
//     0.000578631, 0.000582955, 0.000585927, 0.000587477, 0.000587526,
//     0.000585983, 0.000582744, 0.000577688, 0.000570675, 0.000561536,
//     0.000550066, 0.000536016, 0.000519072, 0.000498828, 0.00050164,
//     0.000510661, 0.000519567, 0.000528358, 0.000537031, 0.000545586,
//     0.000554023, 0.000562341, 0.000570538, 0.000578614, 0.000586569,
//     0.000594401, 0.000602109, 0.000609692, 0.00061715,  0.000624482,
//     0.000631686, 0.000638762, 0.000645709, 0.000652525, 0.00065921,
//     0.000665763, 0.000672182, 0.000678466, 0.000684615, 0.000690627,
//     0.000696501, 0.000702236, 0.00070783,  0.000710801, 0.000713525,
//     0.000715997, 0.000718212, 0.000720168, 0.000721858, 0.000723278,
//     0.000724423, 0.000725287, 0.000725864, 0.000726149, 0.000726135,
//     0.000725814, 0.000725182, 0.000724228, 0.000722946, 0.000721328,
//     0.000719363, 0.000727069, 0.000734764, 0.000742449, 0.000750125,
//     0.000757791, 0.000765446, 0.000773091, 0.000780726, 0.000788349,
//     0.000795962, 0.000803565, 0.000811157, 0.000818736, 0.000826306,
//     0.000833864, 0.00084141,  0.000848944, 0.000856468, 0.000863978,
//     0.000871479, 0.000878965, 0.00088644,  0.000893902, 0.000901352,
//     0.000908789, 0.000916214, 0.000923628, 0.000931025, 0.000938411,
//     0.000945784};
// double K_0_2_array[116] = {
//     0.27156,  0.271627, 0.271705, 0.271796, 0.271898, 0.272012, 0.272138,
//     0.272275, 0.272424, 0.272585, 0.272053, 0.27105,  0.270039, 0.269019,
//     0.26799,  0.26695,  0.265898, 0.264834, 0.263756, 0.262663, 0.261554,
//     0.260428, 0.259283, 0.258118, 0.256659, 0.254038, 0.251303, 0.248443,
//     0.245445, 0.242293, 0.23897,  0.235456, 0.231727, 0.227754, 0.2235,
//     0.218921, 0.213962, 0.208548, 0.202582, 0.201149, 0.200946, 0.200743,
//     0.20054,  0.200336, 0.200131, 0.199926, 0.199719, 0.199511, 0.199302,
//     0.199092, 0.198879, 0.198665, 0.198448, 0.19823,  0.198009, 0.197785,
//     0.197558, 0.197329, 0.197096, 0.19686,  0.19662,  0.196377, 0.19613,
//     0.195878, 0.195622, 0.195361, 0.195096, 0.194826, 0.194189, 0.193537,
//     0.192868, 0.192183, 0.19148,  0.190759, 0.19002,  0.18926,  0.188479,
//     0.187678, 0.186853, 0.186005, 0.185133, 0.184235, 0.183312, 0.182359,
//     0.181377, 0.180366, 0.180675, 0.180986, 0.181299, 0.181614, 0.181929,
//     0.182248, 0.182567, 0.182889, 0.183213, 0.183537, 0.183863, 0.18419,
//     0.18452,  0.18485,  0.185181, 0.185515, 0.185849, 0.186184, 0.186522,
//     0.186859, 0.187198, 0.187539, 0.18788,  0.188223, 0.188566, 0.188911,
//     0.189255, 0.189602, 0.189949, 0.190297};
// double K_0_3_array[116] = {
//     0.0022181,  0.00266169, 0.00310525, 0.00354878, 0.00399226, 0.00443568,
//     0.00487905, 0.00532235, 0.00576557, 0.00620869, 0.00663468, 0.00704585,
//     0.00745266, 0.00785501, 0.00825281, 0.00864594, 0.00903429, 0.00941776,
//     0.00979623, 0.0101696,  0.0105377,  0.0109004,  0.0112575,  0.011609,
//     0.0119422,  0.0122151,  0.0124733,  0.0127159,  0.0129416,  0.0131492,
//     0.0133371,  0.0135034,  0.013646,   0.0137623,  0.0138492,  0.0139027,
//     0.0139181,  0.0138889,  0.0138069,  0.014012,   0.0142965,  0.0145792,
//     0.0148601,  0.0151391,  0.0154164,  0.0156918,  0.0159652,  0.0162368,
//     0.0165064,  0.0167741,  0.0170397,  0.0173034,  0.0175651,  0.0178247,
//     0.0180822,  0.0183376,  0.0185909,  0.018842,   0.019091,   0.0193377,
//     0.0195822,  0.0198244,  0.0200644,  0.020302,   0.0205372,  0.0207701,
//     0.0210006,  0.0212287,  0.0214173,  0.0216017,  0.0217818,  0.0219572,
//     0.0221281,  0.0222942,  0.0224556,  0.0226118,  0.0227631,  0.022909,
//     0.0230495,  0.0231845,  0.0233137,  0.0234371,  0.0235544,  0.0236653,
//     0.0237697,  0.0238674,  0.0241211,  0.0243745,  0.0246273,  0.0248796,
//     0.0251314,  0.0253827,  0.0256335,  0.0258838,  0.0261337,  0.0263829,
```

```
//     0.0266316,  0.0268798,  0.0271275,  0.0273745,  0.0276209,  0.0278669,
//     0.0281123,  0.0283569,  0.0286012,  0.0288446,  0.0290876,  0.02933,
//     0.0295717,  0.0298129,  0.0300533,  0.0302932,  0.0305322,  0.0307709,
//     0.0310088,  0.031246};
// 1.0 , 0 , 1.0, 0.0  200
double K_0_0_array[135] = {
    0,         0,         0.0499449, 0.0499185, 0.049892,  0.0498656, 0.0498393,
    0.0498129, 0.0497866, 0.0497602, 0.0497339, 0.0497077, 0.0496814, 0.0496552,
    0.0496291, 0.0496029, 0.0495768, 0.0495507, 0.0495247, 0.0494986, 0.0494727,
    0.0494467, 0.0494209, 0.049395,  0.0493692, 0.0493434, 0.0493177, 0.0492921,
    0.0492665, 0.0492409, 0.0492154, 0.04919,   0.0491646, 0.0491393, 0.0491141,
    0.0490889, 0.0490638, 0.0490387, 0.0490138, 0.0489889, 0.048964,  0.0489393,
    0.0489146, 0.04889,   0.0488656, 0.0488412, 0.0488168, 0.0487926, 0.0487684,
    0.0487444, 0.0487205, 0.0486965, 0.0486728, 0.0486492, 0.0486256, 0.0486021,
    0.0485788, 0.0485555, 0.0485324, 0.0485094, 0.0484864, 0.0484636, 0.0484408,
    0.0484182, 0.0483957, 0.0483734, 0.0483511, 0.0483289, 0.0483068, 0.0482849,
    0.048263,  0.0482413, 0.0482197, 0.0481982, 0.0481767, 0.0481555, 0.0481342,
    0.0481132, 0.0480921, 0.0480713, 0.0480504, 0.0480427, 0.0480228, 0.0480031,
    0.0479835, 0.0479641, 0.0479448, 0.0479256, 0.0479065, 0.0478876, 0.0478689,
    0.0478502, 0.0478318, 0.0478134, 0.0477952, 0.0477771, 0.0477591, 0.0477413,
    0.0477237, 0.0477061, 0.0476887, 0.0476714, 0.0476543, 0.0476372, 0.0476204,
    0.0476036, 0.047587,  0.0475705, 0.0475541, 0.0475379, 0.0475218, 0.0475057,
    0.0474899, 0.0474742, 0.0474585, 0.047443,  0.0474276, 0.0474123, 0.0473972,
    0.0473821, 0.0473673, 0.0473524, 0.0473378, 0.0473232, 0.0473088, 0.0472944,
    0.0472802, 0.047266,  0.047252,  0.0472381, 0.0472243, 0.0472107, 0.047197,
    0.0471835, 0.0471702};

double K_0_1_array[135] = {
    0,           0,           0.000173969, 0.000260825, 0.000347592,
    0.000434267, 0.000520846, 0.000607327, 0.000693706, 0.00077998,
    0.000866147, 0.000952203, 0.00103815,  0.00112397,  0.00120968,
    0.00129526,  0.00138071,  0.00146604,  0.00155123,  0.00163629,
    0.0017212,   0.00180598,  0.0018906,   0.00197508,  0.0020594,
    0.00214357,  0.00222757,  0.00231142,  0.00239509,  0.00247859,
    0.00256192,  0.00264506,  0.00272803,  0.00281081,  0.00289339,
    0.00297579,  0.00305798,  0.00313997,  0.00322176,  0.00330333,
    0.00338469,  0.00346584,  0.00354676,  0.00362745,  0.00370793,
    0.00378816,  0.00386815,  0.00394791,  0.00402742,  0.00410668,
    0.00418569,  0.00426444,  0.00434294,  0.00442117,  0.00449914,
    0.00457683,  0.00465426,  0.0047314,   0.00480826,  0.00488485,
    0.00496113,  0.00503714,  0.00511285,  0.00518827,  0.00526338,
    0.00533821,  0.00541272,  0.00548692,  0.00556081,  0.0056344,
    0.00570767,  0.00578062,  0.00585326,  0.00592558,  0.00599756,
    0.00606923,  0.00614055,  0.00621157,  0.00628224,  0.0063526,
    0.0064226,   0.00649412,  0.00656359,  0.00663272,  0.00670153,
    0.00677,     0.00683814,  0.00690594,  0.00697342,  0.00704055,
    0.00710736,  0.00717382,  0.00723996,  0.00730575,  0.00737121,
    0.00743634,  0.00750113,  0.00756558,  0.0076297,   0.00769348,
    0.00775693,  0.00782005,  0.00788283,  0.00794527,  0.00800738,
    0.00806916,  0.00813061,  0.00819173,  0.00825251,  0.00831297,
    0.0083731,   0.00843289,  0.00849237,  0.00855152,  0.00861033,
    0.00866883,  0.00872699,  0.00878485,  0.00884238,  0.00889958,
    0.00895648,  0.00901304,  0.00906931,  0.00912524,  0.00918088,
    0.00923618,  0.0092912,   0.00934588,  0.00940025,  0.00945435,
    0.00950811,  0.0095616,   0.00961475,  0.0096676,   0.00972019};
double K_0_2_array[135] = {
```

```
    0,        0,        0.472074, 0.472207, 0.472381, 0.472596, 0.472853,
    0.473152, 0.473491, 0.47387,  0.474291, 0.474751, 0.475251, 0.475791,
    0.47637,  0.476987, 0.477644, 0.478338, 0.47907,  0.479839, 0.480644,
    0.481486, 0.482364, 0.483277, 0.484224, 0.485206, 0.486222, 0.48727,
    0.488351, 0.489464, 0.490608, 0.491783, 0.492988, 0.494222, 0.495485,
    0.496776, 0.498094, 0.49944,  0.500811, 0.502208, 0.50363,  0.505075,
    0.506544, 0.508036, 0.50955,  0.511085, 0.512641, 0.514217, 0.515812,
    0.517425, 0.519056, 0.520705, 0.52237,  0.524051, 0.525748, 0.527458,
    0.529183, 0.530921, 0.532671, 0.534433, 0.536207, 0.537992, 0.539786,
    0.54159,  0.543403, 0.545224, 0.547054, 0.54889,  0.550733, 0.552582,
    0.554436, 0.556296, 0.55816,  0.560028, 0.5619,   0.563775, 0.565653,
    0.567532, 0.569413, 0.571296, 0.573179, 0.575168, 0.577058, 0.57895,
    0.580841, 0.58273,  0.58462,  0.586508, 0.588394, 0.590278, 0.59216,
    0.59404,  0.595916, 0.59779,  0.599661, 0.601527, 0.603391, 0.60525,
    0.607104, 0.608955, 0.6108,   0.612641, 0.614477, 0.616308, 0.618133,
    0.619952, 0.621766, 0.623574, 0.625376, 0.627171, 0.628961, 0.630744,
    0.63252,  0.63429,  0.636053, 0.637809, 0.639559, 0.641301, 0.643036,
    0.644764, 0.646484, 0.648198, 0.649903, 0.651602, 0.653292, 0.654976,
    0.656651, 0.658319, 0.659979, 0.661632, 0.663276, 0.664913, 0.666542,
    0.668163, 0.669776};
double K_0_3_array[135] = {
    0,          0,          0.00164412, 0.00246625, 0.00328838, 0.00411048,
    0.00493249, 0.00575439, 0.00657612, 0.00739766, 0.00821895, 0.00903996,
    0.00986063, 0.0106809,  0.0115008,  0.0123202,  0.0131391,  0.0139574,
    0.0147751,  0.015592,   0.0164083,  0.0172238,  0.0180384,  0.0188521,
    0.0196648,  0.0204765,  0.0212871,  0.0220965,  0.0229046,  0.0237115,
    0.024517,   0.025321,   0.0261236,  0.0269245,  0.0277237,  0.0285212,
    0.0293169,  0.0301107,  0.0309025,  0.0316923,  0.0324799,  0.0332653,
    0.0340484,  0.0348292,  0.0356075,  0.0363833,  0.0371565,  0.037927,
    0.0386948,  0.0394597,  0.0402217,  0.0409808,  0.0417368,  0.0424896,
    0.0432394,  0.0439858,  0.0447289,  0.0454686,  0.0462049,  0.0469376,
    0.0476668,  0.0483924,  0.0491142,  0.0498323,  0.0505466,  0.051257,
    0.0519636,  0.0526662,  0.0533648,  0.0540594,  0.0547499,  0.0554362,
    0.0561184,  0.0567964,  0.0574703,  0.0581398,  0.0588052,  0.0594661,
    0.0601228,  0.060775,   0.061423,   0.0620753,  0.0627151,  0.0633505,
    0.0639815,  0.064608,   0.0652302,  0.0658479,  0.0664613,  0.0670701,
    0.0676746,  0.0682747,  0.0688703,  0.0694616,  0.0700484,  0.0706308,
    0.0712088,  0.0717825,  0.0723517,  0.0729166,  0.0734771,  0.0740333,
    0.0745851,  0.0751328,  0.075676,   0.076215,   0.0767497,  0.0772802,
    0.0778066,  0.0783287,  0.0788465,  0.0793604,  0.07987,    0.0803755,
    0.0808771,  0.0813744,  0.0818679,  0.0823572,  0.0828426,  0.0833241,
    0.0838016,  0.0842753,  0.084745,   0.085211,   0.085673,   0.0861314,
    0.0865859,  0.0870368,  0.087484,   0.0879274,  0.0883673,  0.0888034,
    0.0892361,  0.0896653,  0.0900907};
double feedforward_gains[5] = {1.0, 1.0, 1.0, 1.0, 1.0};
LqrCherryFollower::LqrCherryFollower() {}

void LqrCherryFollower::InitPublishedTopics() {}

void LqrCherryFollower::InitStateParams() {
  kinetical_model_->InitStateParams();
  reference_curvature_filter.InitParameters(0.05, 0.95);
  InitInterpolation();
}

void LqrCherryFollower::Reset() {
```

```
  reference_curvature_filter.Reset(target_point.curvature());
}

bool LqrCherryFollower::RunOnce(TwistCmdStamped &command_twist) {
  command_twist.mutable_header()->mutable_stamp()->CopyFrom(
      control_context_->frame_time);
  command_twist.mutable_twist()->mutable_linear()->set_x(
      control_context_->longitudinal_cmd.velocity());
  auto lqr_state = lateral_monitor_msg_->mutable_model_state();

  const auto &ego_car_state = control_context_->ego_car_state;
  auto &model_state = kinetical_model_->model_state_;
  // 根据自车的odom坐标，预测自车几何中心点坐标
  const double longitudinal_bias =
      common::config::GetEgoCarStaticConfig().baselink_to_car_front;
  Point predict_geometric_center_point;
  CenterPointPredictor(control_context_->odom_pose_point,
                       control_context_->ego_car_state.linear_velocity(),
                       control_context_->ego_car_state.angular_velocity(),
                       longitudinal_bias, predict_geometric_center_point);
  // 根据接收到的轨迹点，选择近处的控制点
  CalculateLateralControlPointNear(control_context_->lateral_cmd.trajectory(),
                                   predict_geometric_center_point,
                                   target_point);
  lateral_monitor_msg_->mutable_follower_target()->set_x(target_point.x());
  lateral_monitor_msg_->mutable_follower_target()->set_y(target_point.y());
  lateral_monitor_msg_->mutable_follower_target()->set_z(target_point.r());
  lateral_monitor_msg_->set_kappa(target_point.curvature());
  target_point.set_curvature(target_point.curvature() * 0.35);

  auto target_point_baselink =
      common::coordinate::ConvertToRelativeCoordinatePoint(
          target_point, control_context_->odom_pose_point);
  lateral_monitor_msg_->mutable_follower_target_basklink()->set_x(
      target_point_baselink.x());
  lateral_monitor_msg_->mutable_follower_target_basklink()->set_y(
      target_point_baselink.y());
  lateral_monitor_msg_->mutable_follower_target_basklink()->set_z(
      target_point_baselink.z());

  // 对计算出来的控制点做曲率的滤波
  model_state.ref_curvature =
      reference_curvature_filter.ProcessMeasurement(target_point.curvature());
  // target_point.set_r(target_point.r() * 0.9);
  model_state.reference_yaw = target_point.r();
  lqr_state->set_ref_curvature(model_state.ref_curvature);
  // 计算误差项
  ComputeLateralErrors(
      control_context_->odom_pose_point.x(),
      control_context_->odom_pose_point.y(),
      control_context_->odom_pose_point.z(), ego_car_state.linear_velocity(),
      ego_car_state.angular_velocity(), ego_car_state.linear_acceleration(),
      target_point, model_state);
  // 计算反馈值
  UpdateErrorCoefficient(model_state);
  // 计算前馈值
```

```
  ComputeFeedForward(model_state);
  //对lqr角度做滤波后下发到控制器
  command_twist.mutable_twist()->mutable_angular()->set_z(
      command_twist.twist().linear().x() * tan(lqr_steer_angle_) /
      kinetical_model_->wheel_base_);
  command_twist.mutable_twist()->set_front_wheel_angle_directly(true);
  command_twist.mutable_twist()->set_front_wheel_angle(lqr_steer_angle_);
  return true;
}
void LqrCherryFollower::CenterPointPredictor(const Point &rear_axle_pose,
                                             const double linear_v,
                                             const double angular_v,
                                             const double longitudinal_bias,
                                             Point &predict_center_point) {
  const double target_delay = GetLateralControllerConfig().target_delay;
  const double angular_delay = angular_v * target_delay;
  const double distance_delay = linear_v * target_delay;
  const double angular_middle = rear_axle_pose.z() + 0.5 * angular_delay;
  predict_center_point.set_z(
      common::angles::normalize_angle(rear_axle_pose.z() + angular_delay));
  predict_center_point.set_x(rear_axle_pose.x() +
                             longitudinal_bias *
                                 std::cos(predict_center_point.z()) +
                             distance_delay * std::cos(angular_middle));
  predict_center_point.set_y(rear_axle_pose.y() +
                             longitudinal_bias *
                                 std::sin(predict_center_point.z()) +
                             distance_delay * std::sin(angular_middle));
}
void LqrCherryFollower::CalculateLateralControlPointNear(
    const Trajectory &trajectory, const Point &predict_center_point,
    WaypointCell &lateral_control_wpc) {
  if (trajectory.waypoints_size() < 2) {
    lateral_control_wpc.set_x(predict_center_point.x());
    lateral_control_wpc.set_y(predict_center_point.y());
    lateral_control_wpc.set_r(predict_center_point.z());
    lateral_control_wpc.set_curvature(0.);
  } else {
    double d_front =
        std::pow(predict_center_point.x() - trajectory.waypoints(0).x(), 2) +
        std::pow(predict_center_point.y() - trajectory.waypoints(0).y(), 2);
    double d_rear = d_front;
    int waypoint_id = 0;
    while (waypoint_id < trajectory.waypoints_size() - 1) {
      auto &wpc = trajectory.waypoints(waypoint_id + 1);
      d_rear = std::pow(predict_center_point.x() - wpc.x(), 2) +
               std::pow(predict_center_point.y() - wpc.y(), 2);
      if (d_front > d_rear) {
        d_front = d_rear;
        waypoint_id++;
      } else {
        break;
      }
    }
    lateral_control_wpc.set_x(trajectory.waypoints(waypoint_id).x());
    lateral_control_wpc.set_y(trajectory.waypoints(waypoint_id).y());
```

```
    lateral_control_wpc.set_r(trajectory.waypoints(waypoint_id).r());
    lateral_control_wpc.set_curvature(
        trajectory.waypoints(waypoint_id).curvature());
  }
}

void LqrCherryFollower::ComputeLateralErrors(
    const double x_rear_axle, const double y_rear_axle, const double theta,
    const double linear_v, const double angular_v, const double linear_a,
    const WaypointCell &target_waypoint, ModelState &model_state) {
  NM_DEBUG("---ComputeLateralErrors---");
  NM_DEBUG(
      "curr_x_rear_axle:{}, curr_y_rear_axle:{}, curr_theta:{}, curr_v:{}, "
      "curr_w:{}, curr_a:{}",
      x_rear_axle, y_rear_axle, theta, linear_v, angular_v, linear_a);
  auto lqr_state = lateral_monitor_msg_->mutable_model_state();
  // 首先求解自车的预测三个中心点坐标
  auto wheel_base = kinetical_model_->wheel_base_;
  const float time_delay = GetLateralControllerConfig().target_delay;
  const float delta_d = linear_v * time_delay;
  const float delta_r = angular_v * time_delay;
  Point predict_rear_axle_center_point;
  Point predict_geometric_center_point;
  Point predict_front_axle_center_point;

  predict_rear_axle_center_point.set_x(x_rear_axle +
                                       delta_d * cosf(theta + 0.5f * delta_r));
  predict_rear_axle_center_point.set_y(y_rear_axle +
                                       delta_d * sinf(theta + 0.5f * delta_r));
  predict_rear_axle_center_point.set_z(
      common::angles::normalize_angle(theta + delta_r));

  predict_geometric_center_point.set_z(predict_rear_axle_center_point.z());
  predict_geometric_center_point.set_x(
      predict_rear_axle_center_point.x() +
      0.5f * wheel_base * cosf(predict_geometric_center_point.z()));
  predict_geometric_center_point.set_y(
      predict_rear_axle_center_point.y() +
      0.5f * wheel_base * sinf(predict_geometric_center_point.z()));

  predict_front_axle_center_point.set_z(predict_rear_axle_center_point.z());
  predict_front_axle_center_point.set_x(
      predict_rear_axle_center_point.x() +
      wheel_base * cosf(predict_front_axle_center_point.z()));
  predict_front_axle_center_point.set_y(
      predict_rear_axle_center_point.y() +
      wheel_base * sinf(predict_front_axle_center_point.z()));

  NM_DEBUG(
      "target_x:{}, target_y:{}, target_r:{}, "
      "target_v:{}, target_k:{}",
      target_waypoint.x(), target_waypoint.y(), target_waypoint.r(), linear_v,
      target_waypoint.curvature());

  const double dx = predict_rear_axle_center_point.x() - target_waypoint.x();
  const double dy = predict_rear_axle_center_point.y() - target_waypoint.y();
```

```
  const double cos_target_heading = std::cos(target_waypoint.r());
  const double sin_target_heading = std::sin(target_waypoint.r());

  lateral_error_ = cos_target_heading * dy - sin_target_heading * dx;
  model_state.lateral_error = lateral_error_;
  lqr_state->set_lateral_error(lateral_error_);
  heading_error_ = common::angles::normalize_angle(
      predict_rear_axle_center_point.z() - model_state.reference_yaw);
  model_state.heading_error = heading_error_;
  lqr_state->set_heading_error(heading_error_);

  lateral_error_rate_ = linear_v * std::sin(heading_error_);
  model_state.lateral_error_rate = lateral_error_rate_;
  lqr_state->set_lateral_error_rate(lateral_error_rate_);

  model_state.heading_rate = angular_v;
  model_state.ref_heading_rate = model_state.ref_curvature * linear_v;
  heading_error_rate_ = model_state.heading_rate - model_state.ref_heading_rate;
  model_state.heading_error_rate = heading_error_rate_;
  lqr_state->set_heading_error_rate(heading_error_rate_);
}
void LqrCherryFollower::ComputeFeedForward(ModelState &model_state) {
  auto lqr_state = lateral_monitor_msg_->mutable_model_state();
  float lr = kinetical_model_->lr_;
  float lf = kinetical_model_->lf_;
  float cr = kinetical_model_->cr_;
  float cf = kinetical_model_->cf_;
  float mass = kinetical_model_->vehicle_mass_;
  float wheelbase = kinetical_model_->wheel_base_;
  kv_ = lr * mass / 2 / cr / wheelbase - lf * mass / 2 / cf / wheelbase;
  lateral_monitor_msg_->set_stanley_delta_theta_phi(kv_);
  const float v = control_context_->ego_car_state.linear_velocity();
  steer_angle_feedforwardterm_ = wheelbase * model_state.ref_curvature +
                                 kv_ * v * v * model_state.ref_curvature;
  lateral_monitor_msg_->set_stanley_delta_theta_sum(
      steer_angle_feedforwardterm_);
  // v 越大 , curvature为正, steer越大, curvature为负, steer越小(大)
  steady_steer_angle_error_ =
      k2_ *
      (lr * model_state.ref_curvature -
       lf * mass * v * v * model_state.ref_curvature / 2 / cr / wheelbase);
  lateral_monitor_msg_->set_stanley_delta_theta_y(steady_steer_angle_error_);
  steer_angle_feedforward_ =
      steer_angle_feedforwardterm_ - steady_steer_angle_error_;
  steer_angle_feedforward_ = steer_angle_feedforward_;

  model_state.feed_forward_steering_angle = steer_angle_feedforward_;
  model_state.steady_state_steering_angle = steady_steer_angle_error_;
  lqr_state->set_feed_forward_steering_angle(
      steer_angle_feedforward_ *
      common::config::GetEgoCarConfig().steering_ratio);
}
void LqrCherryFollower::UpdateErrorCoefficient(ModelState &model_state) {
  auto lqr_state = lateral_monitor_msg_->mutable_model_state();
  k0_ = k1_interpolation_.Interpolate(
```

```
      control_context_->ego_car_state.linear_velocity());
  k1_ = k2_interpolation_.Interpolate(
      control_context_->ego_car_state.linear_velocity());
  k2_ = k3_interpolation_.Interpolate(
      control_context_->ego_car_state.linear_velocity());
  k3_ = k4_interpolation_.Interpolate(
      control_context_->ego_car_state.linear_velocity());

  lateral_monitor_msg_->set_current_velocity(
      control_context_->ego_car_state.linear_velocity());
  lateral_monitor_msg_->set_stanley_frontwheel_angle_cmd(k2_);

  model_state.K_0_0 = k0_;
  model_state.K_0_1 = k1_;
  model_state.K_0_2 = k2_;
  model_state.K_0_3 = k3_;

  lqr_state->set_k_0_0(k0_);
  lqr_state->set_k_0_1(k1_);
  lqr_state->set_k_0_2(k2_);
  lqr_state->set_k_0_3(k3_);
  float feedback_0_0 = k0_ * lateral_error_;
  float feedback_0_1 = k1_ * lateral_error_rate_;
  float feedback_0_2 = k2_ * heading_error_;
  float feedback_0_3 = k3_ * heading_error_rate_;

  lqr_state->set_input_0_lateral_error_part(
      feedback_0_0 * common::config::GetEgoCarConfig().steering_ratio);
  lqr_state->set_input_0_laterla_error_rate_part(
      feedback_0_1 * common::config::GetEgoCarConfig().steering_ratio);
  lqr_state->set_input_0_heading_error_part(
      feedback_0_2 * common::config::GetEgoCarConfig().steering_ratio);
  lqr_state->set_input_0_heading_error_rate_part(
      feedback_0_3 * common::config::GetEgoCarConfig().steering_ratio);

  steer_angle_feedback_ =
      feedback_0_0 + feedback_0_1 + feedback_0_2 + feedback_0_3;
  lqr_state->set_input_0(steer_angle_feedback_ *
                         common::config::GetEgoCarConfig().steering_ratio);
  lqr_steer_angle_ = steer_angle_feedforward_ - steer_angle_feedback_;
  lqr_state->set_mpc_result(lqr_steer_angle_ *
                            common::config::GetEgoCarConfig().steering_ratio);
}
void LqrCherryFollower::GetLqrResult(NM_DOUBLE &steering_angle) {}

void LqrCherryFollower::InitInterpolation() {
  auto &config = GetLateralControllerConfig();
  common::Interpolation::InterpolatData xy1, xy2;
  xy1.emplace_back(std::make_pair(
      4, const_cast<double *>(&config.lat_err_gain_scheduler_4)));
  xy1.emplace_back(std::make_pair(
      8, const_cast<double *>(&config.lat_err_gain_scheduler_8)));
  xy1.emplace_back(std::make_pair(
      12, const_cast<double *>(&config.lat_err_gain_scheduler_12)));
  xy1.emplace_back(std::make_pair(
      20, const_cast<double *>(&config.lat_err_gain_scheduler_20)));
```

```
  xy1.emplace_back(std::make_pair(
      25, const_cast<double *>(&config.lat_err_gain_scheduler_25)));
  lat_err_interpolation_.Init(xy1);

  xy2.emplace_back(std::make_pair(
      4, const_cast<double *>(&config.heading_err_gain_scheduler_4)));
  xy2.emplace_back(std::make_pair(
      8, const_cast<double *>(&config.heading_err_gain_scheduler_8)));
  xy2.emplace_back(std::make_pair(
      12, const_cast<double *>(&config.heading_err_gain_scheduler_12)));
  xy2.emplace_back(std::make_pair(
      20, const_cast<double *>(&config.heading_err_gain_scheduler_20)));
  xy2.emplace_back(std::make_pair(
      25, const_cast<double *>(&config.heading_err_gain_scheduler_25)));
  heading_err_interpolation_.Init(xy2);
  // lqr offline Matrix_K table
  // q0 0.05 interpolation q2 1.0 interpolation r0 200

  common::Interpolation::InterpolatData xy3, xy4, xy5, xy6;
  for (int i = 1; i < 135; ++i) {
    xy3.emplace_back(
        std::make_pair(i / 3.6, const_cast<double *>(&K_0_0_array[i])));
    xy4.emplace_back(
        std::make_pair(i / 3.6, const_cast<double *>(&K_0_1_array[i])));
    xy5.emplace_back(
        std::make_pair(i / 3.6, const_cast<double *>(&K_0_2_array[i])));
    xy6.emplace_back(
        std::make_pair(i / 3.6, const_cast<double *>(&K_0_3_array[i])));
  }

  k1_interpolation_.Init(xy3);
  k2_interpolation_.Init(xy4);
  k3_interpolation_.Init(xy5);
  k4_interpolation_.Init(xy6);
  //前馈增益插值
  common::Interpolation::InterpolatData xy7;
  xy7.emplace_back(
      std::make_pair(0, const_cast<double *>(&feedforward_gains[0])));
  xy7.emplace_back(
      std::make_pair(30, const_cast<double *>(&feedforward_gains[1])));
  xy7.emplace_back(
      std::make_pair(60, const_cast<double *>(&feedforward_gains[2])));
  xy7.emplace_back(
      std::make_pair(90, const_cast<double *>(&feedforward_gains[3])));
  xy7.emplace_back(
      std::make_pair(120, const_cast<double *>(&feedforward_gains[4])));
  feedforward_gain_interpolation_.Init(xy7);
}
void LqrCherryFollower::UpdateNextWaypointId(NM_DOUBLE target_delay) {
  auto &trajectory = control_context_->lateral_cmd.trajectory();
  if (trajectory.waypoints_size() <= 0) {
    return;
  }
  if (control_context_->control_input.trick_vars().apa_driving_direction() ==
      ApaDriveDirection::BACKWARD) {
    control_context_->is_reverse = true;
```

```
  } else {
    control_context_->is_reverse = false;
  }
  Point wp_baselink;
  while (control_context_->next_waypoint_id < trajectory.waypoints_size() - 1) {
    auto &wpc = trajectory.waypoints(control_context_->next_waypoint_id);
    common::coordinate::ConvertToRelativeCoordinate(
        wpc, control_context_->odom_pose_point, wp_baselink);
    NM_DOUBLE delay_distance =
        target_delay * control_context_->ego_car_state.linear_velocity();

    if (!control_context_->is_reverse) {
      if (wp_baselink.x() >= delay_distance) {
        break;
      }
    } else {
      if (wp_baselink.x() <= delay_distance) {
        break;
      }
    }
    control_context_->next_waypoint_id++;
  }

  if (control_context_->next_waypoint_id >= trajectory.waypoints_size()) {
    NM_WARN("passing through all final waypoints");
  }
}
double InterpolateVal(const double x, double array_x[], double array_y[]) {
  const int size_x = 116;
  const int size_y = 116;
  if (x <= array_x[0]) {
    return array_y[0];
  }
  if (x >= array_x[size_x - 1]) {
    return array_y[size_y - 1];
  }
  int start = 0;
  int end = size_x - 1;
  while ((end - start) > 1) {
    int mid = (end - start) / 2 + start;
    if (array_x[mid] > x)
      end = mid;
    else
      start = mid;
  }
  float x1 = array_x[start];
  float x2 = array_x[start + 1];
  float y1 = array_y[start];
  float y2 = array_y[start + 1];
  if (x1 == x2) {
    return y1;
  }
  return y1 + (x - x1) * (y2 - y1) / (x2 - x1);
}

}  // namespace control
```

```
}  // namespace nullmax
```

MPC控制算法

参考文献

误差方程建立

车辆横向控制模型构建

PP
PP
PP

LQR求解方法

LQR离散系统求解方法

Lqr横向控制求解过程