清华大学—联合电子
合作项目

结题报告

2019 年 3 月

Content

1 背景：

随着高速公路的迅速发展以及车辆保有量的增加，传统的车辆被动安全和常规主动安全技术不能满足现代交通日益发展的要求。基于计算机信息处理技术、传感器技术和车辆控制技术的进步，以及高效环境感知的辅助驾驶技术或全自动驾驶技术迅速发展，以主动控制为核心的先进车辆安全技术必将是现代交通系统和未来高度智能化交通系统的关键要素之一。实现全自主无人驾驶车辆在复杂路面安全稳定的轨迹跟踪控制具有十分重要的意义。轨迹跟踪控制系统是无人驾驶车辆实现智能化和实用化的必要条件。如果轨迹跟踪控制系统能通过预测来满足动力学约束，通过主动前轮转向控制车辆，在保证车辆稳定性的前提下跟踪规划轨迹，则可有效地避免事故发生。因此，分析车辆动力学模型对解决上述问题具有关键的作用，以二自由度车辆动力学模型作为预测模型，在保证车辆稳定的前提下实现轨迹跟踪。

传统的控制方法如纯跟踪算法、线性二次型调节跟踪控制器、前馈-反馈控制等，大都是基于运动学控制，很少涉及到车辆动力学特性。而且传统的最优控制方法需要精确的控制模型，而车辆动力学模型是一个复杂的非线性系统，在工程使用中往往需要对模型简化，模型的精确性很难保证。而模型预测控制算法是一类滚动求解带约束优化问题的控制方法。该方法考虑了控制系统的动力学模型并预测未来一段时间内系统的输出行为，通过求解最优控制问题使得系统在未来一段时间内的跟踪误差最小。模型预测控制算法具有预测模型、滚动优化和反馈校正的基本特性，尤其适用于不易建立精确数学模型且存在约束条件的控制系统，对解决无人驾驶车辆的轨迹跟踪控制问题具有独特的优势。

图 1-1.城市路面工况展示

拟在城市道路场景，设计智能汽车的控制器，实现车辆纵横向控制的轨迹跟踪。如图 1-1 所示为城市路面，拟（1）在右转和左转两种转弯工况下，根据曲率情况选取多组代表性车速范围测试，（2）连续换道定速工况的轨迹跟踪控制，（3）直路前车缓变速工况下的恒定车距策略的前车跟随和自车车道保持功能验证。

2 物理模型：

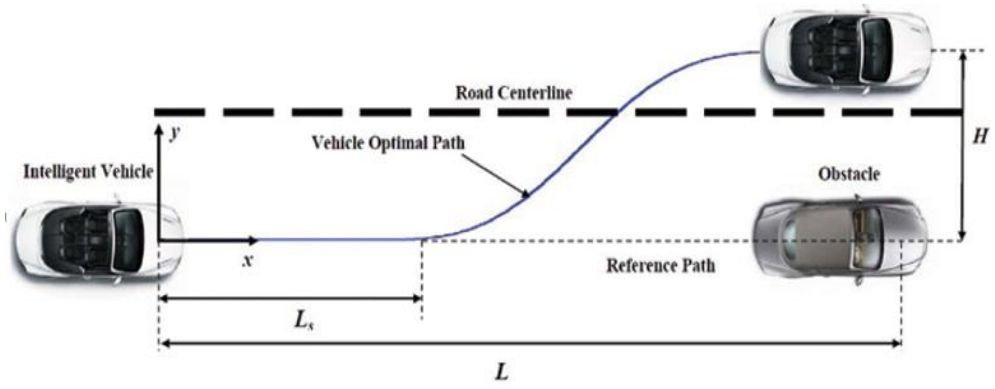


图 2-1.轨迹跟踪

如图 2-1 所示为轨迹跟踪功能，该功能的实现需要纵横向联合控制。通过控制方向盘转角跟踪期望的轨迹，完成横向的路径跟踪；通过控制扭矩和制动压力跟踪期望的速度，实现速度的跟驰，实现轨迹跟踪功能。

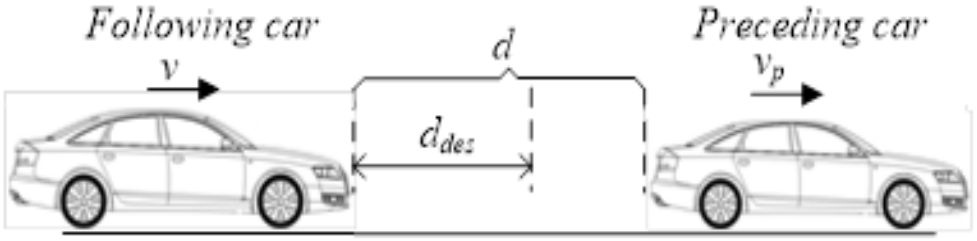


图 2-2.自适应巡航

如图 2-2 所示为自适应巡航功能（ACC），该功能的实现主要依托车辆的纵向控制。通过参照驾驶员的期望行车距离，结合驾驶员动态行为、燃油经济型、驾驶舒适性等设计控制率。常用的方式有恒定车距巡航和恒定时距巡航。自适应巡航控制目前多用于高速公路直线工况的前车巡航跟踪。

本合作项目中要实现的功能为轨迹跟踪功能，实现图 1-1 所示的转弯工况的轨迹跟踪。轨迹跟踪功能的实现中，采用分别设计纵向控制器和横向控制器的方案。纵向部分分为上下位控制器，其中上位控制器在巡航模型基础上进行了修改，给定期望的速度通过 MPC 算法得到期望的加速度；下位控制器采用逆模型的方式，由期望的加速度计算得到扭矩和制动压力。横向控制中，采用了二自由度动力学模型，给定期望的参考轨迹通过 MPC 控制算法得到方向盘转角。

3 模型预测控制简介（MPC）

模型预测控制（Model Predictive Control , MPC）是指在每一采样时刻，根据系统当前状态，利用模型，预测系统的未来输出，与期望状态轨迹比较，构建含有多约束的控制问题，将求解的控制序列的第一个元素作用于系统，预测时域不断向前滚动优化。从局部看，MPC 在某个预测时域内求解的值为最优的，从全局看 MPC 求解的值为次优的，但因其滚动优化可以减小误差，提高控制精度。

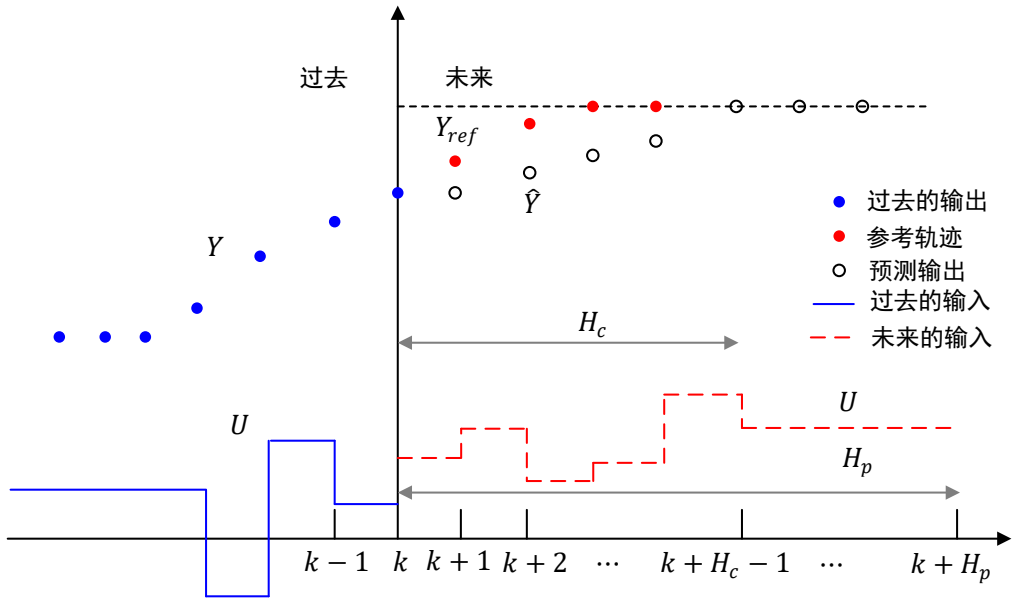


图 3-1. MPC 算法框架

如图 3-1 所示为 MPC 算法的基本框架，通过预测系统的未来的动态计算最优控制率。MPC 算法的基本组成为：

(1).预测模型

预测控制具有预测功能，即能够根据系统当前时刻的状态输入预测系统未来的输出，因此，需要一个描述系统动态行为的模型作为预测模型。

预测模型具有展示过程未来动态行为的功能，这样就可像在系统仿真时那样，任意的给出未来控制策略，观察过程不同控制策略下的输出变化，从而为比较这些控制策略的优劣提供了基础。

(2).反馈校正

在预测控制中，采用预测模型进行过程输出值的预估只是一种理想的方式。而在实际过程中，由于存在非线性、模型失配和干扰等不确定因素，使基于模型的预测不可能准确地与实际相符。因此，在预测控制中，通过输出的测量值$Y(k)$与模型的预估值$Y_{ref}(k)$进行比较，得出模型的预测误差，再利用模型预测误差来对模型的预测值进行修正。

由于对模型施加了反馈校正的过程，使预测控制具有很强的抗扰动和克服系统不确定性的能力。预测控制不仅考虑了模型，而且利用了反馈信息，使预测控制成为一种闭环优化控制算法。

(3).滚动优化

预测控制是一种优化控制算法，需要通过某一性能指标的最优化来确定未来的控制作用。这一性能指标还涉及到过程未来的行为，它是根据预测模型由未来的控制策略决定的。

预测控制中的优化与通常的离散最优控制算法不同，它并非采用一成不变的全局最优目标，而是采用滚动式的有限时域优化策略。即优化过程不是一次离线完成的，而是反复在线进行的。在每一采样时刻，优化性能指标只涉及从该时刻起到未来有限的时间，而到下一个采样时刻，这一优化时段会同时向前。所以，预测控制不是用一个对全局相同的优化性能指标，而是在每一个时刻有一个相对于该时刻的局部优化性能指标。

(4).参考值

在预测控制中。考虑到过程的动态特性，为了使过程避免出现输入和输出的急剧变化，往往要求过程输出 y(k)沿着一条期望的、平缓的曲线达到设定值 r。这条曲线通常称为参考轨迹 y。

模型预测控制的优点为：无需精确模型、显示处理约束、预测系统动态。

4 二次规划简介（QP）

二次规划是非线性规划中的一类特殊数学规划问题，在很多方面都有应用，如投资组合、约束最小二乘问题求解、序列二次规划在非线性优化问题中应用等。二次规划的一般形式可以表示为公式（4-1）：

$$\min_x \frac{1}{2}x^T Px + q^T x + r \quad \text{s.t. } Gx \leq h,\ Ax = b \tag{4-1}$$

其中$P \epsilon S_+^n$，$G \in R^{m\times n}$，$A \in R^{p\times n}$。求解二次规划问题时，我们在一个多面体集中最小化代价函数，如图 4-1 所示。

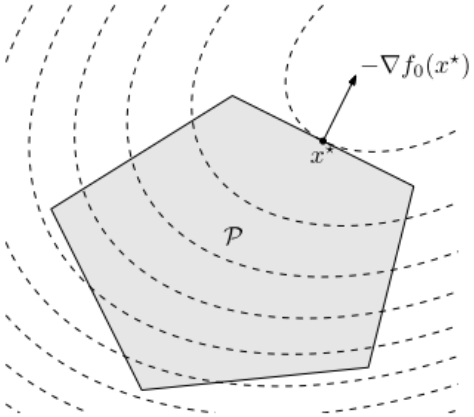


图 4-1.QP 问题的几何阐述

图中的阴影部分为可行域$\mathcal{P}$，虚线为代价函数的等高线，图中的x^*为最优解。

对于二次规划问题的求解，有拉格朗日乘子法、有效集法（积极集法）、内点法等，下面将简单介绍这几种方法的思路。

拉格朗日法主要用于求解等式约束的优化问题，例如形如（4-2）中的二次规划问题：

$$\min_x \frac{1}{2}x^T Px + q^T x + r \quad \text{s.t. } Ax = b \tag{4-2}$$

通过引入拉格朗日乘子λ，将（4-2）变换为（4-3）

$$\min L(x,\lambda) = \frac{1}{2}x^T Px + q^T x + r - \lambda^T(Ax - b) \tag{4-3}$$

令

$$\nabla_x L(x,\lambda) = 0,\ \nabla_\lambda L(x,\lambda) = 0 \tag{4-4}$$

得到优化问题的解。拉格朗日乘子法是一种经典的求解条件极值的解析方法，可将所有的等式约束问题转化为无约束极值问题求解。

有效集法和内点法为求解 QP 问题的数值解法，通过迭代寻优。拉格朗日乘子法可以求解等式约束问题不能解决不等式约束。有效集法和内点法都可用于求解带有不等式约束的二次规划问题，他们的思路均为将不等式约束变为只有等式约束的优化问题。有效集法的思

路是：剔除未有效的不等式约束，得到最优点处的有效约束，将原问题转化成更易求解的等式约束命题，该方法的主要工作是找到最优点处的有效约束。而内点法的思路为，将不等式约束近似为 indicator function ，使原问题变为只包含等式约束的优化问题。

$$\begin{aligned} &min\, f_0(x) \\ &\text{s.t. } f_i(x) \le 0, i = 1, \dots, m,\ Ax = b \end{aligned}$$	（4-5）

如（4-5）所示为带有不等式约束的凸优化问题，将不等式约束用 indicator function 近似可将（4-5）写为：

$$\begin{aligned} &min\, f_0(x) + \sum_{i=1}^{m} I_-(f_i(x)) \\ &\text{s.t. } Ax = b \end{aligned}$$	（4-6）

其中$I_-: R \to R$是 indicator function 可写为（4-7）的形式

$$I_-(\mathrm{u}) = \begin{cases} 0 & u \le 0 \\ \infty & u \ge 0 \end{cases}$$	（4-7）

即满足不等式约束时和原优化问题相同，不满足约束时就会使代价函数趋于无穷，这样将不等式约束的优化问题转化为只有等式约束的优化问题再进行求解。

5 Yalmip 工具箱简介

Yalmip 是 Lofberg 开发的一种免费的优化求解工具。其最大的特色在于将外界多种最优化求解器进行集成，形成一种统一的建模求解语言，提供了 Matlab 的调用 API。Yalmip 在安装时，打开 Matlab 软件添加 path，安装后可以通过 yalmiptest 指令在 Matlab 界面查看是否安装成功。

在使用 Yalmip 工具箱求解优化问题时分为下面 5 步：

(1).定义变量

在 Yalmip 中常用的为默认格式变量，如：$x = sdpvar(m, n, [option])$。其他的变量还有整数格式变量，如$x = intvar(m, n, [option])$和 0-1 格式变量，如$x = binvar(m, n, [option])$.

(2).设定目标函数

求解优化问题时，一般都是包含目标函数（代价函数）和约束。在编写目标函数时，直接写为$objective = \frac{1}{2}x^T px + p^T x + r$的形式即可。

(3).设定约束条件

Yalmip 中设定约束条件可直接写为$\text{Constraints} = [\text{sun(x)} \leq 10, \text{x(1)} == 0]$的形式

(4).设定求解要求及参数

命令语句：$\text{sdpsettings(option1, value1, option2, value2, \dots)}$，例如可以写为$\text{ops} = \text{sdpsettings('solver', 'gurobi', 'verbos', 2)}$。'solver'指定优化算法采用gurobi求解器求解，若不指定求解器，Yalmip 会自动根据决策变量类型挑选已安装的、最合适的求解器。'verbose'指定显示冗余度，冗余度越大就可以看到越详细的求解过程信息。

(5).求解

采用指令$\text{result} = \text{optimize(Constraints, Objective, options)}$即可完成优化问题求解。

6 MPC 问题等价为 QP 问题

在第三节的控制器设计中，横向控制器和纵向的上位控制器均采用 MPC 控制方法，通过构建代价函数求解最有控制问题得到控制量。本小节主要介绍将 MPC 问题等价为 QP 问题。

$$J(U,x_t)=x_{N|t}^T P x_{N|t}+\sum_{k=0}^{N-1} x_{k|t}^T Q x_{k|t}+u_{k|t}^T R u_{k|t} \quad (6\text{-}1)$$

（6-1）所示为一般线性 MPC 的代价函数，其中$P=P^T>0$，$Q=Q^T>0$，$R=R^T>0$为权重系数。

$$x_{k+1|t}=\mathrm{A}x_{k|t}+Bu_{k|t} \quad (6\text{-}2)$$

（6-2）所示为线性状态空间模型，其中$x_{k|t}\in R^n$，$u_{k|t}\in R^m$。由（6-2）推导得到

$$x_{k|t}=A^k x_{0|t}+\sum_{j=0}^{k-1} A^i B u_{k-1-j|t} \quad (6\text{-}3)$$

由（6-3）推导可以得到

$$\begin{bmatrix} x_{1|t} \\ x_{2|t} \\ \cdots \\ x_{N|t} \end{bmatrix}=\begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \cdots & 0 \\ \cdots & \cdots & \cdots & 0 \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix}\begin{bmatrix} u_{0|t} \\ u_{1|t} \\ \cdots \\ u_{N-1|t} \end{bmatrix}+\begin{bmatrix} A \\ A^2 \\ \cdots \\ A^N \end{bmatrix}x_{0|t} \quad (6\text{-}4)$$

令

$$\bar{S}=\begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \cdots & 0 \\ \cdots & \cdots & \cdots & 0 \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix},\ U=\begin{bmatrix} u_{0|t} \\ u_{1|t} \\ \cdots \\ u_{N-1|t} \end{bmatrix},\ \bar{T}=\begin{bmatrix} A \\ A^2 \\ \cdots \\ A^N \end{bmatrix} \quad (6\text{-}5)$$

将（6-1）中的代价函数展开得到：

$$\begin{aligned} J=x_{0|t}^T Q x_{0|t}&+\begin{bmatrix} x_{1|t} \\ x_{2|t} \\ \cdots \\ x_{N-1|t} \\ x_{N|t} \end{bmatrix}^T\begin{bmatrix} Q & 0 & 0 & \cdots & Q \\ 0 & Q & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & Q & 0. \\ 0 & 0 & \cdots & 0 & P \end{bmatrix}\begin{bmatrix} x_{1|t} \\ x_{2|t} \\ \cdots \\ x_{N-1|t} \\ x_{N|t} \end{bmatrix} \\ &+\begin{bmatrix} u_{0|t} \\ u_{1|t} \\ \cdots \\ u_{N-1|t} \end{bmatrix}^T\begin{bmatrix} R & 0 & \cdots & 0 \\ 0 & R & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & R \end{bmatrix}\begin{bmatrix} u_{0|t} \\ u_{1|t} \\ \cdots \\ u_{N-1|t} \end{bmatrix} \end{aligned} \quad (6\text{-}6)$$

令

$$\bar{Q}=\begin{bmatrix} Q & 0 & 0 & \cdots & Q \\ 0 & Q & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & Q & 0. \\ 0 & 0 & \cdots & 0 & P \end{bmatrix},\ \bar{R}=\begin{bmatrix} R & 0 & \cdots & 0 \\ 0 & R & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & R \end{bmatrix} \quad (6\text{-}7)$$

将（6-4）（6-5）（6-7）带入（6-6）得到

$$\begin{aligned} J&=(\bar{S}U+\bar{T}x_0)^T\bar{Q}(\bar{S}U+\bar{T}x_0)+U^T\bar{R}U+x_0^T\mathrm{Q}x_0 \\ &=\frac{1}{2}U^T 2(\bar{R}+\bar{S}^T\bar{Q}\bar{S})U+x_{0|t}^T 2\bar{T}^T\bar{Q}\bar{S}U+\frac{1}{2}x_{0|t}^T 2(Q+\bar{T}^T\bar{Q}\bar{T})x_{0|t}^T \end{aligned} \quad (6\text{-}8)$$

令$H=2(\bar{R}+\bar{S}^T\bar{Q}\bar{S})$，$F^T=2\bar{T}^T\bar{Q}\bar{S}$，$Y=2(Q+\bar{T}^T\bar{Q}\bar{T})$则（6-1）中的代价函数变为

$$J(U,x_t)=\frac{1}{2}U^THU+x_{0\|t}^TF^TU+\frac{1}{2}x_{0\|t}^TYx_{0\|t}$$	（6-9）

通过上述推导，代价函数已经变为了标准的 QP 问题形式。对于带约束的 MPC 问题，其约束部分也要整理为标准形式。假设上述代价函数存在状态约束和输入约束，如（6-10）所示。

$$u_{min}\le u_{k\|t}\le u_{max},\ k=0,\dots,N-1$$ $$x_{min}\le x_{k\|t}\le x_{max},\ \mathrm{k}=1,\dots,\mathrm{N}$$	（6-10）

对于输入约束可整理为

$$u_{k\|t}\le u_{max},\ k=0,\dots,N-1$$ $$-u_{k\|t}\le -u_{min},\ k=0,\dots,N-1$$	（6-11）

将输入约束在预测时域N展开得到

$$\begin{bmatrix}1&0&\cdots&0\\0&1&\dots&0\\\cdots&\cdots&\dots&\dots\\0&\cdots&0&1\\-1&0&\cdots&0\\0&-1&\cdots&0\\\cdots&\cdots&\cdots&\cdots\\0&\cdots&0&-1\end{bmatrix}\begin{bmatrix}u_{0\|t}\\u_{1\|t}\\\cdots\\u_{N-1\|t}\end{bmatrix}\le\begin{bmatrix}u_{max}\\u_{max}\\\cdots\\u_{max}\\-u_{min}\\-u_{min}\\\cdots\\-u_{min}\end{bmatrix}$$	（6-12）

由（6-12）得到标准形式$G_1U\le W_1$

对于状态约束可整理为

$$x_{k\|t}=A^kx_{0\|t}+\sum_{j=0}^{k-1}A^jBu_{k-1-j\|t}\le x_{max},\ k=1,\dots,N$$ $$-x_{k\|t}=-A^kx_{0\|t}-\sum_{j=0}^{k-1}A^jBu_{k-1-j\|t}\le -x_{min},\ k=1,\dots,N$$	（6-13）

将状态约束在预测时域N展开得到

$$\begin{bmatrix}B&0&\cdots&0\\AB&B&\dots&0\\\cdots&\cdots&\dots&\dots\\A^{N-1}B&A^{N-2}B&\cdots&B\\-B&0&\cdots&0\\-AB&-B&\cdots&0\\\cdots&\cdots&\cdots&\cdots\\-A^{N-1}B&A^{N-2}B&\dots&-B\end{bmatrix}\begin{bmatrix}u_{0\|t}\\u_{1\|t}\\\cdots\\u_{N-1\|t}\end{bmatrix}\le\begin{bmatrix}y_{max}\\y_{max}\\\cdots\\y_{max}\\-y_{min}\\-y_{min}\\\cdots\\-y_{min}\end{bmatrix}-\begin{bmatrix}A\\A^2\\\cdots\\A^N\\-A\\-A^2\\\cdots\\-A^N\end{bmatrix}x_{0\|t}$$	（6-14）

由（6-14）得到标准形式$G_1U\le W_2+S_2x_{0\|t}$。整理输入约束的标准形式和状态约束的标准形式得到：

$$\begin{bmatrix}G_1\\G_2\end{bmatrix}U\le\begin{bmatrix}W_1\\W_2\end{bmatrix}+\begin{bmatrix}0\\S_2\end{bmatrix}x_{0\|t}$$	（6-15）

7 数学模型和控制器设计：

（根据对研究对象所观察到的现象及实践经验，归结成一套反应其内部因素数量关系的数学公式，逻辑准则和具体算法，用以描述和研究客观现象的运动规律。）

纵横向轨迹跟踪控制器的设计采用单独设计的方式，即单独设计横向控制器控制方向盘转角和单独设计纵向控制器控制电机的驱动扭矩和制动压力。其中横向控制采用二自由度动力学模型，建立前轮转角和方向盘转角的比例关系，通过控制方向盘转角实现车辆的横向控制；纵向控制采用分层控制的思想，上位控制器采用三阶运动学模型，输出期望的加速度，下位控制器采用逆模型的方式由期望加速度计算得到电机扭矩和制动压力。

7.1 横向动力学模型

如图 7-1 所示为车辆横向控制的二自由度自行车模型，描述车辆的侧向和横摆运动。做出如下假设：（1）左右车轮的力由于对称性而集中在单个车轮上；（2）自行车模型描述了车辆的二维平面运动，忽略车辆的俯仰和侧倾运动；（3）后轮作为车辆的驱动轮，前轮只用于转向；（4）忽略转向及悬架倾斜引起的左右车轮的载荷变化。

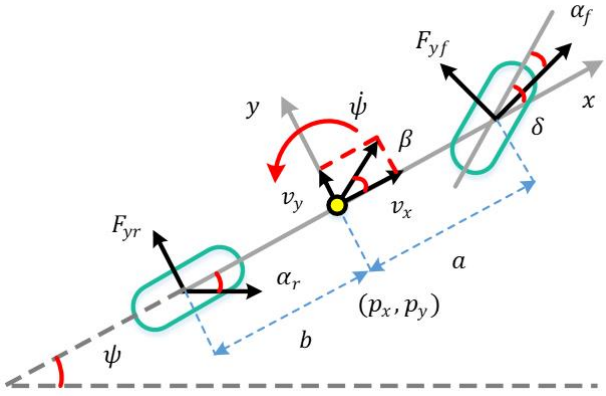


图 7-1.二自由度自行车模型

由图 7-1 可知，二自由度汽车受到的外力沿y轴方向的合力与绕质心的力矩和为：

$$ma_y = F_{yf}\cos\delta + F_{yr}$$ $$I_{zz}\ddot{\psi} = aF_{yf}\cos\delta - bF_{yr}$$	（7-1）

在惯性参考系中，车辆的横向加速度为：

$$a_y = \dot{v}_y + \dot{\psi}v_x$$	（7-2）

通过小角度假设（质心侧偏角很小）得到：

$$\beta = \tan^{-1}\left(\frac{v_y}{v_x}\right) \approx \frac{v_y}{v_x}$$	（7-3）

根据坐标系规定，前后轮侧偏角为:

$$\alpha_f = \beta + \frac{a\dot{\psi}}{v_x} - \delta$$ $$\alpha_r = \beta - \frac{b\dot{\psi}}{v_x}$$	（7-4）

侧偏力与侧偏角的关系为：

$$F_{yf}=k_1\alpha_f$$ $$F_{yr}=k_2\alpha_r$$	（7-5）

将公式（7-4）和（7-5）带入（7-1）得到二自由度汽车的运动微分方程：

$$(k_1+k_2)\beta+\frac{1}{v_x}(ak_1-bk_2)\dot{\psi}-k_1\delta=m(\dot{v}_y+v_x\dot{\psi})$$ $$(ak_1-bk_2)\beta+\frac{1}{v_x}(a^2k_1+b^2k_2)\dot{\psi}-ak_1\delta=I_{zz}\ddot{\psi}$$	（7-6）

其中，m是车辆质量，v_x是车辆坐标系下车辆的纵向速度，v_y是车辆坐标系下车辆的横向速度，β是质心侧偏角，I_{zz}是绕z轴的横摆惯性力矩，$\dot{\psi}$是横摆角速度，F_{yf}是前轮侧偏力，F_{yr}是后轮侧偏力，a是质心距前轴的距离，b是质心距后轴的距离，α_f是前轮侧偏角，α_r是后轮侧偏角，δ是前轮转角，k_1是前轮侧偏刚度，k_2是后轮侧偏刚度。

7.2 横向控制器设计

选取v_y、$\dot{\psi}$、y、ψ为状态量，由（7-2）、（7-3）、（7-6）得到：

$$\dot{v}_y=\frac{k_1+k_2}{mv_x}v_y+\left(\frac{ak_1-bk_2}{mv_x}-v_x\right)\dot{\psi}-\frac{k_1}{m}\delta$$ $$\ddot{\psi}=\frac{ak_1-bk_2}{I_{zz}v_x}v_y+\frac{a^2k_1-b^2k_2}{I_{zz}v_x}\dot{\psi}-\frac{ak_1}{I_{zz}}\delta$$ $$\dot{y}=v_y+v_x\psi$$	（7-7）

整理得到状态方程：

$$\dot{x}(t)=Ax(t)+Bu(t)$$ $$Y(t)=Cx(t)$$	（7-8）

其中，$x(t)=[v_y(t),\dot{\psi}(t),y(t),\psi(t)]^T$，$Y(t)=[y(t),\psi(t)]^T$。

$$A=\begin{bmatrix}\frac{k_1+k_2}{mv_x} & \frac{a^2k_1-b^2k_2}{I_{zz}v_x}-v_x & 0 & 0\\ \frac{ak_1-bk_2}{I_{zz}v_x} & \frac{a^2k_1-b^2k_2}{I_{zz}v_x} & 0 & 0\\ 1 & 0 & 0 & v_x\\ 0 & 0 & 0 & 1\end{bmatrix},\ B=\begin{bmatrix}\frac{-k_1}{mR_a}\\ \frac{-ak_1}{I_{zz}R_a}\\ 0\\ 0\end{bmatrix},\ C=\begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$	（7-9）

其中，y为横向位置，ψ为横摆角，R_a为方向盘转角和前轮转角的比值，控制量u为方向盘转角。对（7-8）中的连续时间系统离散化得到：

$$x(k+1)=A_dx(k)+B_du(k)$$ $$Y(k)=C_dx(k)$$	（7-10）

其中$A_d=AT+I$，$B_d=TB$，$C_d=C$，T为采样时间，I为单位矩阵。

构造 MPC 控制问题：

$$\min\sum_{i=1}^{N}\|\|Y(k+i\|k)-Y_{ref}(k+i\|k)\|\|_Q^2+\sum_{i=0}^{N-1}\|\|u(k+i\|k)\|\|_R^2$$ $$\text{s.t.}\ x(k+i+1\|k)=A_dx(k+i\|k)+B_du(k+i\|k),i=0,1,\dots N-1$$	（7-11）

其中N是预测步长，k为当前时刻，$Y_{ref}(k+i|k)$为当前时刻向前预测第i步的参考值，$Y(k+i|k)$为当前时刻向前预测第i步的预测值，Q和R为权重系数。

将（7-11）进一步展开

$$\begin{aligned}\min \sum_{i=1}^{N} q_y \left(y(k+i|k)-y_{ref}(k+i|k)\right)^2 \\ + q_\psi \left(\psi(k+i|k)-\psi_{ref}(k+i|k)\right)^2 + \sum_{i=0}^{N-1} ||u(k+i|k)||_R^2 \\ \text{s.t.}\, x(k+i+1|k)=A_d x(k+i|k)+B_d u(k+i|k), i=0,1,\dots N-1\end{aligned} \quad (7\text{-}12)$$

7.3 纵向运动学模型

$$\begin{aligned}\dot{s}&=v+aT\\ \dot{v}&=a\\ \dot{a}&=\frac{1}{\tau}u-\frac{1}{\tau}a\end{aligned} \quad (7\text{-}13)$$

式（7-13）为纵向控制的三阶运动学模型，其中s为位移，v为速度，a为加速度，T为采样时间，u 为期望加速度，τ为车辆实际加速度跟踪期望加速度的时间常数。

7.4 纵向控制器设计

选取s、v、a为状态量， 整理得到状态方程：

$$\begin{aligned}\dot{x}(t)&=Ax(t)+Bu(t)\\ Y(t)&=Cx(t)\end{aligned} \quad (7\text{-}14)$$

其中，$x(t)=[s,v,a]^T, Y(t)=[v]$

$$A=\begin{bmatrix}0 & 1 & T\\ 0 & 0 & 1\\ 0 & 0 & -\frac{1}{\tau}\end{bmatrix},\ B=\begin{bmatrix}0\\0\\\frac{1}{\tau}\end{bmatrix} \quad (7\text{-}15)$$

转弯工况中$C=[0\quad 1\quad 0]$，跟车工况，$C=[1\quad 0\quad 0]$

对（7-14）中连续系统离散化得到：

$$\begin{aligned}x(k+1)&=A_d x(k)+B_d u(k)\\ Y(k)&=C_d x(k)\end{aligned} \quad (7\text{-}16)$$

其中，$A_d=AT+I$，$B_d=TB$，$C_d=C$，I为单位矩阵。

构造 MPC 控制问题：

$$\begin{aligned}\min \sum_{i=1}^{N} ||Y(k+i|k)-Y_{ref}(k+i|k)||_Q^2 + \sum_{i=0}^{N-1} ||u(k+i|k)||_R^2 \\ \text{s.t.}\, x(k+i+1|k)=A_d x(k+i|k)+B_d u(k+i|k), i=0,1,\dots N-1\end{aligned} \quad (7\text{-}17)$$

其中N是预测步长，k为当前时刻，$Y_{ref}(k+i|k)$为当前时刻向前预测第i步的参考值，$Y(k+i|k)$为当前时刻向前预测第i步的预测值，Q和R为权重系数。

将（7-17）进一步展开得到转弯工况代价函数：

$$\begin{aligned}\min \sum_{i=1}^{N} q_v \left(v(k+i|k)-v_{ref}(k+i|k)\right)^2 + \sum_{i=0}^{N-1} ||u(k+i|k)||_R^2 \\ \text{s.t.}\, x(k+i+1|k)=A_d x(k+i|k)+B_d u(k+i|k), i=0,1,\dots N-1\end{aligned} \quad (7\text{-}18)$$

其中控制量u为期望的加速度，$v_{ref}(k+i|k)$为当前时刻向前预测第i步的参考速度，$v(k+i|k)$为当前时刻向前预测第i步的预测速度，q_v和R为权重系数。

将（7-17）进一步展开得到跟车工况代价函数：

$$\min \sum_{i=1}^{N} q_v \left(s(k+i|k) - s_{ref}(k+i|k)\right)^2 + \sum_{i=0}^{N-1} ||u(k+i|k)||_R^2$$
$$\text{s.t.}\, x(k+i+1|k) = A_d x(k+i|k) + B_d u(k+i|k), i = 0,1,\dots N-1 \quad (7\text{-}19)$$

其中控制量u为期望的加速度，$s_{ref}(k+i|k)$为当前时刻向前预测第i步的参考位移，$s(k+i|k)$为当前时刻向前预测第i步的预测位移，q_v和R为权重系数。

7.5 纵向控制逆模型设计

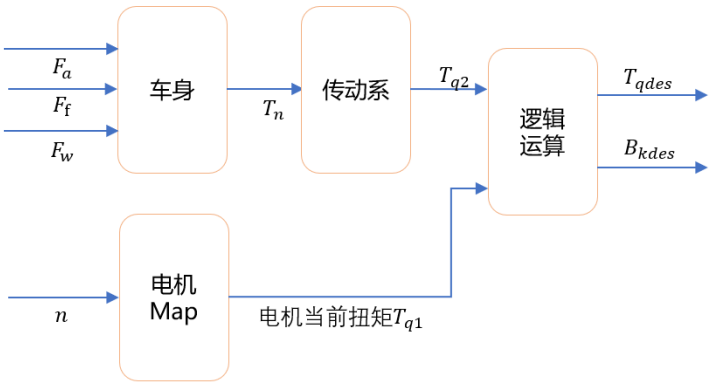


图 7-2.纵向控制逆模型

如图 7-2 所示为纵向控制逆模型。本项目中为城市路面下的轨迹跟踪，假定路面为无坡度阻力的水平路面，同时假定为无风工况，风阻计算中的速度即为当前车速。通过车身受力分析由力矩关系和传动比得到期望的电机扭矩T_{q2}。由电机转速n查表得到电机的当前扭矩T_{q1}，通过控制逻辑得到控制量输出扭矩T_{qdes}和制动压力B_{kdes}。

备注：第三节中的横向动力学模型出自机械工业出版社《汽车理论》第五章的二自由度模型，纵向逆模型部分的车身受力可以在《汽车理论》第一章“汽车的动力性”中查阅。对于纵向上位控制器的巡航控制模型可以参阅《Vehicle Dynamic and Control》，该书第二章节为汽车横向动力学包含有横向运动学模型和横向动力学模型；第四章节为汽车纵向动力学模型；第五章节的巡航控制有助于读者对纵向上位控制器的理解。

8 控制器架构设计

产学研项目合作中的纵横向控制器采用纵向控制器和横向控制器分别设计的思路，横向控制器通过控制方向盘转角实现路径跟踪，纵向控制器通过控制电机扭矩和制动压力实现速度跟踪，完成轨迹跟踪任务。

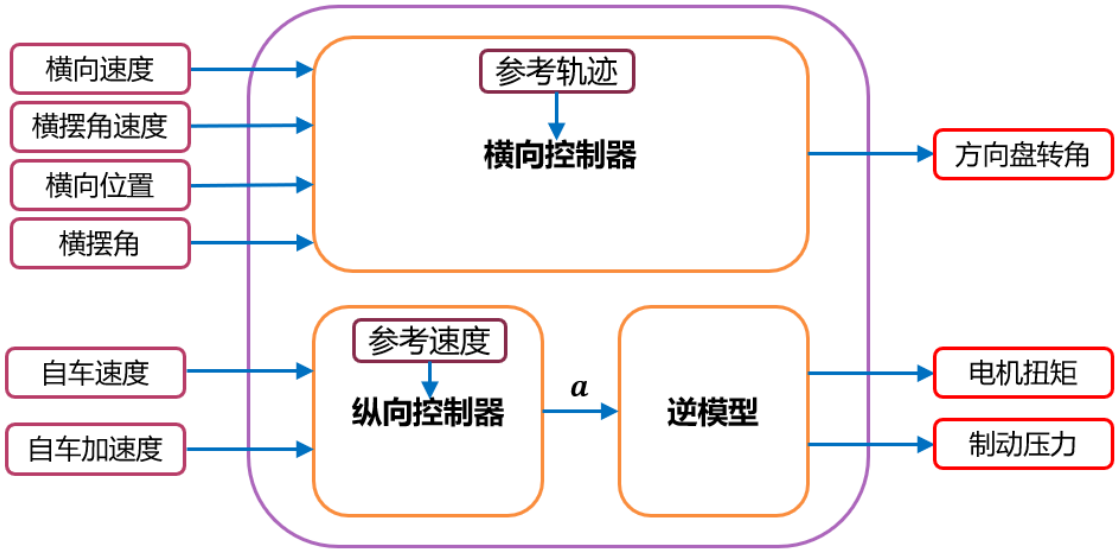


图 8-1.控制器架构

如图 8-1 所示为控制器架构图，横向控制器和纵向控制器解耦分别设计。横向控制器的输入量为横向速度、横摆角速度、横向位置、横摆角，由参考轨迹计算得到期望的横向位置和期望的横摆角，通过使代价函数中的横向位置偏差和横摆角偏差最小求解最优控制率，横向控制器的输出为方向盘转角。纵向控制器的输入为自车速度和自车加速度，由参考速度得到期望的速度，通过使代价函数中的速度偏差最小求解最优控制率，计算得到期望的加速度，将得到的加速度作为输入给入纵向逆模型，输出电机的扭矩和制动压力。在轨迹跟踪模型中，纵向控制的目标是跟随期望的速度。

9 模型搭建和控制算法说明

模型的搭建整体分为 Simulink 模型的建立和 Carsim 仿真环境模型的搭建。在第三节纵横向控制器的设计中，采用了横向控制器和纵向控制器单独设计。

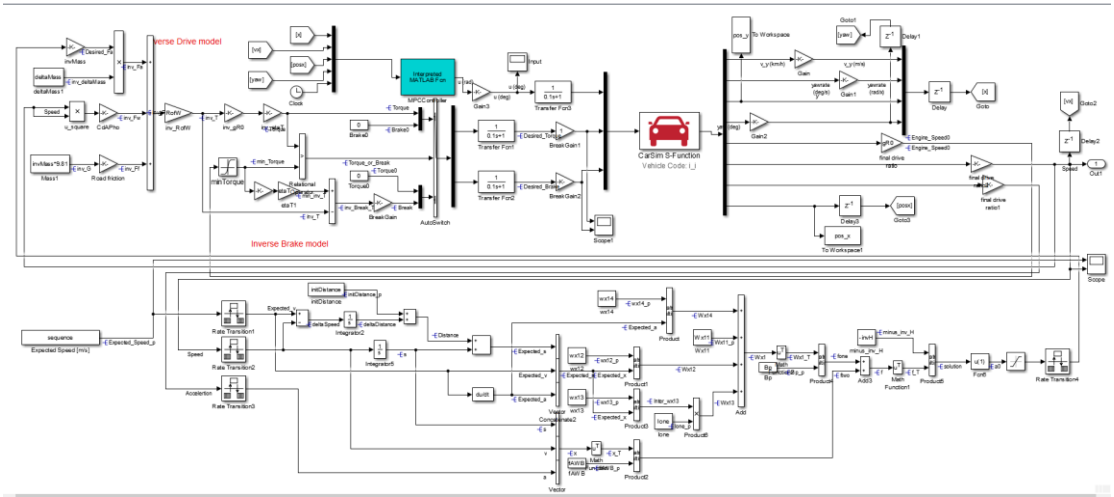


图 9-1.纵横向控制器 Simulink 模型

如图 9-1 所示为整体搭建的 Simulink 模型，模型包含有 Carsim S-Function、纵向上位控制器和纵向逆模型、横向控制器模块，下面将分 Carsim 模型搭建、纵向控制模块、横向控制模块三部分展开介绍。

9.1 Carsim 模型搭建

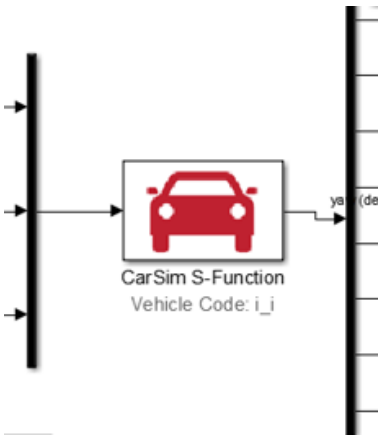


图 9-2.Carsim S-Function 模块

如图 9-2 所示为 Carsim S-Function 模块，左边为从 Simulink 输入到 Carsim 的 3 个控制量，从上到下依次为横向控制量（方向盘转角）和纵向控制量（期望的扭矩和制动压力）。右边为从 Carsim 输出的 8 个状态量，从上到下依次为横向速度、横摆角速度、所在位置的横坐标 Y、横摆角、轮速（用来计算电机转速）、纵向速度、纵向加速度、所在位置的纵坐标 X。通过在 Carsim 中设置接口，将需要的状态信息由 Carsim 输出得到，通过 Simulink 计算将控制量输入 Carsim 进行联合仿真。

关于 Carsim 模型搭建，我的 Carsim 版本为 Carsim 2016.1，不同的版本配置会略有不同。

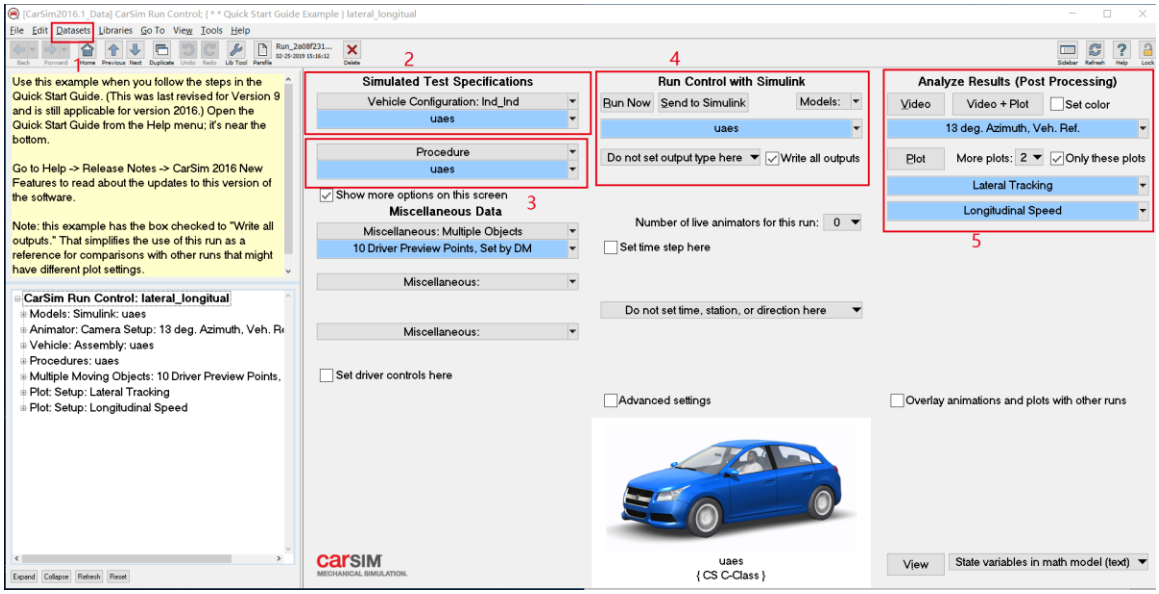


图 9-3.Carsim 主界面

打开 Carsim 进入主界面如图 9-3 所示。在轨迹跟踪的联合仿真中主要用到的是图框中标记的 5 处，先对这几部分进行总体介绍。标号 1 所在的位置为 database，这里是 Carsim 软件的数据库，里面包含有很多软件自带的工况，点击 database，默认所在的位置为 Quick Start Guide Sample->baseline。点击 Duplicate，复制一个新的 database 并命名，图 9-3 的示例中新命名为lateral_longitudinal可参见图的左上方。这里需要说明的是，Carsim 中在右上方点击解锁图标，处于 unlock 状态下才可以修改参数，但是不建议修改 Carsim 自带的参数。每次需要修改参数时，建议将想要修改参数的界面做一个复制即 Duplicate，对新命名的 database 进行修改，防止不能复原 Carsim 本来的参数。标号 2 所在的位置包含有车体的各类参数，比如 powertrain、suspension、tire 等等都可以在标号 2 中进行设置；标号 3 所在的位置包含控制相关设置，比如仿真时常、起始车速、参考轨迹等；标号 4 的部分设置 Carsim 的输入输出信息，与 Simulink 进行数据传输设置；标号 5 为视频和图表演示，用来查看仿真效果。

点击图 9-3 中标号 2 的位置出现如图 9-4 所示的界面，在 PC 端的仿真验证中，仅修改了 Powertrain、Break system 和 Steering system 这三个部分，其他的部分采用了原有的默认设置。在 Simulink 仿真时，横纵向控制器相关的参数与 Carsim 中的参数要对应起来，否则模型失陪会导致控制效果较差甚至出现跑飞现象。对于车身参数的各部分介绍将不详细说明，点击默认的参数查看即可。下面分别介绍修改的三个部分，并对各部分修改做简单说明。

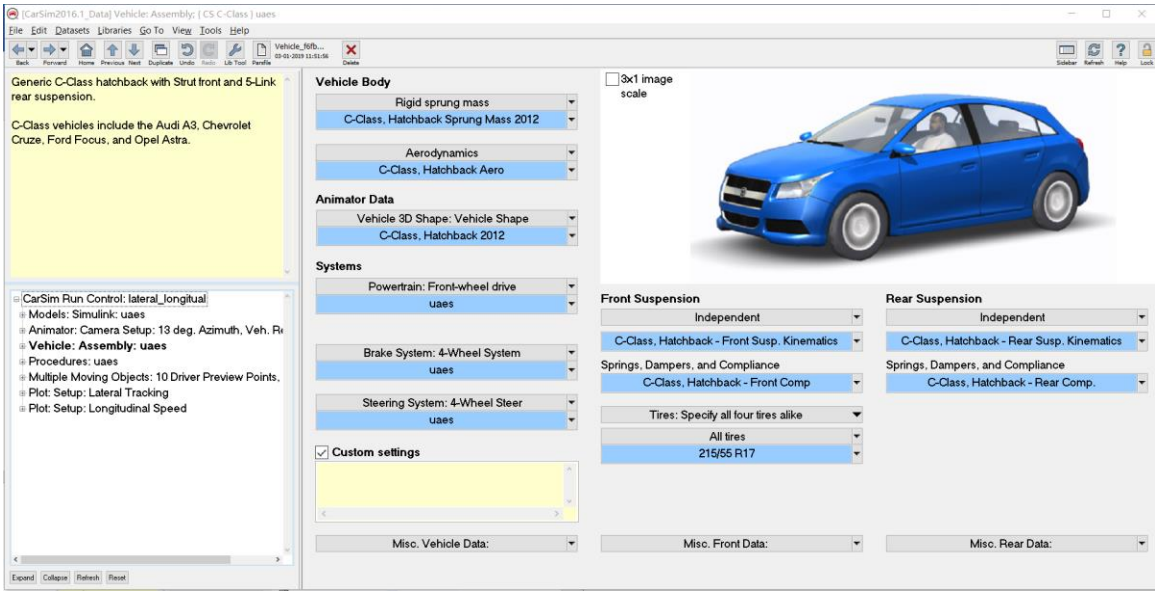


图 9-4.车体参数配置

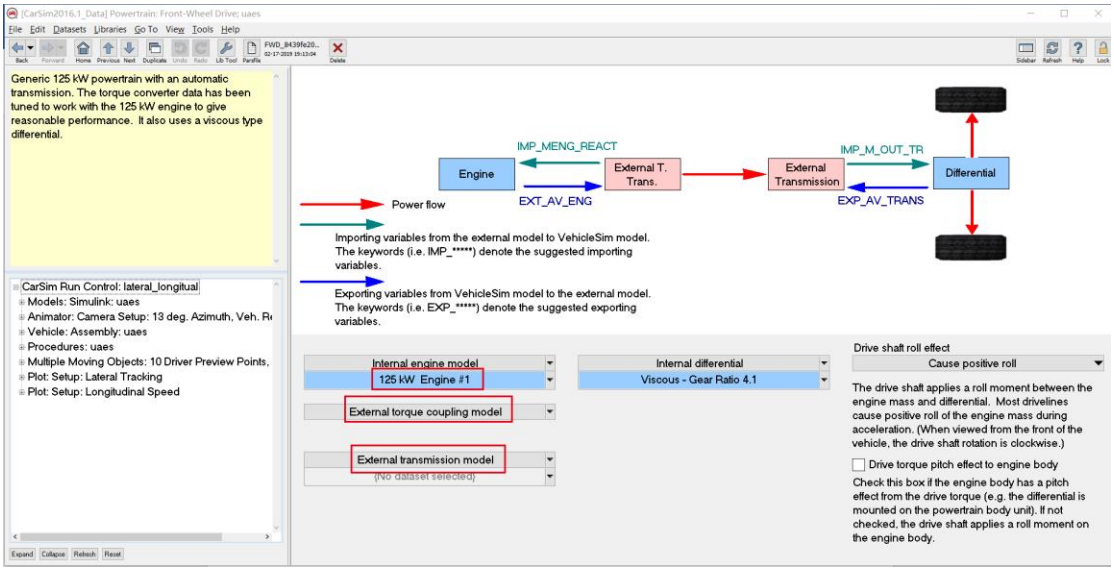


图 9-5.Powertrain 的配置修改

如图 9-5 所示为 Powertrain 的配置截图，Carsim 自带的为发动机模型，本项目中要求修改为电机模型，所以传动系部分仅保留差速器。发动机、离合器、变速箱均不使用。在仿真时发现，发动机模块如果仅改为 External engine model 的配置下，Carsim 仍有动力输出，解决方案为将发动机输出设置为 0 即可。

附录发动机参数

0.0, 0, 0.1, 0.15, 0.2, 0.35, 0.5, 0.7, 0.85, 0.95, 1

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

6100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

6500, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

6700, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

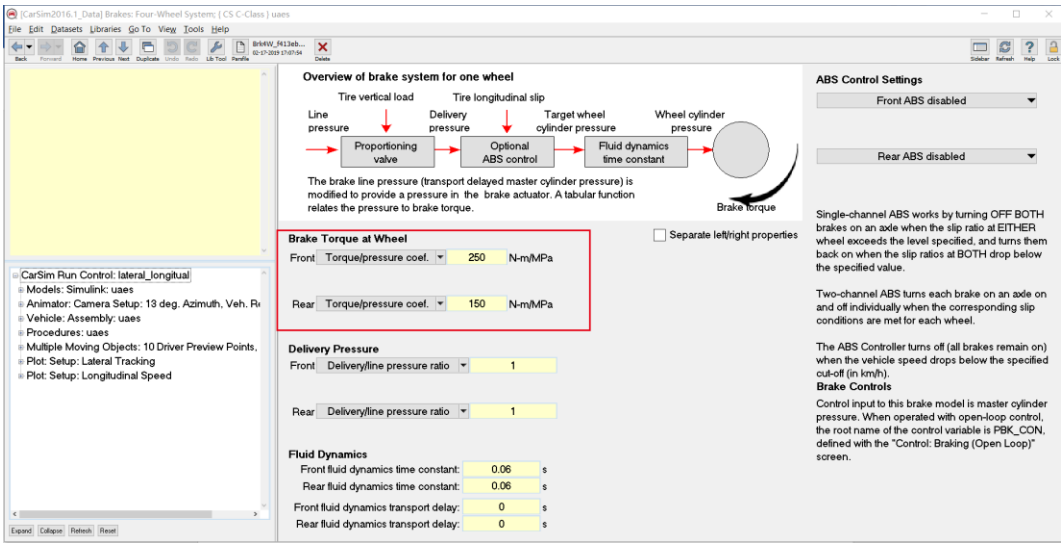


图 9-6.Break system 配置

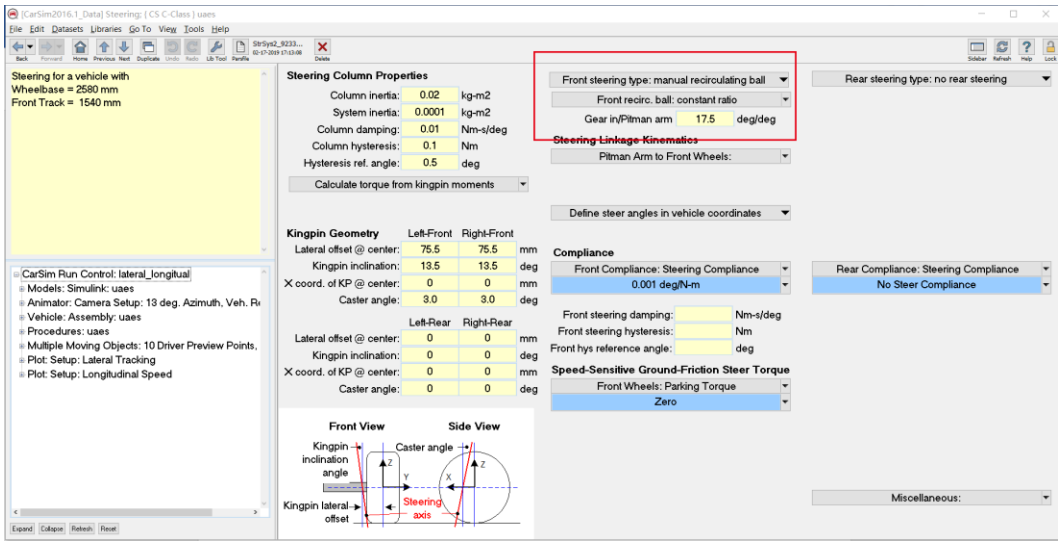


图 9-7.Steering system 配置

如图 9-6 所示为制动系统的参数修改，这里修改的参数为前后轮的制动压力。最大制动压力的大小将影响车辆行驶过程中所能达到的最大减速度。在纵向控制算法中，设置的减速度上界需和 Carsim 中配置相同。如图 9-7 所示为转向系统的配置，这里给出了方向盘转角和前轮转角的比例关系。由于这种比例关系，可通过控制方向盘转角来控制前轮转角实现车辆的横向控制完成转向功能。

点击图 9-3 中标号 3 的位置进入如图 *9-8* 所示界面。由于车辆的纵向速度由纵向控制器控制，在 Driver Controls 中，选择初始速度为 0km/h；Braking 中选择无开环制动压力；Shifting Control 中选择默认配置即可；3D Road 主要是对路面摩擦系数，道路的路障等进行配置，影响最后的演示视频效果。在图 *9-8* 中，需要明确的是 Start and Stop Conditions 这个部分可以设置仿真时长和仿真停止条件；Steering : Driver Path Follower 这个部分设置的为参考路径，由于在联合仿真的控制中，控制量由 Simulink 计算获得，参考轨迹也已经

在 Matlab 中进行设置，故 Carsim 中的参考轨迹仅作为演示使用，可用于轨迹跟踪效果比对。对于轨迹设置部分的参数不在详细叙述，只需将坐标点给出即可。

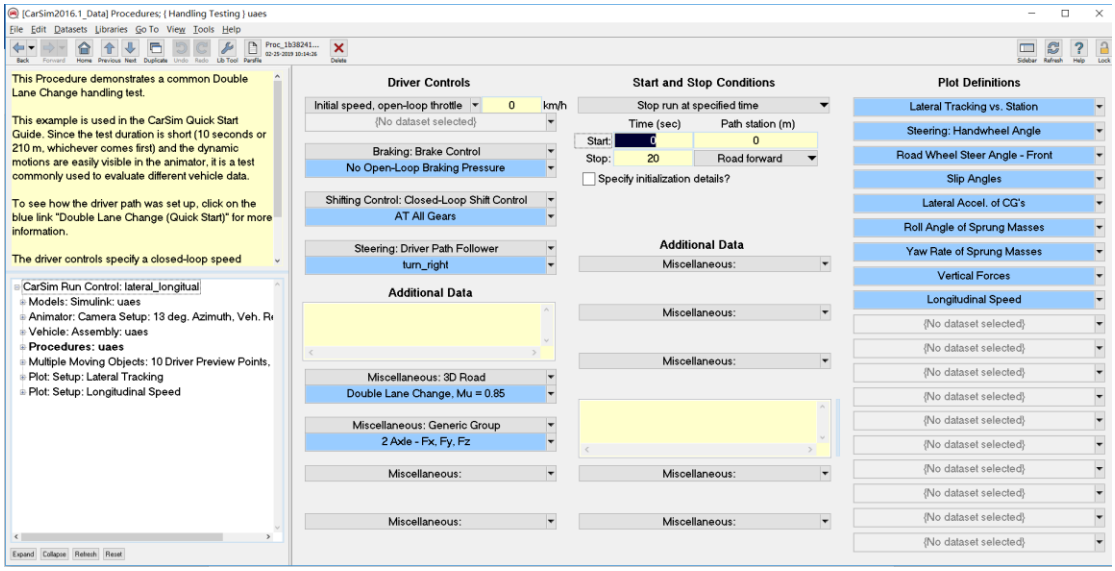


图 9-8.控制参数配置

最后需要设置的为输入输出界面。在图 9-3 的主界面中点击标号 4 位置的 Model 选择 Models: Simulink 选项，通过 Link to New Database 建立新的 database。进入建立的 database 进入输入输出配置界面如图 9-9 所示。右上角红框标注的位置为关联的 Simulink 模型，左下角红框标注的位置为输入输出配置窗口。

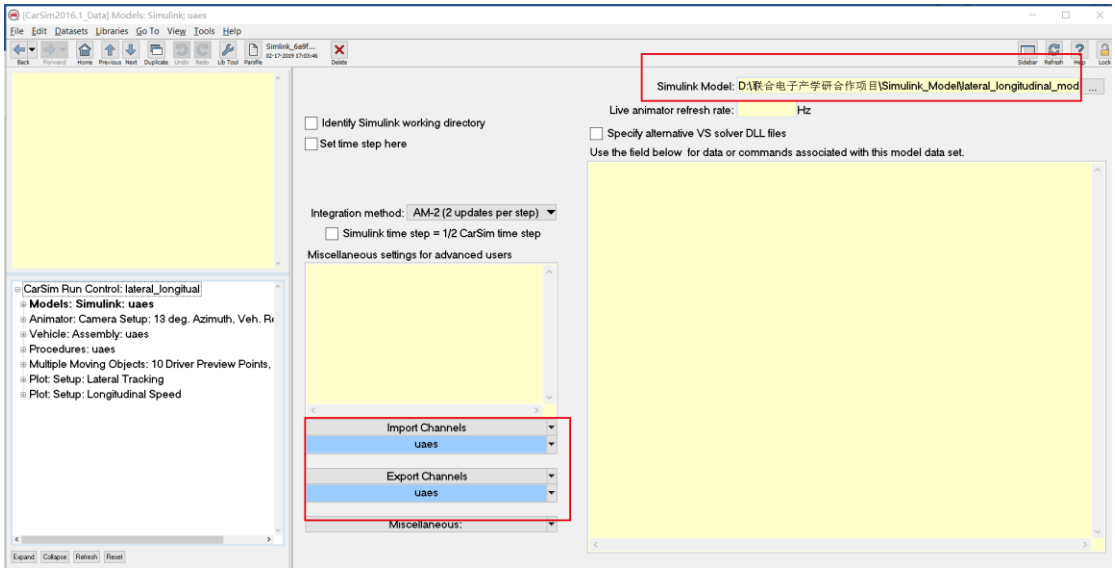


图 9-9.输入输出参数配置界面

点击图 9-9 中 Import Channels 进入输入参数配置界面，点击 Export Channels 进入输出参数配置界面。根据需要配置即可。在本章节最初的部分已经说明了这次仿真所需的输入输出量，读者可参照图 9-10、图 9-11 进行配置。

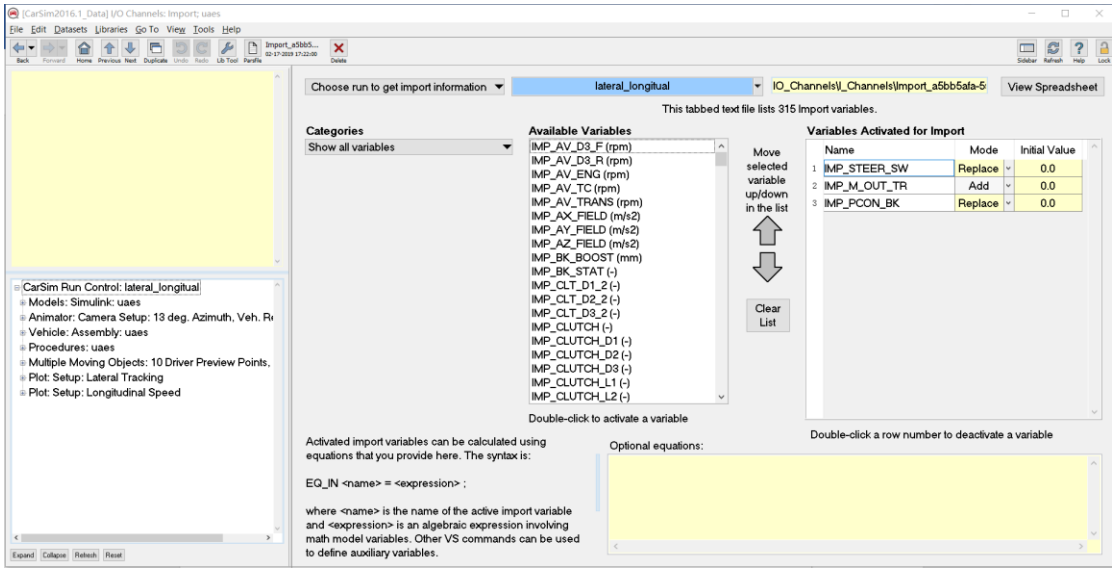


图 9-10.输入界面

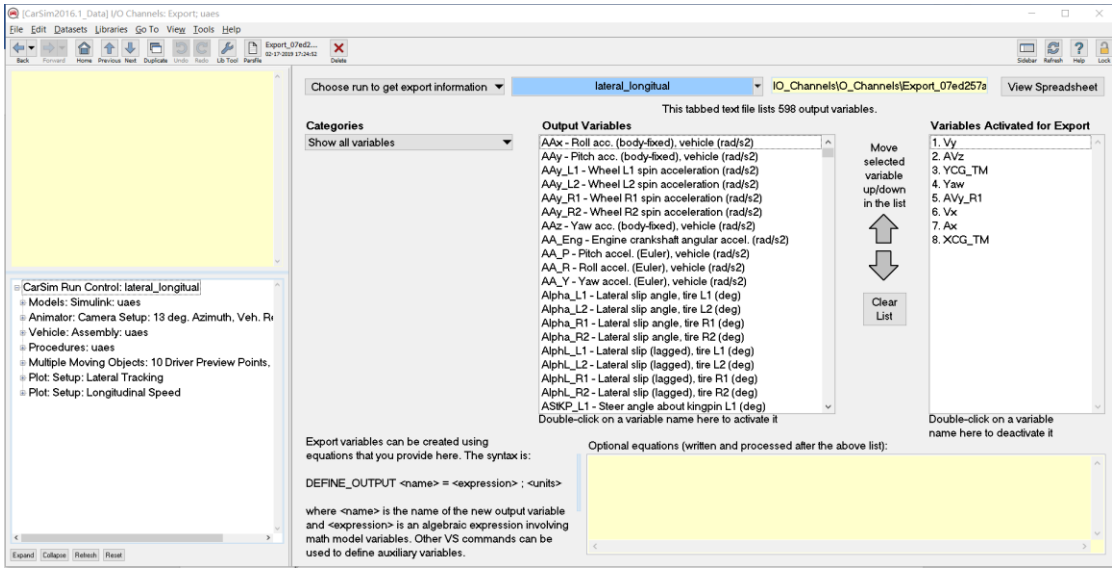


图 9-11.输出界面

主界面中的 video 部分和 plot 部分主要用作可视化仿真结果，无需配置。

9.2 横向控制模块

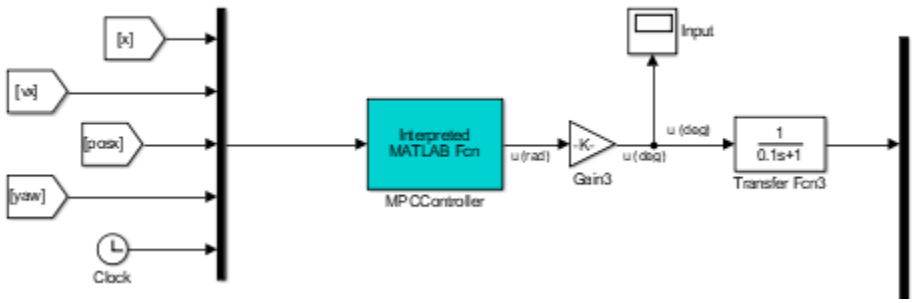


图 9-12.横向控制模块

如图 9-12 所示为横向控制的 Simulink 模型，其中名为 MPCControler 的 S-Function 模块为 MPC 算法部分。左边的部分为 S-Function 中的控制算法所需的输入参数，来自 Carsim 的输出。其中第一项为当前的状态量，包含有横向速度、横摆角速度、横向位置坐标和横摆角四项；第二项为纵向速度；第三项为当前车辆所在的纵坐标；第四项为横摆角；第五项不是 Carsim 的输出，是计时器。S-Function 中是 MPC 控制算法，采用了 Yalmip 求解器，这部分会在接下来的部分附录 m 文件进行详细说明。控制器计算得到方向盘转角，由于计算得到的是弧度角，需要角度变化设有增益环节。图示中的一阶惯性环节是为了是输出更平滑一些，可以起到滤波的功能。

横向控制器的设计过程已经在第三章节详细写出，包含有二自由度动力学模型、构建代价函数、设计控制器等。对于横向控制来讲，不同的工况道理都是相通的，即算法通用性很强。不同的工况，只需要替换参考轨迹，就可以实现轨迹跟踪控制。下面以双移线工况为例，说明 m 文件中的各部分功能。由于 Simulink 中存在 S-Function 模块缺少 tlc 文件无法生成 C++代码，对于横向 C++代码的生成会在下一章介绍 C 代码生成器 CVXGEN 时说明。首先附录完成的横向控制 m 文件。

$function\ uout\ =\ unit\_test(currentx, currentu, positionx, yaw, t)$
$if\ t >= 0;$
$if\ currentu < 0.2;$
$U = 0.2;$
$else$
$U = currentu;$
end
$T\ =\ 0.01;$
$N\ =\ 70;$
$dlc\_x\_A\ =\ [65{:}\,5{:}\,95,105,120{:}\,5{:}\,150,160];$
$dlc\_y\_A\ =\ [0,0.2,0.7,1.6,2.5,3.2,3.4,3.5,3.5,3.2,2.7,1.7,0.8,0.3,0.1,0];$
$X = positionx;$
$r\_A\_y\ =\ zeros(N,1);$
$r\_A\_yaw\ =\ zeros(N,1);$
$r\_A\ =\ zeros(2*N,1);$

```
if X > 0;
for i = 1:N
  if X + U * T * (i) * cos(yaw) >= 65 && X + U * T * (i) * cos(yaw) <= 160
   r_A_y(i) = interp1(dlc_x_A, dlc_y_A, X + U * T * (i) * cos(yaw), 'PCHIP');
   r_A_yaw(i) = (interp1(dlc_x_A, dlc_y_A, (X + U * T * (i) * cos(yaw) + 0.01), 'PCHIP')
                - r_A_y(i))/0.01;
  else
   r_A_y(i) = 0;
   r_A_yaw(i) = 0;
  end
end
else
  for i = 1:N
  r_A_y(i) = 0;
  r_A_yaw(i) = 0;
  end
end
for i = 1:N
  r_A((i - 1) * 2 + 1) = r_A_y(i);
  r_A((i - 1) * 2 + 2) = r_A_yaw(i);
end
  Cf = 39912.6;
  Cr = 72200;
  b = 1.895;
  a = 1.015;
  m = 1270;
  Iz = 1536.7;
  is = 17.5;
  A_c = [-(Cf + Cr)/(m * U), -(a * Cf - b * Cr)/(m * U) - U, 0,0; -(a * Cf - b
                * Cr)/(Iz * U), -(a^2 * Cf + b^2 * Cr)/(Iz * U),0,0; 1,0,0, U; 0,1,0,0];
  B_c = [Cf/(is * m); a * Cf/(is * Iz); 0; 0];
  C_c = [0,0,1,0; 0,0,0,1];
  D_c = [0; 0];
  Vehicle_c = ss(A_c, B_c, C_c, D_c);
  Vehicle = c2d(Vehicle_c, T);
  A = Vehicle.A;
  B = Vehicle.B;
  C = Vehicle.C;
  q_y_A = 36;
  q_yaw_A = 10;
  Q = [q_y_A, 0; 0, q_yaw_A];
```

```
  yalmip('clear')
  u = sdpvar(ones(1,N),ones(1,N));
  x = sdpvar(4*ones(1,N+1),ones(1,N+1));
  r = sdpvar(2*N,1);
  constraints = [];
  objective = 0;
  for k = 1:N
  objective = objective + (r(2*(k-1)+1:2*(k-1)+2) - C*x{k+1})'*Q*(r(2
              *(k-1)+1:2*(k-1)+2) - C*x{k+1}) + u{k}'*u{k};
  constraints = [constraints, x{k+1} == A*x{k} + B*u{k}];
  end
  Controller = optimizer(constraints, objective, [], {x{1}, r}, u{1});
  if Controller{currentx, r_A} <= 7.85
  uout = Controller{{currentx, r_A}};
  else
    uout = 7.85;
  end
end
end
```

Function函数中的输入量，均由Carsim输出，在关于图 9-12的介绍时已有说明。代码中U代表纵向速度，在计算系数矩阵A时，纵向速度为分母项不能为0，而在纵横向联合控制时为从静止——加速——减速——制动这样的工况，起始速度为0会引起计算错误，在代码中做出低速处理。T为采样时间，N为预测步长，dlc_x_A和dlc_y_A分别为参考轨迹的纵坐标和横坐标。$Positionx$表示汽车当前所在位置的纵坐标，r_A_y和r_A_yaw分别为参考的横向位置和横摆角。C_f、C_r为前轮侧偏刚度和后轮侧偏刚度；a、b为前轴到质心的距离和后轴到质心的距离；m 为汽车的质量；Iz 表示转动惯量；is 表示方向盘转角和前轮转角的比例关系。这部分符号表示和设计文档中的符号不是完全一致，请读者和设计文档中标注的含义进行对应。A_c 、B_c 、C_c 、D_c 为系数矩阵，使用 $Vehicle = c2d(Vehicle\_c, T)$ 指令实现连续系统离散化，A、B、C为离散化后的系数矩阵。 q_y_A和 q_yaw_A分别为横向位置和横摆角的惩罚系数。

下面的部分为 Yalmip 工具箱的求解过程，sdpvar 为该工具箱定义变量的方式。定义状态变量，控制变量和参考变量，按照标准形式写入代价函数和约束，使用$optimizer$求解。其中 $Controller = optimizer(constraints, objective, [], \{x\{1\}, r\}, u\{1\})$构建了控制器，输入约束和代价函数后，每次调用仅需要输入状态量x和参考轨迹r即可输出控制量u。最后部分的逻辑判断表示，方向盘转角的最大限度为 7.85 弧度。

9.3 纵向控制模块

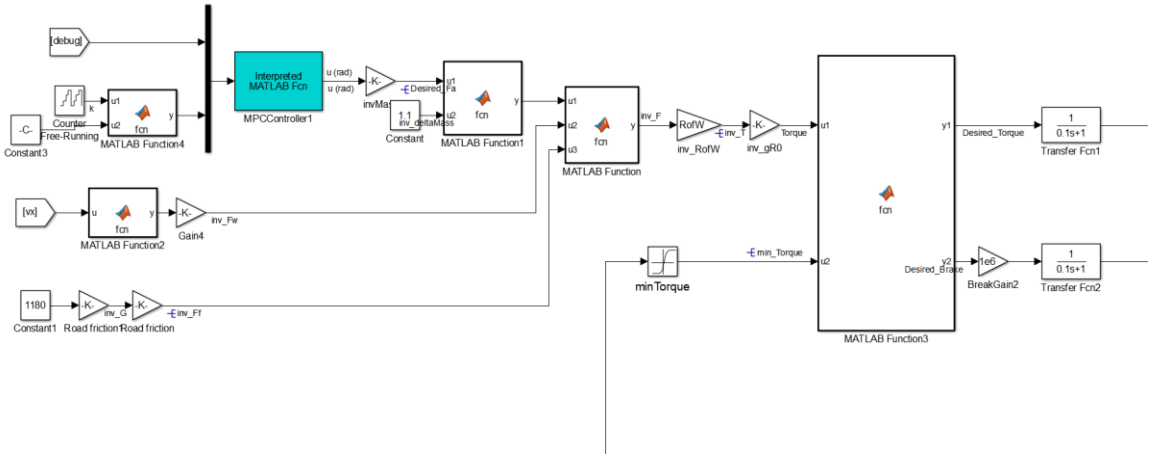


图 9-13.纵向控制模型

如图 9-13 所示为纵向控制模块的 Simulink 模型，其中 MPC 控制算法部分与横向部分类似，通过 MPC 控制器模块计算得到期望的加速度需要经过逆模型得到期望的扭矩或者制动。MPCController1 为纵向 MPC 控制的算法，采用了 S-Function 模块，左边为 S-Function 中函数所需的状态量和参照的速度谱，经过 S-Function 的 MPC 求解器后，通过纵向受力，由加速阻力、滚动阻力、风阻计算得到所需的扭矩。将计算得到的所需扭矩与查表所得的当前电机所需的最小扭矩进行逻辑运算得到期望的扭矩和制动压力发送到 Carsim 模型中。

纵向控制部分的 MPC 求解过程与横向部分相同，均采用 Yalmip 工具箱进行求解，C++ 代码的实现依然采用的是 CVXGEN 工具箱。下面将附录纵向部分的 m 文件，并对代码进行介绍：

首先附录纵向控制 m 文件。

```
function uout = debuger(currentx, currentr)
T = 0.01;
N = 50;
tao = 0.35;
A_c = [0,1,T; 0,0,1; 0,0,-(1/tao)];
B_c = [0; 0; 1/tao];
C_c = [0,1,0];
D_c = 0;
Vehicle_c = ss(A_c, B_c, C_c, D_c);
Vehicle = c2d(Vehicle_c, T);
A = Vehicle.A;
B = Vehicle.B;
C = Vehicle.C;
Wv = 40;
```

```
yalmip('clear')
u = sdpvar(ones(1,N),ones(1,N));
  x = sdpvar(3*ones(1,N+1),ones(1,N+1));
  r = sdpvar(N,1);
  constraints = [];
  objective = 0;
  for k = 1:N
  objective = objective + (r(k)-C*x{k+1})'*Wv*(r(k)-C*x{k+1})+u{k}'
              *u{k};
  constraints = [constraints,x{k+1} == A*x{k}+B*u{k}];
  end
  Controller = optimizer(constraints,objective,[],{x{1},r},u{1});
  uout = Controller{{currentx,currentr}};
end
```

纵向控制的状态量为s、v、a，由于存在$s=vt+\frac{1}{2}at^2$的关系，实际仅需要由 Carsim 输出当前的车速和加速度信息。先根据纵向运动学模型得到连续系统的状态方程，经过离散化后即可采用 Yalmip 工具箱进行求解。关于 Yalmip 工具箱部分内容与章节 5.2 中横向内容部分相同，此处便不再赘述。

10CVXGEN 工具箱介绍

CVXGEN 为斯坦福大开发的 QP 问题求解工具箱，对于中小型 QP 问题可以快速生成 C 代码供嵌入式系统使用。在发布的采用 CVXGEN 工具箱的论文中提到，在嵌入式环境中采用-Os 优化指令可以大幅减少求解器的计算时间。在这次项目合作中，横纵向控制均采用了 CVXGEN 工具箱求解 QP 问题，其中横向部分的预测步长为 100 步没有约束项，纵向部分的预测步长为 50 步没有约束项。在嵌入式 XCU 环境中的计算时间小于 10ms，保证了 HIL 实验的实时求解。

关于 CVXGEN 工具箱本节将主要介绍两方面内容：第一如何使用 CVXGEN 快速生成代码；第二求解结果的正确性证明。

10.1 如何使用 CVXGEN 快速生成代码

在网页搜索 CVXGEN 即可进入界面，首页有使用者指南和一些关于 QP 问题的典型案例比如 Simple QP，SVM，MPC 等。同时在 Introduction 也有一些相关的论文，在本次合作中使用的-Os 优化指令就来自首页的文献。本节主要介绍如何快速生成 MPC 代码。

采用该工具箱生成代码时，只需要按照生成器的格式编写构建的 MPC 问题。在 edit 部分分为维数、参数、变量和代价函数四个部分。在维数部分定义状态维数、预测步长等；在参数部分定义优化问题会用到的参数；变量部分为要求解的未知量比如控制量u，最后就是写明代价函数以及需要的约束。

下面是一个完整的 MPC 问题：

```
dimensions
 m = 1
 n = 3
 N = 70
end

parameters
 A(n,n)
 B(n,m)
 pho_s(m,m) psd
 pho_u(m,m) psd
 x[0](n)
 ref_s[t](1),t = 0..N
end

variables
 x[t](n),t = 1..N+1
 u[t](m),t = 0..N
end

minimize
 sum[t = 0..N-1](quad((ref_s[t] - x[t+1][1]),pho_s) + quad(u[t],pho_u))
subject to
 x[t+1] == A*x[t] + B*u[t],t = 0..N
end
```

其中A、B为系数矩阵，pho_s和pho_u为惩罚系数，在惩罚系数后需要加psd进行标识。$x[0]$为当前时刻状态，$ref_s[t]$为参考量如期望的距离。在 MPC 问题中待求解的控制量 u 以及除当前状态均为所求变量。最后写出所要求解的代价函数和约束就完成了问题的编写，构建问题后点击 generate C 即可生成 C 代码。

CVXGEN 会自动评估矩阵项，过多矩阵运算将导致生成失败，适用于中小型 QP 问题求解。生成的代码后，可通过

10.2 求解结果的正确性证明

PC 环境中，在 Simulink 和 CARSIM 联合仿真时，采用的是 YALMIP 工具箱，利用 MATLAB 中的 QP 求解器来求解优化问题。下面正确性证明采用对比的方式进行展开，构建相同的 MPC 问题，采用同样的权重系数对比计算结果是否相同。经试验，简单的 QP 问题和复杂的 MPC 问题均可用该求解器求解。在接下来的证明中，采用一个较为简单的示例，进行比对，若对正确性存在质疑可根据文档的示例自行构建其他 QP 问题进行验证。

首先构建 MPC 问题：

$$\sum_{t=0}^{N-1}((Yref(t)-C*X(t+1))'*Q*\left(Yref(t)-C*X(t+1)\right)+u(t)'*R*u(t) \quad (10\text{-}1)$$

$$\text{Subject to} \quad \mathrm{x(t+1)=A*x(t)+B*u(t)}$$

如公式（10-1）所示为 MPC 问题，在 Matlab 中采用 5.2 节介绍的横向控制部分代码，为简化问题选取预测步长为 5。下面附录测试所用的 m 文件可在 Matlab 中直接运行。

```
Cf = 33525.29;
Cr = 65178;
a = 1.165;
b = 1.165;
m = 1180;
Iz = 1020;
U = 20;
is = 17.5;
A_c = [-(Cf + Cr)/(m * U), -(a * Cf - b * Cr)/(m * U) - U, 0, 0; -(a * Cf - b * Cr)/(Iz
        * U), -(a^2 * Cf + b^2 * Cr)/(Iz * U), 0, 0; 1, 0, 0, U; 0, 1, 0, 0];
B_c = [Cf/(is * m); a * Cf/(is * Iz); 0; 0];
C_c = [0, 0, 1, 0; 0, 0, 0, 1];
D_c = [0; 0];
Vehicle_c = ss(A_c, B_c, C_c, D_c);
T = 0.01;
Vehicle = c2d(Vehicle_c, T);
A = Vehicle.A;
```

```
B = Vehicle.B;
C = Vehicle.C;
N = 5;
q_y_A = 36;
q_yaw_A = 10;
Q = [q_y_A,0;0,q_yaw_A];
yalmip('clear')
u = sdpvar(ones(1,N),ones(1,N));
x = sdpvar(4*ones(1,N+1),ones(1,N+1));
constraints = [];
objective = 0;
r = [0;0;0;0;0;0;0;0;0;0];
for k = 1:N
objective = objective + (r(2*(k-1)+1:2*(k-1)+2) - C*x{k+1})'*Q*(r(2
                *(k-1)+1:2*(k-1)+2) - C*x{k+1}) + u{k}'*u{k};
constraints = [constraints,x{k+1} == A*x{k} + B*u{k}];
end
Controller = optimizer(constraints,objective,[],x{1},u{1});
currentx = [1; 1;1;1];
uout = Controller{currentx};
```

其中*uout*为最终的计算结果。

在 Matlab 工作区可以看到该问题的全部参数，可自行与下面 C++代码中参数进行比对，需要说明的是，Matlab 中的参考值为r，而 C++代码中对应的参考值为Y_ref。运行上述 m 文件得到计算结果为 -0.1552。

根据上述问题，根据 6.1 节中生成代码的介绍，构建如下的 CVXGEN 问题描述：

```
dimensions
  m = 1   # inputs.
  n = 4   # states.
  q = 2
  T = 5   # horizon.
end
parameters
  A (n,n)  # dynamics matrix.
  B (n,m)  # transfer matrix.
  C (q,n)
  Q (q,q) psd  # state cost.
  R (m,m) psd  # input cost.
```

```
  x[0] (n)   # initial state.
  Y_ref[t](q,m),t = 0..T − 1
end
variables
  x[t] (n),t = 1..T   # state.
  u[t] (m),t = 0..T − 1   # input.
end
minimize
  sum[t = 0..T − 1](quad((Y_ref[t] − C ∗ x[t + 1]),Q) + quad(u[t],R))
subject to
  x[t + 1] == A ∗ x[t] + B ∗ u[t],t = 0..T − 1   # dynamics constraints.
end
```

生成代码后，其中的 solver.h,ldl.cpp,matri_support.cpp,solver.cpp 这四个函数构成求解调用函数。其中的 testsolver.c 为示例 main 函数，在下面的对比验证中，直接对 testsolver.c 文件替换参数进行计算。将 Matlab 中的计算结果赋值给 testsolver.c 文件的对应参数。

下面附录 testsolver.c 的源代码：

```
#include "solver.h"
#include < iostream >
using namespace std;
Vars vars;
Params params;
Workspace work;
Settings settings;
int main(int argc,char ** argv) {
 int num_iters;
 set_defaults();
 setup_indexing();
 load_default_data();
 /* Solve problem instance for the record.*/
 settings.verbose = 0;
 num_iters = solve();
 cout << vars.u_0[0] << endl;
 getchar();
 return 0;
}
void load_default_data(void) {
params.C[0] = 0;
params.C[1] = 0;
```

$params.C[2] = 0;$
$params.C[3] = 0;$
$params.C[4] = 1;$
$params.C[5] = 0;$
$params.C[6] = 0;$
$params.C[7] = 1;$
$params.Y\_ref\_0[0] = 0;$
$params.Y\_ref\_0[1] = 0;$
$/* \ Make\ this\ a\ diagonal\ PSD\ matrix, even\ though\ it's\ not\ diagonal. */$
$params.Q[0] = 36;$
$params.Q[2] = 0;$
$params.Q[1] = 0;$
$params.Q[3] = 10;$
$/* \ Make\ this\ a\ diagonal\ PSD\ matrix, even\ though\ it's\ not\ diagonal. */$
$params.R[0] = 1;$
$params.Y\_ref\_1[0] = 0;$
$params.Y\_ref\_1[1] = 0;$
$params.Y\_ref\_2[0] = 0;$
$params.Y\_ref\_2[1] = 0;$
$params.Y\_ref\_3[0] = 0;$
$params.Y\_ref\_3[1] = 0;$
$params.Y\_ref\_4[0] = 0;$
$params.Y\_ref\_4[1] = 0;$
$params.A[0] = 0.957454032424067;$
$params.A[1] = 0.0171212013097333;$
$params.A[2] = 0.00979428254786027;$
$params.A[3] = 8.71838365860627e-05;$
$params.A[4] = -0.174634161752780;$
$params.A[5] = 0.934869256059343;$
$params.A[6] = 8.89326273660127e-05;$
$params.A[7] = 0.00967341220565915;$
$params.A[8] = 0;$
$params.A[9] = 0;$
$params.A[10] = 1;$
$params.A[11] = 0;$
$params.A[12] = 0;$
$params.A[13] = 0;$
$params.A[14] = 0.2;$
$params.A[15] = 1;$
$params.B[0] = 0.0139457341748303;$
$params.B[1] = 0.0213075991088152;$

$params.B[2] = 8.06864451509295e-05;$
$params.B[3] = 0.000107494051918160;$
$params.x\_0[0] = 1;$
$params.x\_0[1] = 1;$
$params.x\_0[2] = 1;$
$params.x\_0[3] = 1;$
}
其中需要注意的是，Matlab 中矩阵的赋值顺序和 C++代码中矩阵的赋值顺序是不同的，且 A、B 矩阵的参数均为离散后的参数。
运行上述代码最终计算结果为-0.155249，与 Matlab 计算结果相同。验证了 CVXGEN 求解器的正确性。

11横向控制偏差的计算

在验收指标中有一项为横向误差范围，交流发现工程师对该误差的计算存在疑惑，特在此章节对横向偏差的计算方式进行补充说明。

首先需要介绍在横向控制中用到的两段函数——二插法和坐标变换。二插法可以理解为查表的功能，而坐标变换就等于转换坐标系，从全局坐标转换到所需的坐标系下。其中二插法仅适用于横坐标x是递增时的情况，故在横向控制中，如果实际参考轨迹的横坐标非递增的，则需要先经过坐标变换处理，再放入代码中使用。

由于在右转工况和跟车工况中，参考轨迹的横坐标均为递增的，现以左转工况为例进行说明。

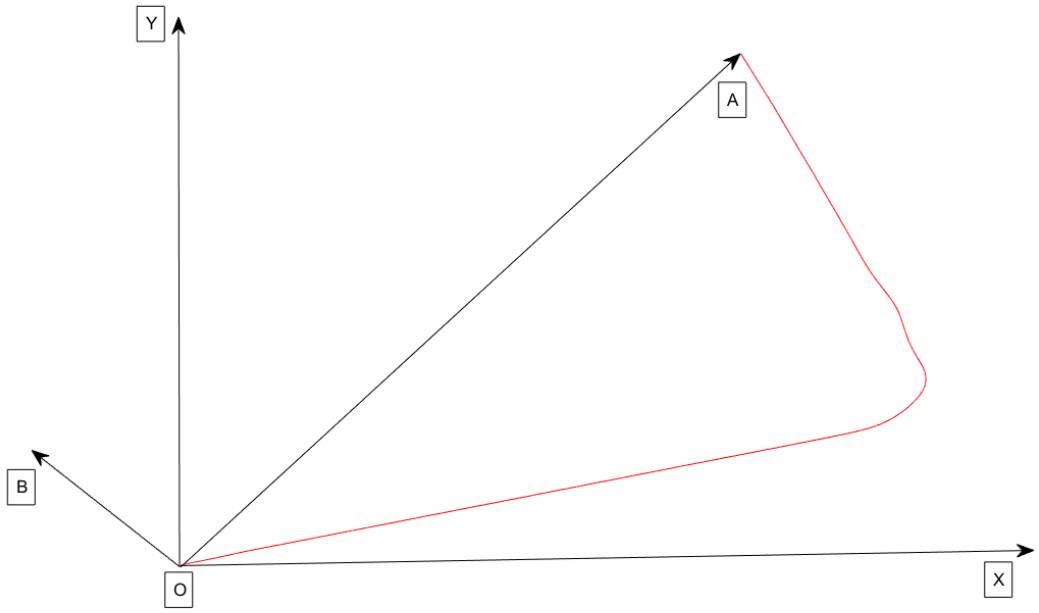


图 11-1.坐标转换示意图

如图 11-1 所示为坐标转换示意图，图中的红色线条为实际的左转工况下的参考轨迹。由于实际的参考轨迹不满足横坐标递增的需求，需将原始的坐标点由 X-O-Y 的全局坐标系处理到 A-O-B 坐标系下，再将处理后的坐标点作为实际运算过程中的参考轨迹点。这里做第一次坐标变换。

如图 11-2 所示为转换到自车坐标系下示意图，图中蓝色的线条表示自车的位置，b-a-c 表示当前的自车坐标系。在横向控制中，需要计算横向位置偏差和横摆角偏差。首先在 A-O-B 坐标系下计算出预测时域的参考横坐标和参考纵坐标，在这一步计算中，先计算得到参考横坐标，通过二插法得到参考纵坐标。然后将得到的参考横坐标和参考纵坐标转换到自车坐标系下，在自车坐标系下，自车的横向位置坐标为 0，自车此刻的横向参考坐标就是横向偏差值。同理自车此刻的横摆角为 0，横摆角偏差就是此时的横摆角参考值。

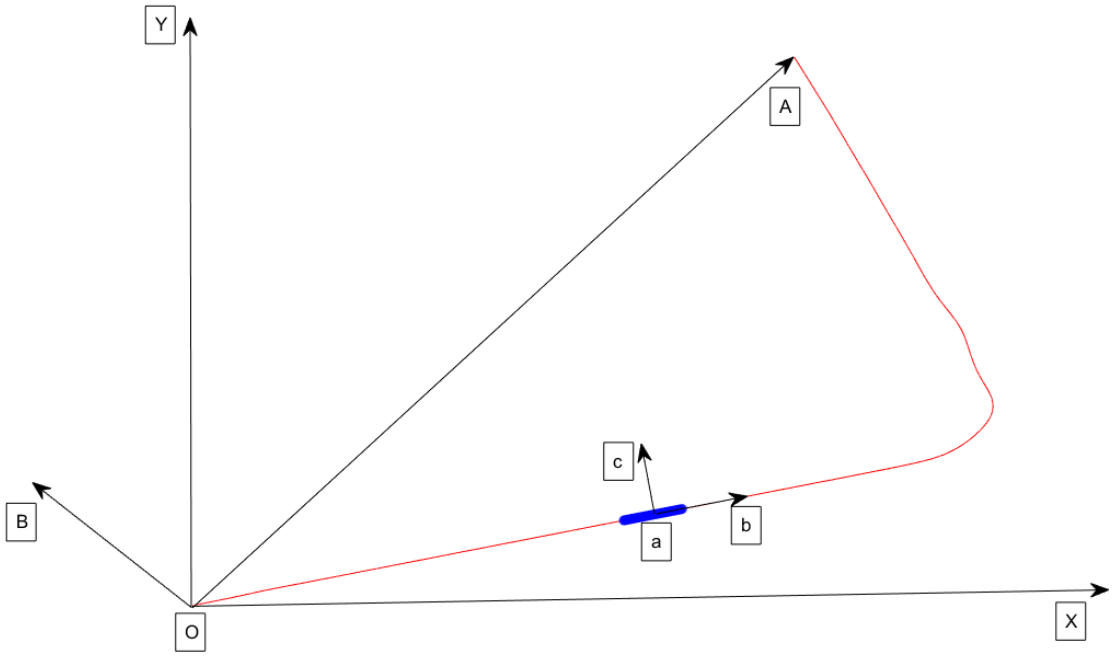


图 11-2.转换到自车坐标系下示意图

对于二插法和坐标转换的 C 代码实现，在控制算法中都有包含，实现的形式也比较简单。同时实现插值功能和坐标转换的方式也有多种，此处就不附录实现源码。如还有问题可以进一步讨论

12单元测试和仿真效果

本节主要展示在 PC 端通过 Simulink 和 Carsim 联合仿真得到的控制效果。根据项目的验收指标，左转和右转工况分别测试了 20km/h 恒速工况、直路 40km/h 转弯 20km/h 工况、直路 60km/h 转弯 20km/h 工况和弯道部分变速工况。记录仿真视频，同时将仿真得到的输出数据记录在 excel 中。下图为部分通过仿真数据拟合得到的对比图。

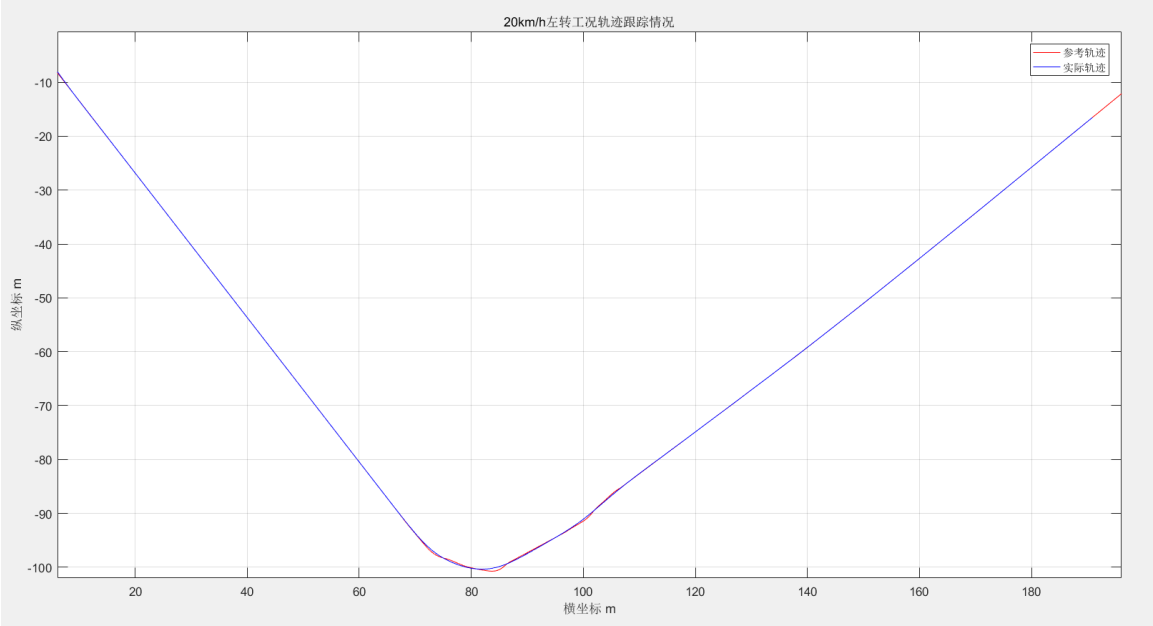


图 12-1. 20km/h 恒速左转工况轨迹跟踪情况

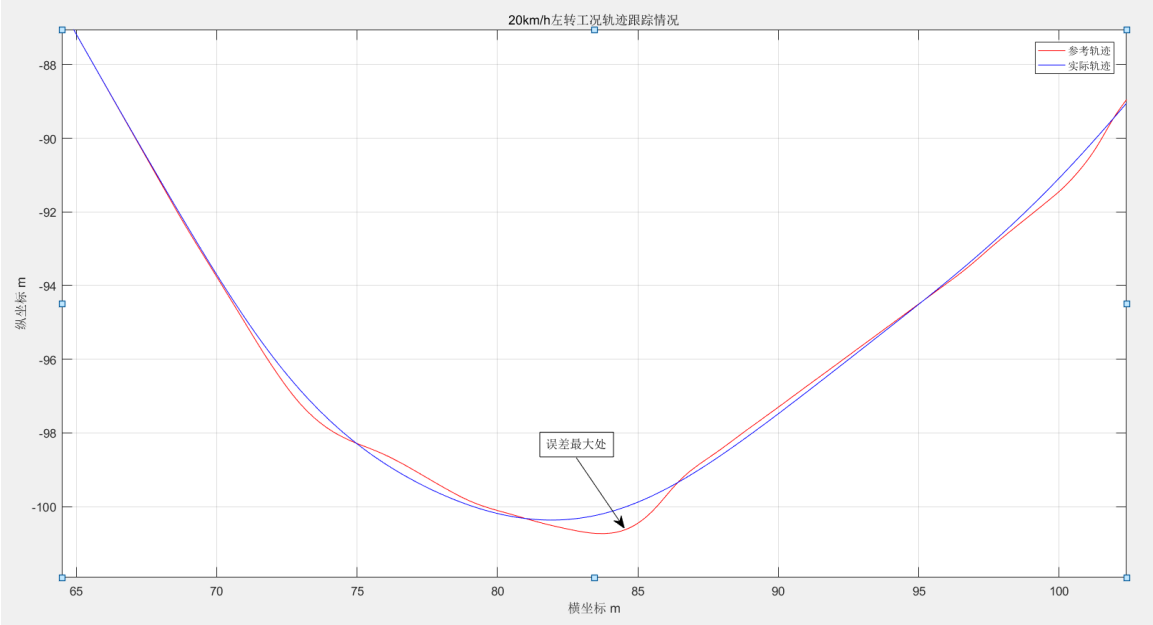


图 12-2. 20km/h 恒速左转工况轨迹跟踪情况放大图

如图 12-1 所示为左转工况，定速 20km/h 的工况下，轨迹跟踪拟合情况。从整体看，直路段部分的期望轨迹和实际轨迹基本吻合，在转弯部分存在跟踪误差。由图 12-2 中可以看

到，参考轨迹在弯路部分并不规则，实际轨迹更平滑一些，牺牲了部分跟踪性能但是使得方向盘转角的变化较小，同时也可以表明算法具有较好的鲁棒性。如果参考轨迹中存在坏值，某路段存在异常也可以较好的实现轨迹跟踪。

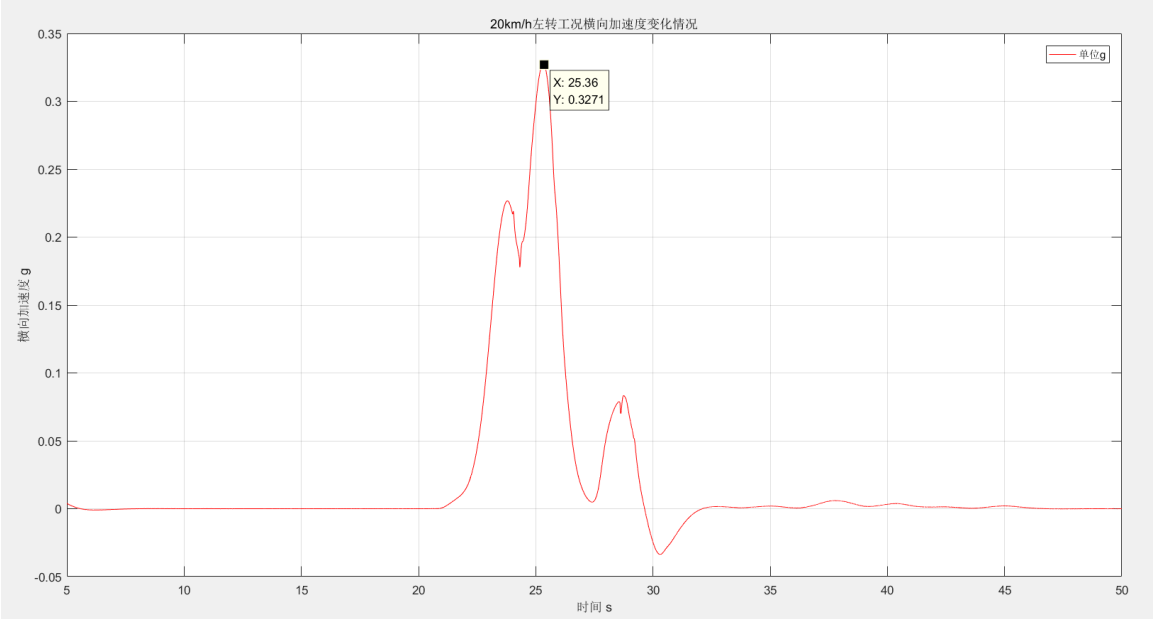


图 12-3. 20km/h 恒速左转工况横向加速度变化

如图 12-3 所示为 20km/h 恒速左转工况仿真时的横向加速度变化情况。图中横向加速度单位为g，峰值加速度达到了 $3.21m/s^2$。

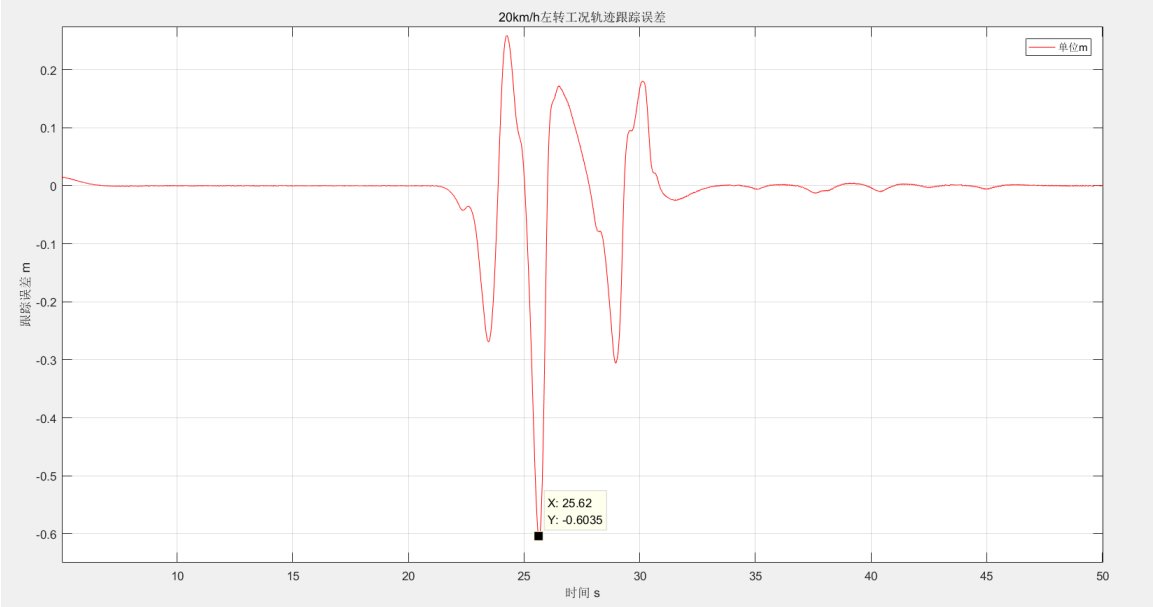


图 12-4.20km/h 恒速左转工况轨迹跟踪误差

如图 12-4 为 20km/h 恒速左转工况的轨迹跟踪误差，在平直路段的跟踪误差基本趋于 0，在转弯部分存在较大的跟踪误差。转弯处的最大跟踪误差约为 0.6m。

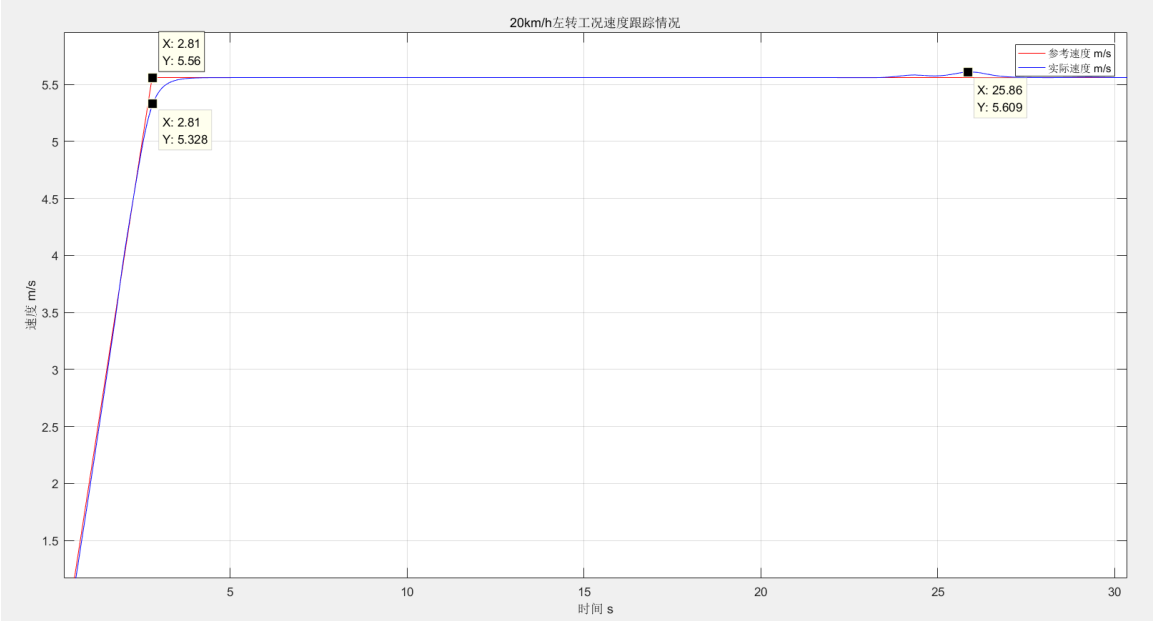


图 12-5. 20km/h 恒速左转工况速度跟踪情况

如图 12-5 所示为 20km/h 恒速左转工况速度跟踪情况。在加速结束部分和弯道处存在速度跟踪的偏差，直路工况实际速度与期望速度基本相同。从图中可以发现在加速结束部分的速度误差要比弯道处的速度误差更大。这是由于 MPC 具有预测特性，在接近期望速度时加速度缓慢减小，加速结束部分跟踪表现更加平滑。

下面再放置一组直路段 40km/h 弯道 20km/h 左转工况的仿真图例，原因与上述相同不做过多说明。

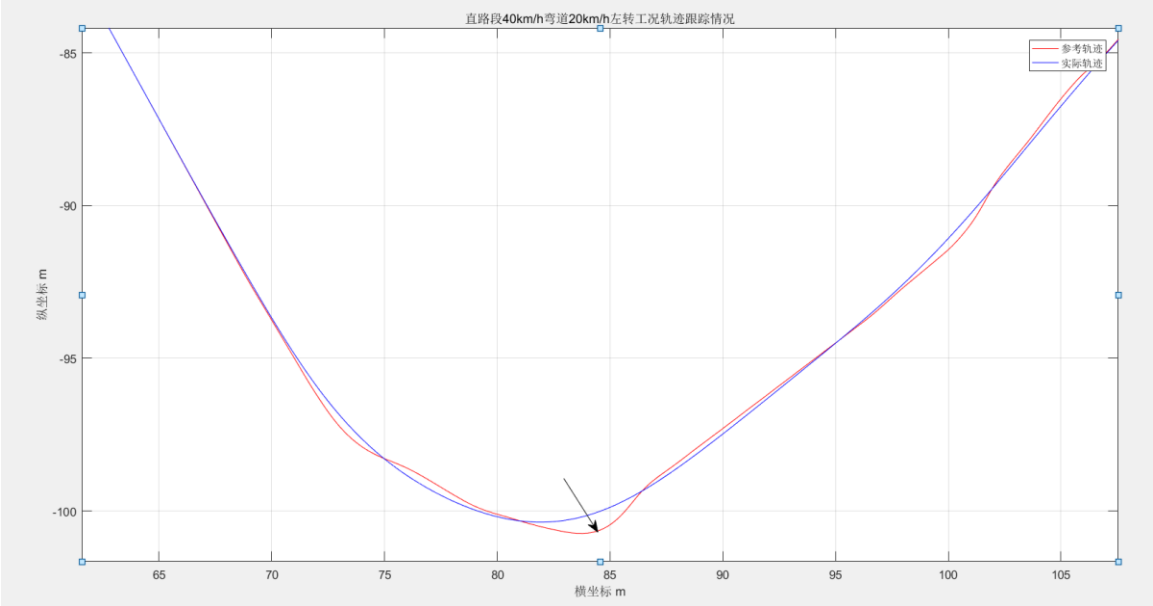


图 12-6.直路段 40km/h 弯道 20km/h 左转工况轨迹跟踪情况

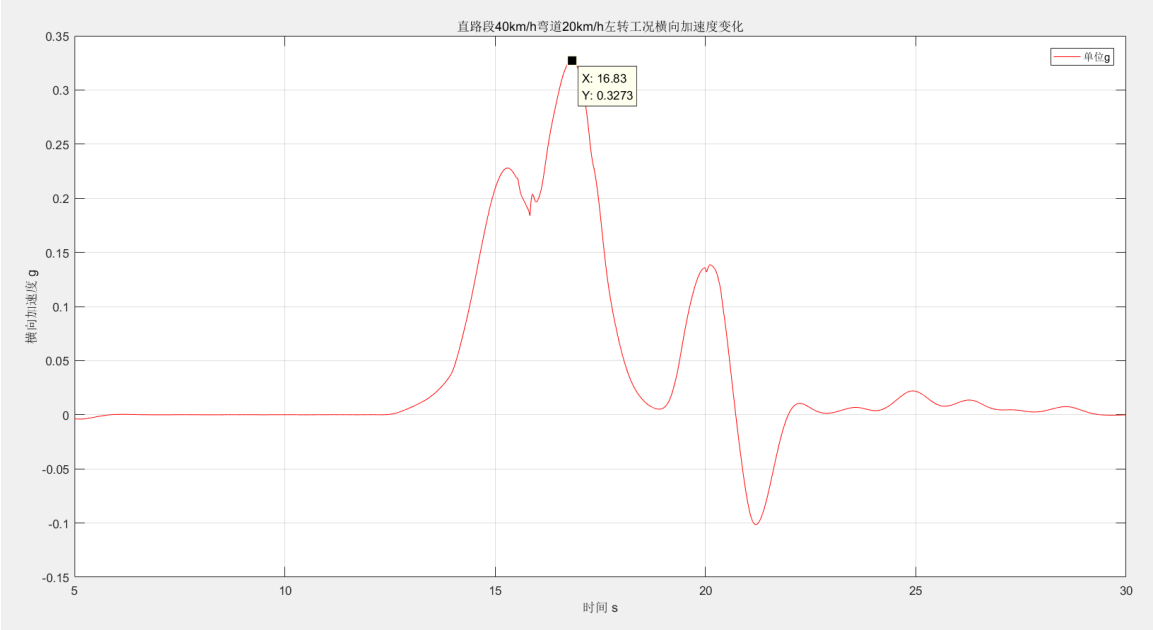


图 12-7.直路段 40km/h 弯道 20km/h 左转工况横向加速度变化

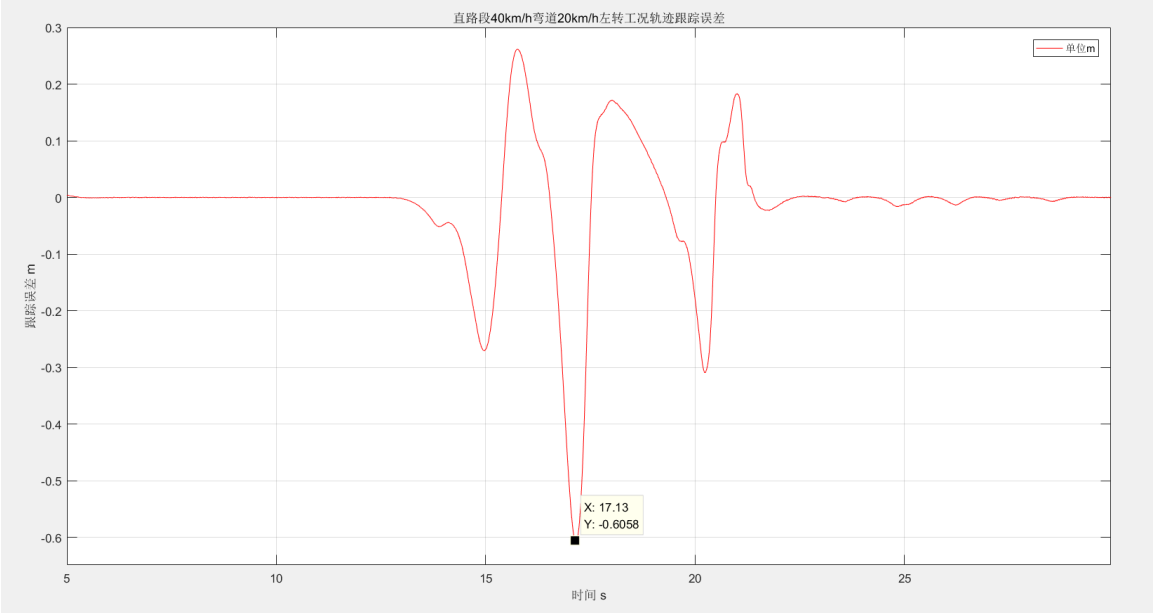


图 12-8.直路段 40km/h 弯道 20km/h 左转工况轨迹跟踪误差

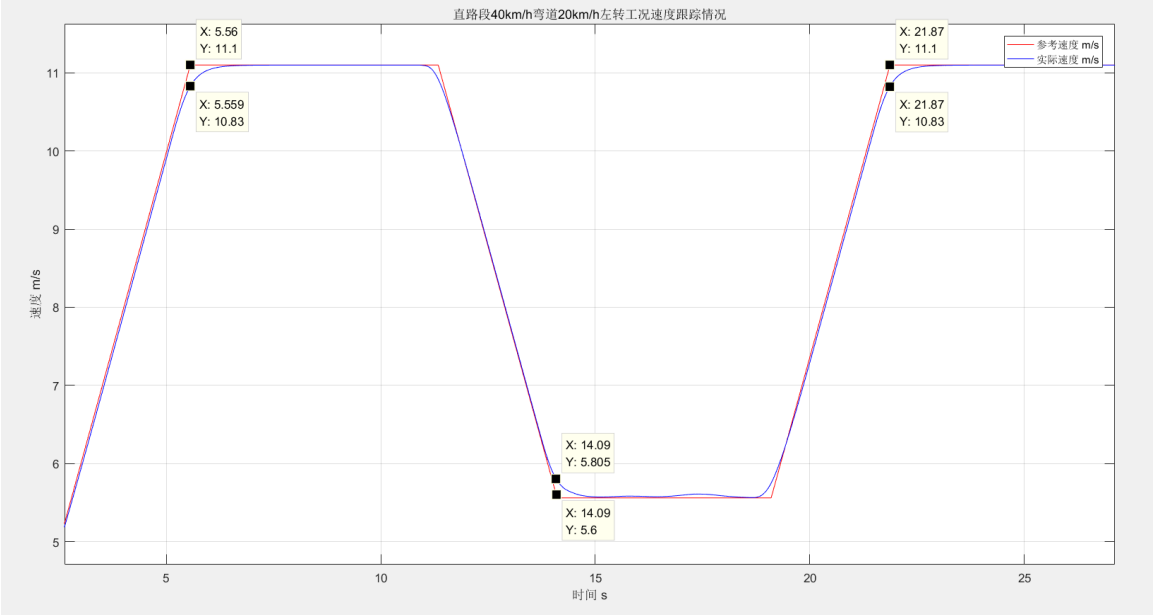


图 12-9.直路段 40km/h 弯道 20km/h 左转工况速度跟踪情况

利用 Carsim 和 Simulink 联合仿真对上面提到的多种工况进行了仿真，都实现了轨迹跟踪功能，本章节之前的部分展示了 2 组利用仿真数据，用 Matlab 作图的对比效果。表格 12-1 中展示了不同工况下的轨迹跟踪误差和速度跟踪最大误差。

表格 12-1.不同工况的跟踪误差

工况		轨迹跟踪弯道最大误差	轨迹跟踪直路最大误差	速度跟踪最大误差
左转	20km/h 恒速	0.6m	<0.1m	0.238m/s
	直路 40km/h，转弯 20km/h	0.6m	<0.1m	0.27m/s
	直路 60km/h，转弯 20km/h	0.583m	<0.1m	0.4m/s
	转弯变速(直路 11.1m/s，弯路 5.56-9m/s)	0.69m	<0.1m	0.39m/s
右转	20km/h 恒速	1.158m	<0.1m	0.267m/s
	直路 40km/h，转弯 20km/h	1.16m	<0.1m	0.28m/s
	直路 60km/h，转弯 20km/h	1.153m	<0.1m	0.29m/s
	转弯变速(直路 16m/s，弯路 5-8m/s)	1.197m	<0.1m	0.303m/s

表格 12-1 中，转弯工况的弯道最大误差均产生于横向加速度最大的部分。弯路段 20k/h 恒速工况下，对于左转工况，轨迹跟踪的横向最大误差为 0.6m，此时横向加速度的峰值为 $3.14\mathrm{m}/s^2$；对于右转工况，轨迹跟踪的横向最大误差为 1.16m，此时横向加速度的峰值为 $3.861\ \mathrm{m}/s^2$。弯路段变速工况下，对于左转工况，轨迹跟踪的横向最大误差为 0.69m，此时横向加速度的峰值为 $6.547\ \mathrm{m}/s^2$；对于右转工况，轨迹跟踪的横向最大误差为 1.197m，此时横向加速度的峰值为 $4.457\ \mathrm{m}/s^2$。

13HIL 测试和性能评价

13.1 转弯工况

本节主要对在 HIL 实验中得到的控制效果进行展示，根据项目的验收指标，左转和右转工况分别测试了 20km/h 恒速工况、直路 40km/h 转弯 20km/h 工况、直路 60km/h 转弯 20km/h 工况和弯道部分变速工况。将仿真得到的输出数据记录在 excel 中，仿真得到的部分视频也做出记录，下面仅展示部分由仿真数据拟合得到的对比图。

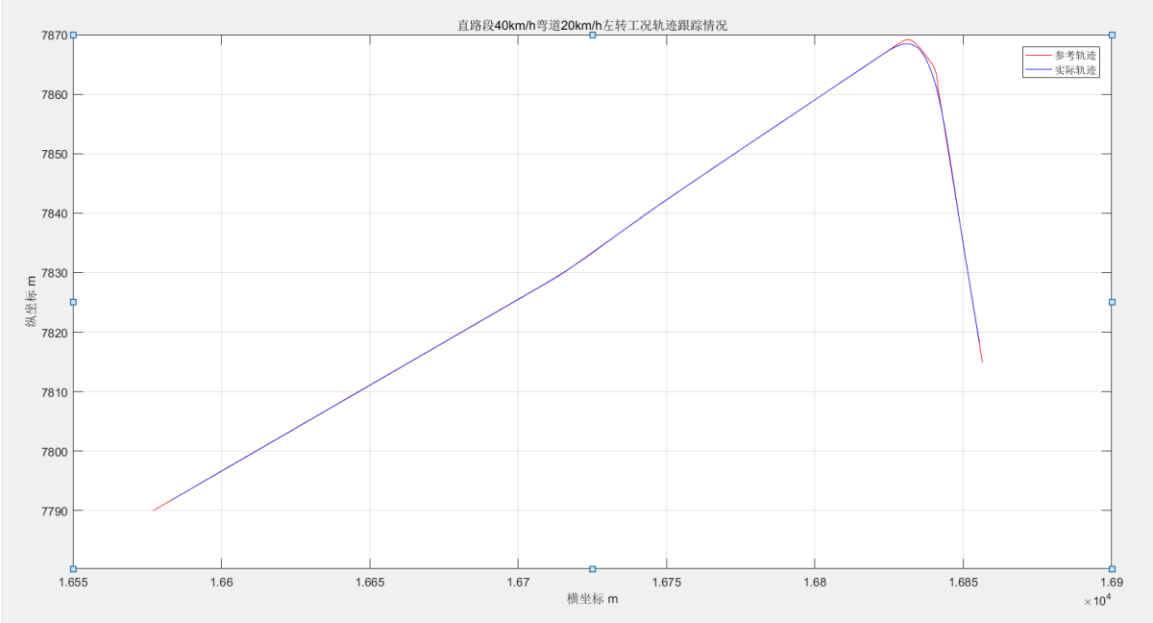


图 13-1.弯道变速右转工况轨迹跟踪整体图

如图 13-1 所示为弯道变速右转工况下的轨迹跟踪情况，从图中可以看出在平直路段的跟踪效果很好，与期望的轨迹基本重合，在弯道处存在跟踪误差。

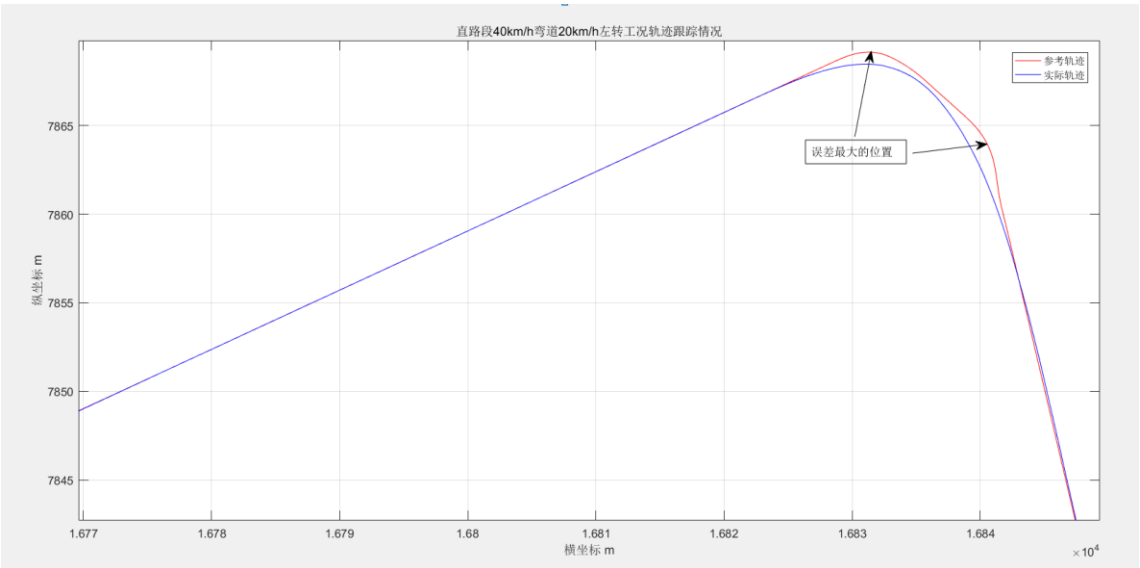


图 13-2.弯道变速右转工况轨迹跟踪弯道放大图

如图 13-2 所示为弯道处的放大图，从图中可以看出，和在 PC 端仿真类似，实际的轨迹比期望的轨迹更为平滑。在弯道处采取了提前转弯，避免了完整跟踪带来的方向盘变化过快导致车辆运动不稳定甚至跑飞的情况。在过弯后的很快的进入稳定跟踪状态，达到了很好的轨迹跟踪效果。

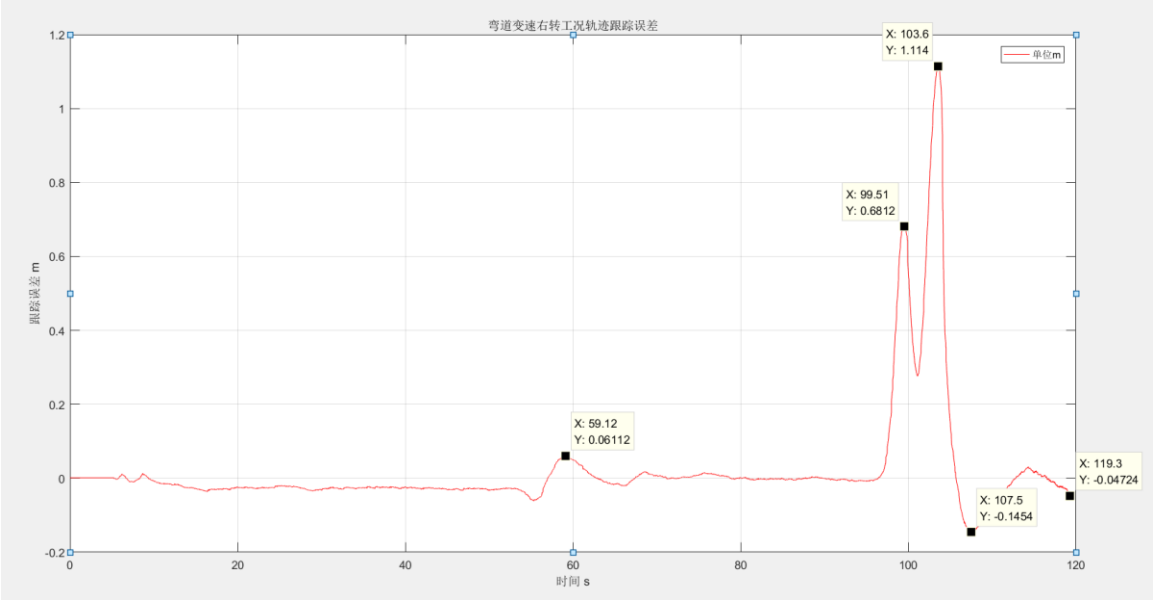


图 13-3.弯道变速右转工况轨迹跟踪误差

如图 13-3 所示为弯道变速右转工况的轨迹跟踪误差，在这里可以看到，平直路段的轨迹跟踪误差基本趋近于 0，在道路曲率较大的位置最大的偏差为 0.061m。在弯道处的偏差最大达到了 1.114m，与 PC 仿真的误差相近。由于硬件在环实验没有输出横向加速度，这里类比 PC 端仿真结果，在转弯处的横向加速度较大，在其他位置的跟踪误差均保持在 0.1m 以内，满足验收标准。

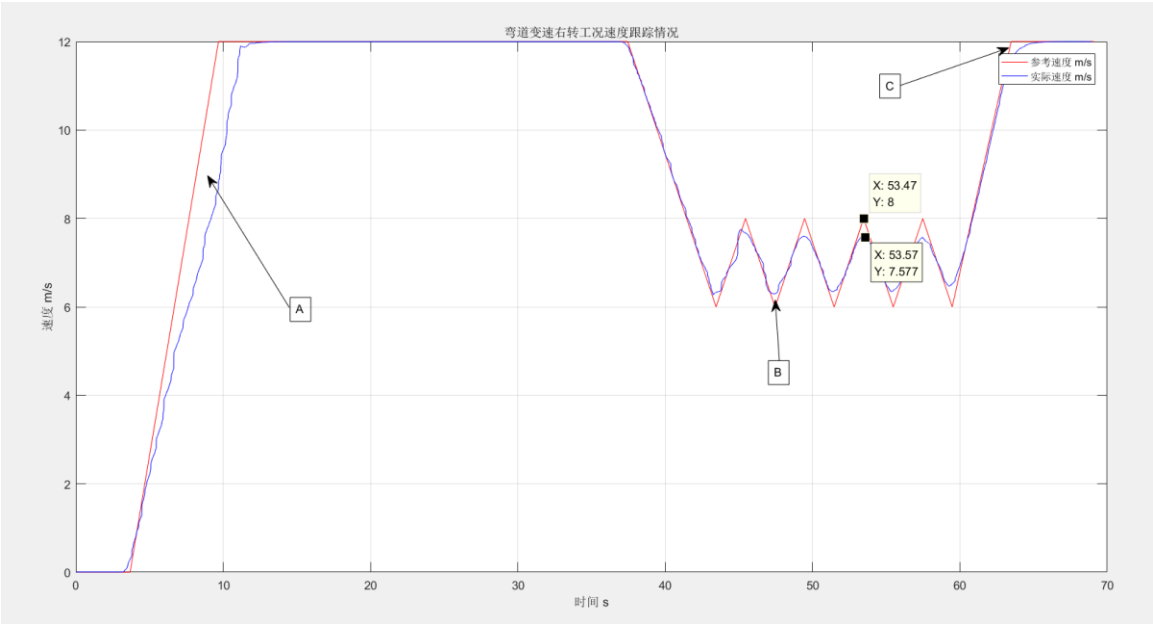


图 13-4.弯道变速右转工况速度跟踪对比图

如图 13-4 所示为弯道变速右转工况速度跟踪对比图，在图中标记了 A、B、C 共 3 处速度偏差大的部分。从图中可以看出，B、C 处的跟踪误差是由于 MPC 算法的控制特点引起的，会牺牲一部分跟踪性能，具有“预测能力”，避免了强加速强制动的情况。从图 13-4 中可以看到在 B、C 处有较好的跟踪性能但是在起步的部分 A 处存在较大的跟踪误差，怀疑误差的原因不是来自控制算法，可能存在执行机构延时等原因，对这一现象进行了测试。

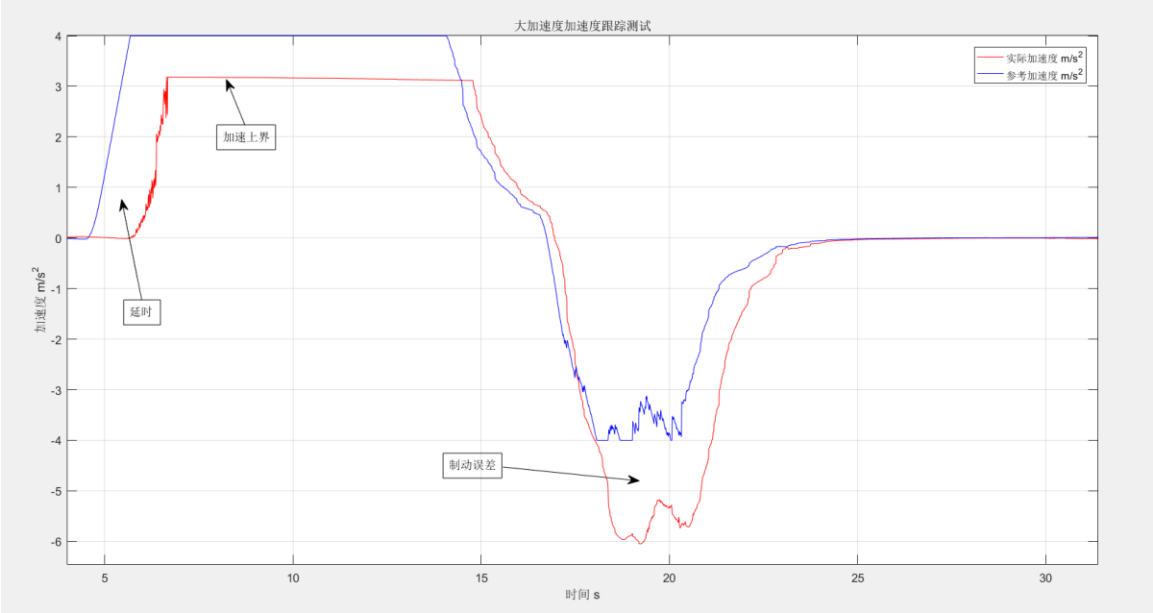


图 13-5.大加速度下的加速度对比图

如图 13-5 所示为大家速度下的加速度对比图，采取的工况为左转工况，期望的加减速度均为 $2m/s^2$。在加速初期，由于速度偏差大，算法计算得到的加速度达到了 4 m/s^2（这里对期望加速度做了约束，根据正常行驶，对加速度设定峰值为 4 m/s^2），而从图 13-5 中可以看到，实际的加速度存在较长时间的延迟，以 10ms 计算一次，延迟大约有 1-2s。同时发现在减速部分的延迟时间相较于启动部分减少了很多，这与图 13-4 中的表现是一致的，即在启动部分的跟踪误差很大，但是在后续的控制中得到了较好的控制效果，论证了之前的推论，误差的来源并不来源于纵向控制算法。同时从图 13-5 中也发现，在扭矩峰值在$300N/m$的情况下，实际的加速度仅能达到 3.18 m/s^2左右，在轨迹跟踪时，如果期望前期有较好的速度跟踪效果，需要采取缓加速的策略，且加速度不能超过实际限额。从图中还可以发现，在制动部分不仅存在延迟还存在减速度偏差，说明制动环节存在一定的模型失配。为了对比验证，在缓加速度工况下可以得到较好的速度跟踪效果，对缓加速工况下的速度跟踪测试和加速度跟踪测试。

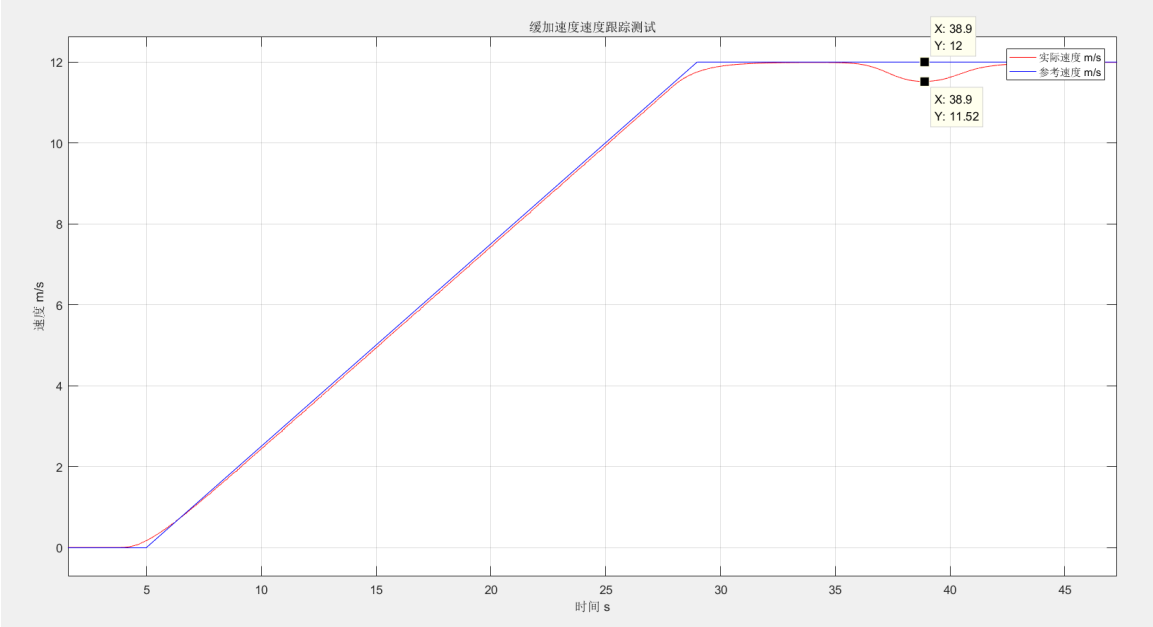


图 13-6.缓加速度速度对比图

如图 13-6 为缓加速度跟踪测试时的速度跟踪情况，从图中可以看到，期望速度和实际速度基本吻合，在弯道处存在较大的速度跟踪误差为 0.48m/s。此时为左转工况，转弯处的速度为 43.2km/h。

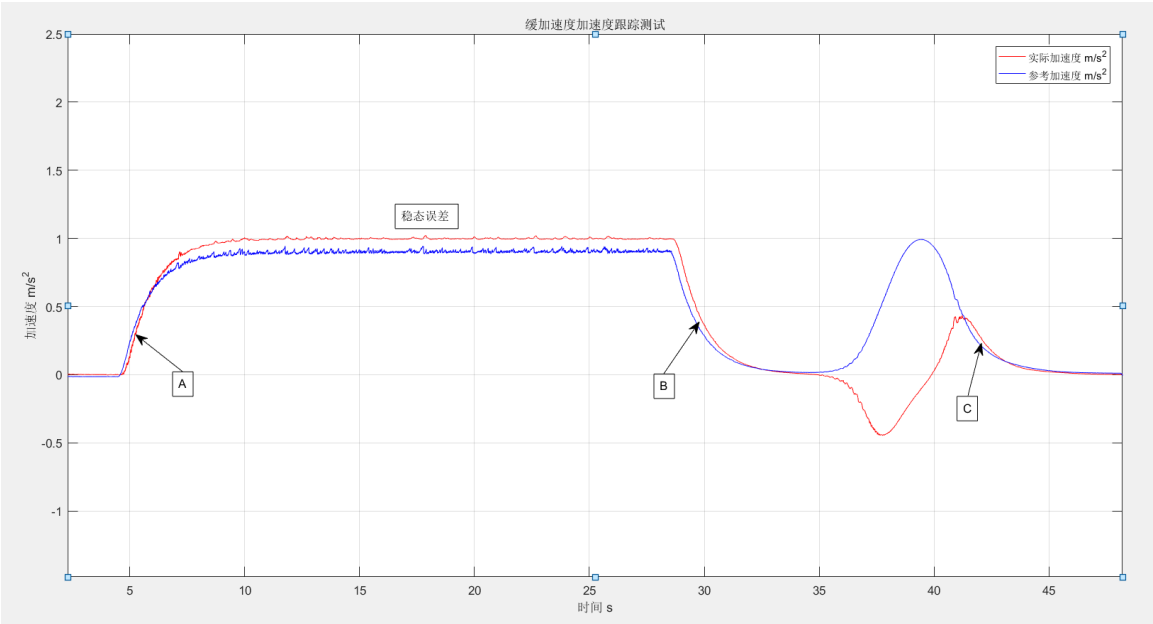


图 13-7.缓加速度下的加速度对比图

如图 13-7 所示为缓加速工况下的加速度对比情况。从 A、B、C 处可以看到在这次测试时的实际加速度延迟较图 13-5 中的起步延时减少了很多，同时可以注意到 A、B 之间的稳态部分存在稳态误差。说明在加速部分存在少量的模型失配，对比图 13-5 中的减速部分，

减速部分的模型失配较大。同时比对图 13-6 和图 13-7 可以论证，即使存在一定的模型失配现象，MPC 算法也可以得到很好的控制效果。

随后对 UC-WinRoad 环境进行了优化处理，解决了之前存在的卡顿问题，在可视化环境优化后，对存在的延时问题再次进行了测试。

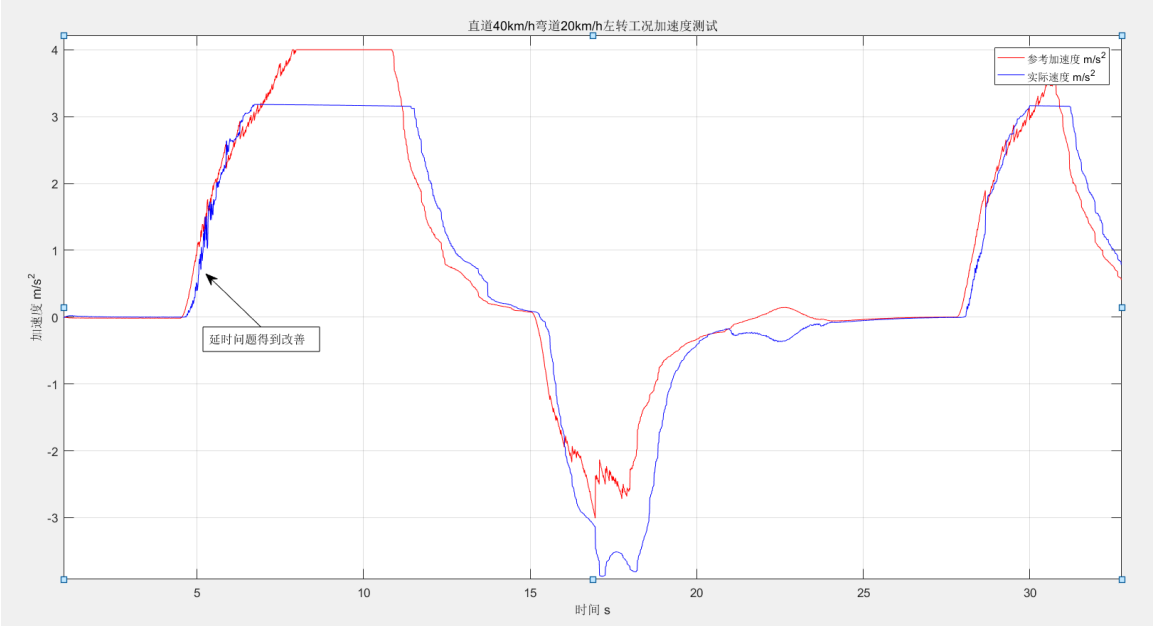


图 13-8.直路 40km/h 转弯 20k/h 左转工况加速度测试图

如图 13-8 为测试结果效果图，对比图 13-5 发现在启动部分的延时问题基本解决，实际的加速度随着期望加速度的增加而增大，同时注意到，Carsim 模型的加速能力是确定的和环境无关不受 UC-WinRoad 环境影响。

下面再展示一组右转工况下的 HIL 测试结果：

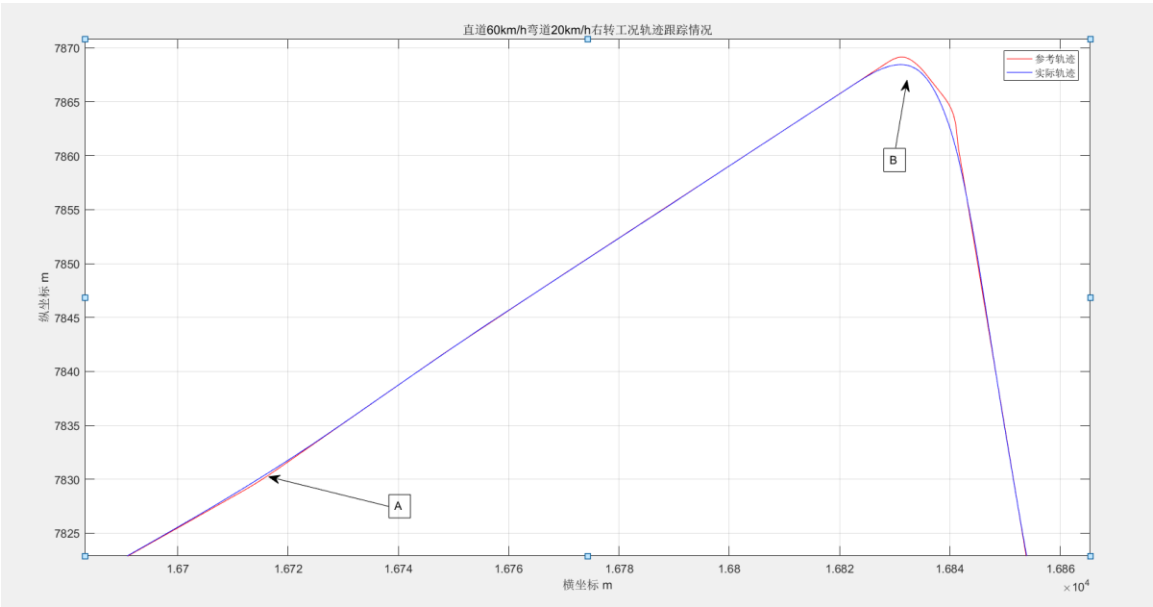


图 13-9.直路 60km/h 转弯 20k/h 右转工况轨迹跟踪对比图

如图 13-9 所示为直路 60km/h 转弯 20k/h 右转工况轨迹跟踪对比图，从图中可以看到，相比图 13-1 大曲率处的跟踪误差更为明显。

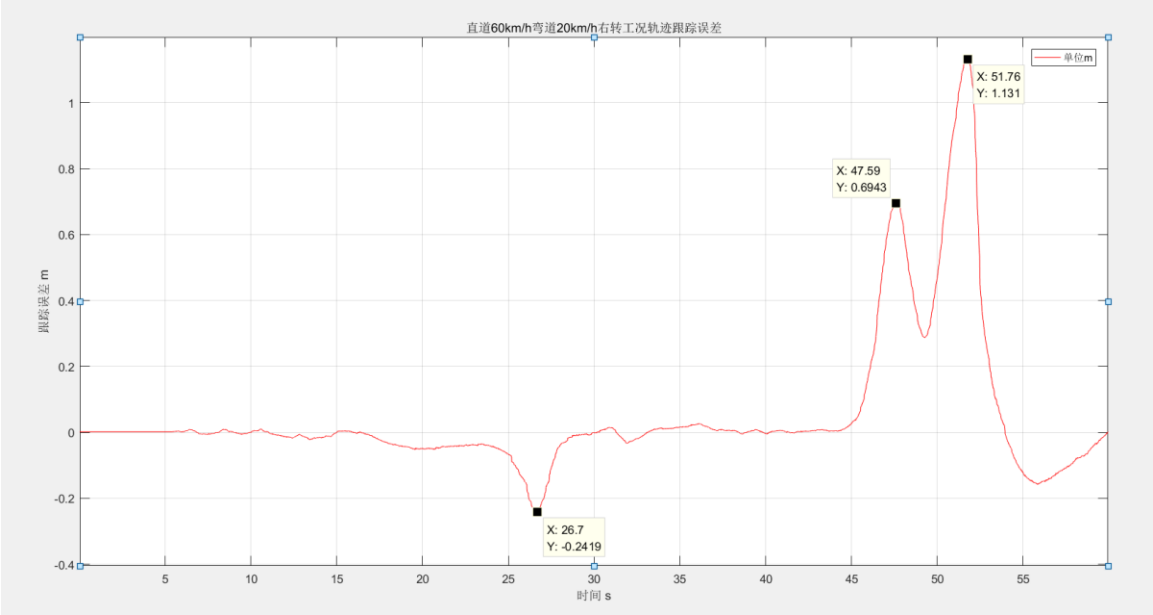


图 13-10.直路 60km/h 转弯 20k/h 右转工况轨迹跟踪误差

从图 13-10 可以看到在大曲率道路部分的跟踪误差已经达到 0.24m，说明在高速工况下，即使曲率很大，较小的方向盘转角控制量就会引起较大的跟踪误差。同时也可以论证，在速度较大时，控制算法中的权重系数要调小一些。

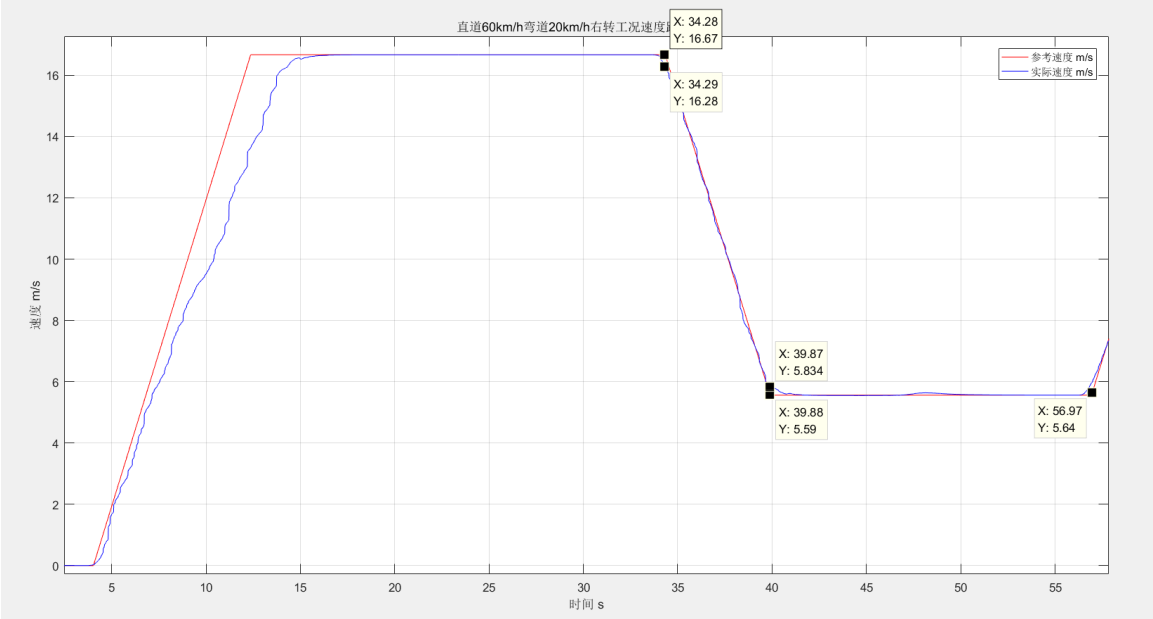


图 13-11.直路 60km/h 转弯 20k/h 右转工况速度跟踪对比图

如图 13-11 所示，速度跟踪情况与图 13-4 类似，均在起步阶段存在较大误差，之后部分得到较好的跟踪效果，最大速度偏差大约为 0.39m/s。

以上展示两组 HIL 实验绘制的对比图例，其他测试工况的对比图类似，表格 13-1 中呈现了不同工况下的轨迹跟踪误差和速度跟踪误差。

表格 13-1.不同工况的跟踪误差

工况		轨迹跟踪弯道误差	轨迹跟踪直路误差	速度跟踪最大误差
左转	20km/h 恒速	0.998m	<0.1m	0.127m/s
	直路 40km/h，转弯 20km/h	0.606m	<0.1m	0.95m/s
	直路 60km/h，转弯 20km/h	0.593m	0.111m	0.927m/s
	转弯变速(直路 12m/s，弯路 6-10m/s)	0.434m	<0.1m	0.95m/s
右转	20km/h 恒速	1.114m	<0.1m	0.3m/s
	直路 40km/h，转弯 20km/h	1.12m	<0.1m	0.59m/s
	直路 60km/h，转弯 20km/h	1.13m	<0.1m	0.39m/s
	转弯变速(直路 12m/s，弯路 6-8m/s)	1.114m	<0.1m	0.42m/s

从表格 13-1 中可以发现，在 HIL 硬件在环实验中速度跟踪误差较大，需要对前述分析的延时问题进行优化，同时对减速部分的较大的模型失配问题也进行一定的优化。对于轨迹跟踪误差与 PC 端的误差基本相近，在左转 20km/h 恒速的工况，存在单次测试误差。弯路左转恒速工况，其他两组的轨迹跟踪误差都在 0.6m 附近与 PC 端的偏差结果相近。

13.2 跟车工况

这部分主要展示在 HIL 实验中的控制效果，根据项目的验收指标，在直路跟车工况测试了前车匀速 5km/h、前车匀速 60km/h、前车匀速 80km/h、前车匀速 120km/h 和前车变速工况。前车变速工况的速度变化范围在 55km/h 到 65km/h，前车加速度范围超过了$\pm 1\mathrm{m}/s^2$范围。下面展示两组前车匀速工况下的跟车误差和一组前车变速工况下的跟车误差。

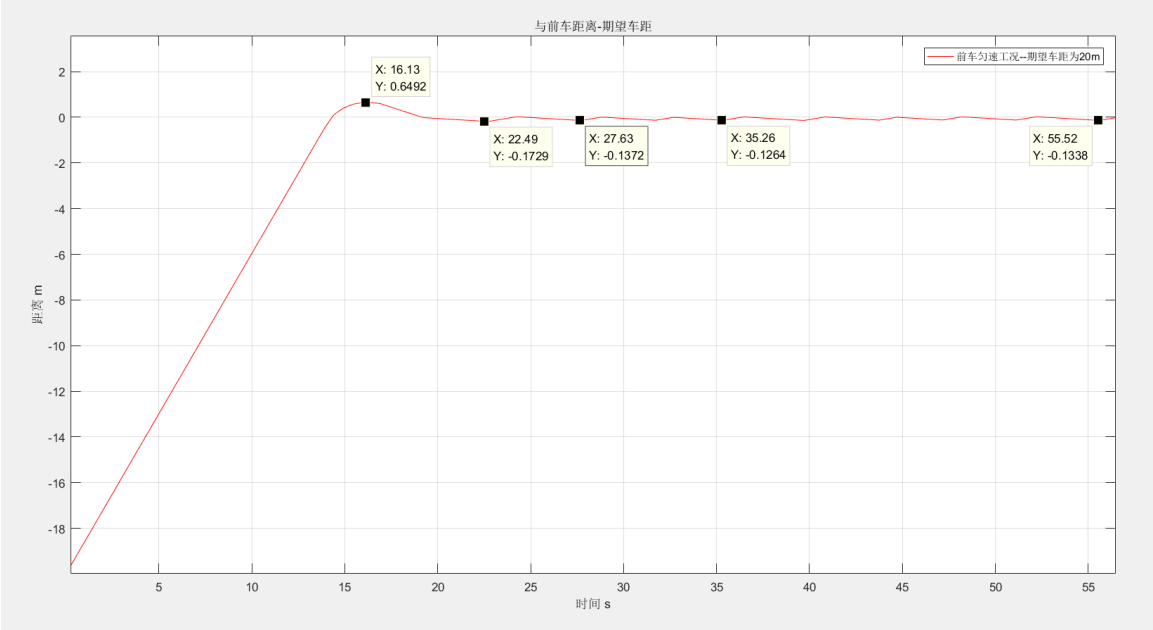


图 13-12.前车匀速 5km/h 工况的跟车误差

图 13-12 所示为前车匀速 5km/h 前进工况下的跟车误差。其中纵轴数值代表前后车距-期望车距。图中的前半部分由于前车与自车处于同一出发点，自车处于制动状态，等前车达到期望的距离后缓慢加速并维持和前车相同的车速前进，保持恒定的跟车距离。从图 13-12 中可以看出，很快的进入了稳定状态，在稳态下的前车跟随误差小于 0.2m 满足验收指标。

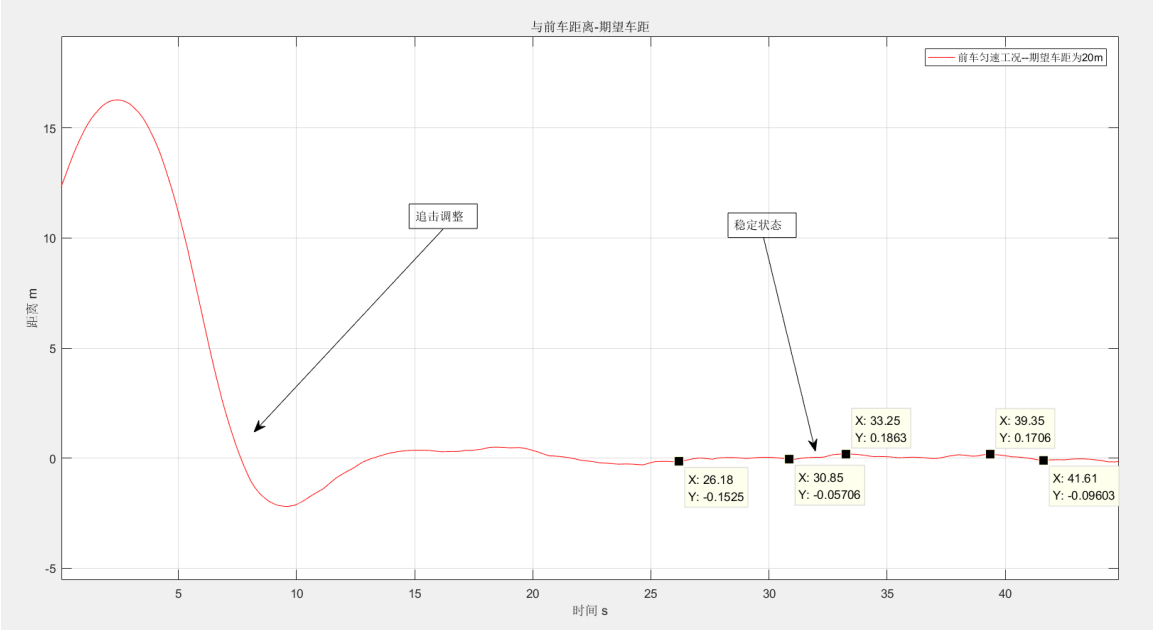


图 13-13.前车匀速 60km/h 工况的跟车误差

如图 13-13 所示为前车匀速 60km/h 工况下的跟车误差。由于其实的前车速度较快，前半部分处于追击和调整阶段，随后进入了稳定跟车状态。从图中的标记点可以看到，在稳定跟车状态的跟车误差小于 0.2m 的误差范围，满足验收指标。

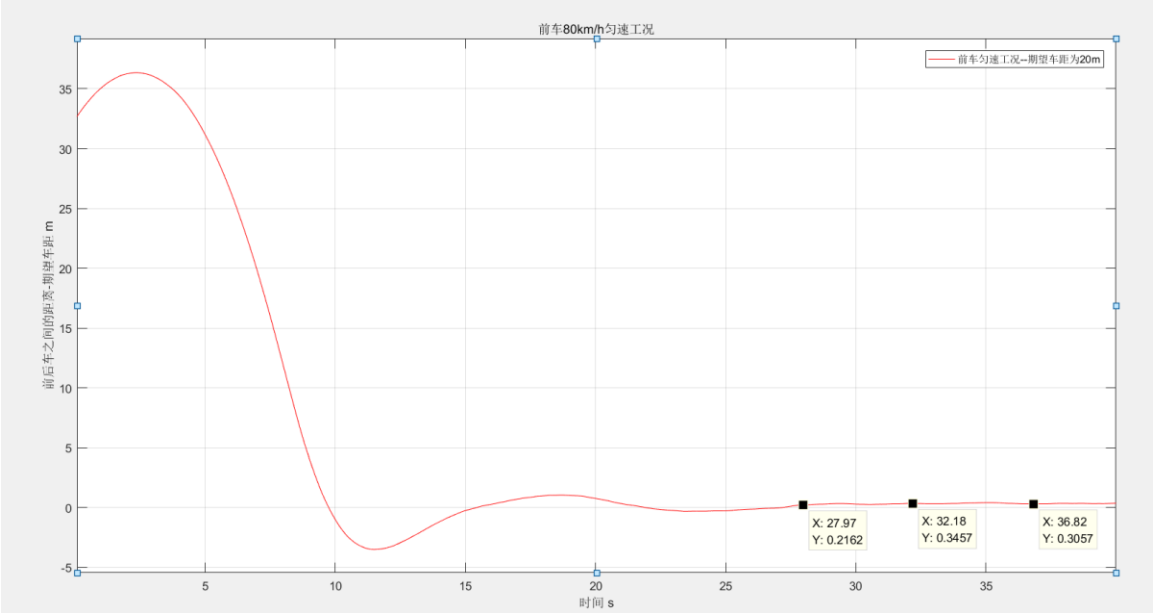


图 13-14.前车匀速 80km/h 工况的跟车误差

如图 13-14 所示为前车匀速 80km/h 工况下的跟车误差，从图中可以看到，在后半部分进入了稳态跟车，误差在 0.3m 左右。分析原因可能由前车的轻微速度波动引起，在高速行驶工况下，小幅的速度波动也会引起一定的跟踪误差。

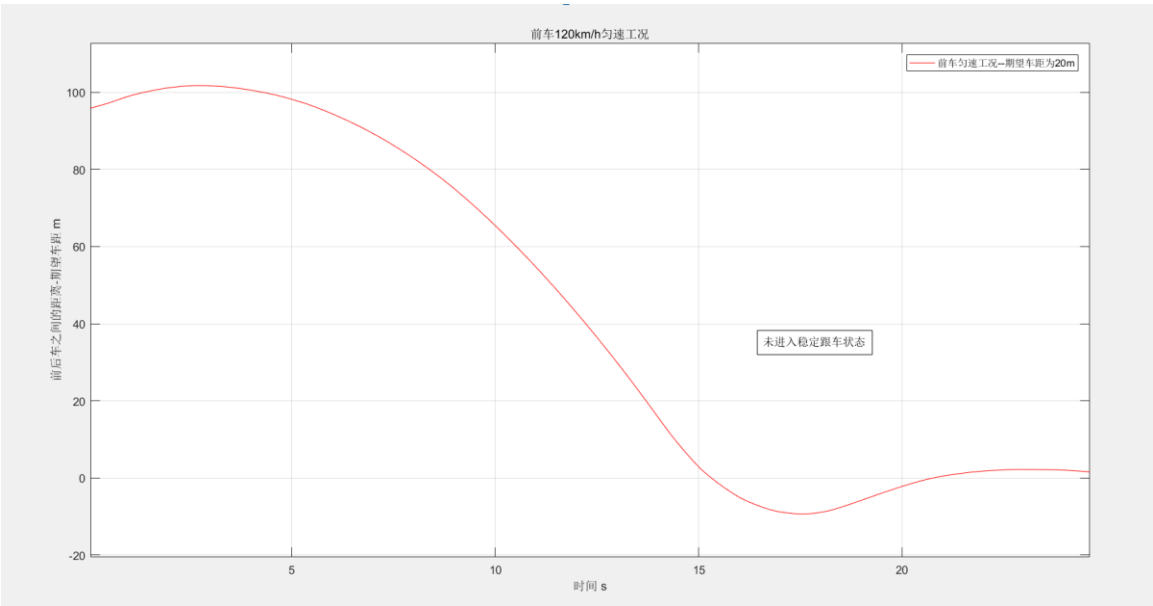


图 13-15.前车匀速 120km/h 工况的跟车误差

如图 13-15 所示为前车匀速 120km/h 工况下的跟车误差，从图中可以看出，由于直路段不够长，还未进入稳定跟车状态，已经跑完了全部路程。

从上述的 4 种工况的跟车行为可以看到，路程足够长，能够进入稳定跟车阶段都可以得到很好的跟车效果。

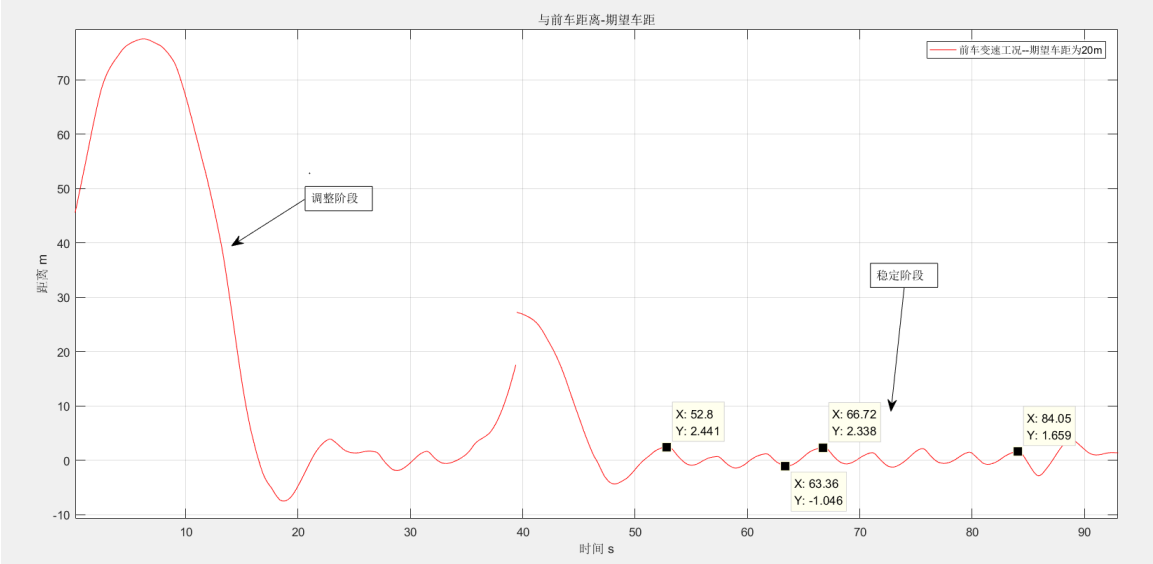


图 13-16.前车变速工况的跟车误差

如图 13-16 所示为前车变速工况下的跟车误差。从图中可以看到，由于前车的车速较快，前半部分处于自车加速追击的阶段，处于动态调整车距的过程。在 40s 的位置出现了数据丢失的问题，在 50s 之后进入了稳态跟车部分，从图中标记的点可以看到在前车速度变化的工况下，自车与前车的车距保持在 3m 以内，满足验收指标。同时需要指出，变速工况下的前车加速度范围超过了验收指标中的$\pm 1 m/s^2$范围。

表格 13-2.不同工况的跟车误差

工况		稳态跟车 最大误差
跟车	前车 5km/h 恒速	0.1729m
	前车 60km/h 恒速	0.1863m
	前车 80km/h 恒速	0.3457m
	前车 120km/h 恒速	直路过短未进入稳态
	前车 55-65km/h 变速	2.441m

表格 13-2 中为不同跟车工况，进入稳态跟车的最大误差情况。从表中可以看到，在前车 5km/h 匀速和 60km/h 匀速的工况，稳态跟车时的最大纵向跟车误差小于 0.2m，满足验收指标。在前车 80km/h 匀速工况，稳态跟车时的最大纵向跟车误差为 0.3457 超过了 0.2m，分析原因，前车存在小幅速度波动，同时可能还处于调整阶段。对于前车 120km/h 的匀速工况，由于前车速度过快，还未调整进入稳态跟车阶段就已经跑完全部路程。对于前车变速工况，稳态跟车的最大纵向跟车误差小于 3m，满足验收指标。

14HIL 调试的经验教训

首先需要强调的是，开发的过程中，需要实现某一功能开发的代码一定要先验证再使用，防止在源头出现错误。接下来对调试过程中容易出现问题的地方进行叙述。

1. 离散化

构建问题时为连续系统，因此在迭代求解优化问题时需首先对连续系统离散化。最初在 PC 端仿真时为单独的横向轨迹跟踪控制，纵向速度为恒定值 20m/s。在 C 代码中采用近似离散对比 Matlab 精确离散时发现误差很小。在横纵向联合控制时，由于速度由 0 开始增加，而作为状态方程系数矩阵中的分母项，速度较小时，采用近似离散会造成较大的误差。对于该问题的较优处理方式为，在纵向速度小于某一定值时取定值高于一定纵向速度后可用实时速度计算系数矩阵。同时可以对近似离散多保留几项，以减少误差。

2. 代码功能验证

无论是自己写的代码还是工具箱生成的代码再或者是从网上下载的代码，在使用前都应通过。即使无需理解代码的内容，也要以黑箱的形式，给定输入求解输出验证代码的正确性。横向控制部分在计算横向偏差时需要使用插值函数，而插值函数分为分段线性插值、牛顿插值、分段三次插值等。首先根据需求选取合适的插值方法，其次完成代码后要先测试代码的正确性在加入到最终的控制代码中。在纵横向控制中都用到了 CVXGEN 求解器在线生成代码，在前述部分也介绍了 CVXGEN 的使用方法。需要注意的时，在第一次使用工具箱生成代码时要与相对应的 Matlab 求解结果比对，在构建代价函数时预测时域部分容易出错。

3. 偏差的计算

以横向控制为例，在计算时需要计算横向偏差和横摆角偏差。这里要注意的横向偏差指的是自车坐标系下的偏差。若全局坐标系与自车坐标系相同且直路工况下计算横向偏差较为容易，若全局坐标系和自车坐标系不同需要采用坐标变换再计算横向偏差。建议每次计算时都以自车作为坐标系原点，并将参考轨迹转换到自车坐标系下进行偏差计算

4. 摩擦圆理论

车辆在运动过程中，包括加速、制动、转向都是靠轮胎的摩擦力来提供的。轮胎的摩擦力是有极限的，不能超过最大静摩擦力，超过最大静摩擦力就会出现打滑失控现象。假设在弯道行驶时，如果一边制动一边转向，车辆同时需要纵向力和横向力，他们为轮胎所能提供的最大静摩擦力的分力。在实际行驶过程中，车辆存在轴荷转移现象，会使轮胎产生不同的形变，再加上轮胎的尺寸、转速、结构不同，轮胎的实际工作状态为半滚半滑。这里可以通过摩擦圆来近似描述调试过程中的这一现象。

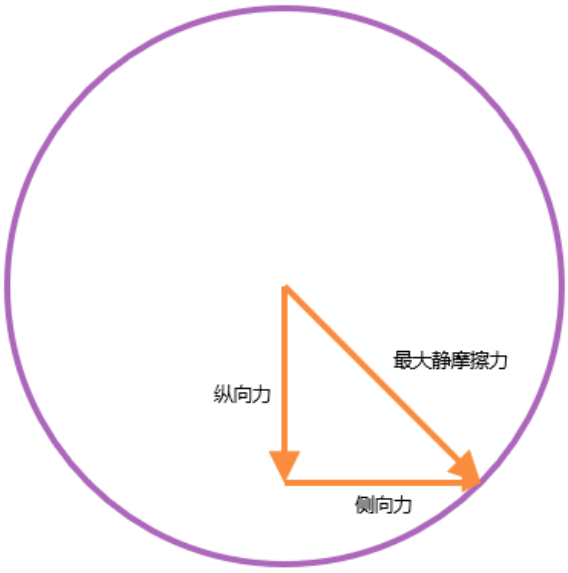


图 14-1.摩擦圆示意图

如图 14-1 所示为摩擦圆示意图，半径表示所能达到的最大静摩擦力。若纵向力和侧向力的合力超过了最大静摩擦力，就会出现轮胎打滑失控的现象。在调试过程中，由于转弯时前轮转角大需要很大的侧向力，如果此时纵向力也很大超出所能提供的极限就会出现车辆失控。合理的驾驶方式即为转弯时速度较慢且没有急剧的速度变化。

5. 模型失配问题

在控制初期要校核模型失配的问题，虽然 MPC 算法具有一定的鲁棒性，但是模型失配会导致控制效果差甚至失控现象。在调试过程中，横向部分的前后轮刚度计算要根据 Carsim 中提供的曲线来测算，由于采用的是二自由度自行车模型，计算得到单个轮胎刚度后需要加倍。目前的调试中，轮胎刚度采用估算的方式，并不为准确值。在控制时，如果不能得到精确的轮胎刚度值，算法中的模型刚度可略大。控制算法将方向盘转角（方向盘转角和前轮转角的比例关系设置为 17.5）传递给实际控制的物理模型，若算法中的刚度值较小，相同的角度，在仿真环境没有失控，但在实际物理模型中由于刚度较大导致真实的侧偏力大于虚拟环境的侧偏力容易产生失控现象。

6. 查找问题

调试过程中需要两方面的能力，第一是查找问题，首先知道问题是什么，第二是解决问题，我个人感觉找到问题的难度更大。下面分享一些关于查找问题的心得体会。在调试过程中的初期，应该是问题最多的时候，问题暴露的也比较明显。其实最好的解决方式应该是从头开始分块验证排查，而不是单个问题解决后就立马进行测试，这样反而容易导致返工。

（1）一定要每一部分都进行验证和排查，问题出现在最多的地方恰恰可能是你认为正确的部分。在最开始调试的过程中，采用了近似离散的方式。由于在 Simulink 和 Carsim 联合仿真的时候没有问题，处理方式也是相同的，就断定离散的方式是没问题的，甚至一度怀疑是使用的 CVXGEN 工具箱出现了问题。经过求解几组 QP 问题，与 Yalmip 工具箱进行比对分析，验证生成代码的正确性后在向上排查发现是离散化部分存在问题。由于纵向速度

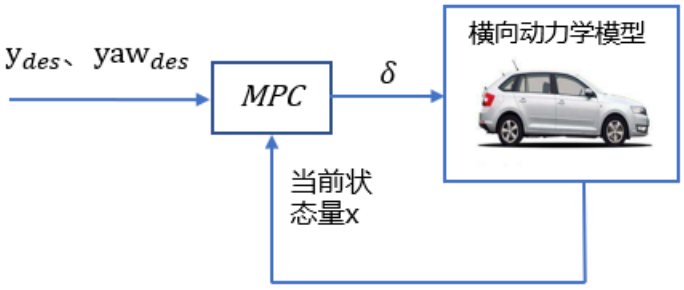
的不同，相同的近似离散导致了离散误差十分大，相同的优化问题求解的结果自然相差很远。

（2）看不出算法的逻辑错误时，采用打印数据的方式，看是哪一部分出现问题。在调试过程中，经常出现找不到问题的现象，尤其是自己写的代码找问题可能更不容易。此时比较好的方式是采用分段打印输出的方式，并将该部分用另一种方式等效复现对比结果。可以认为待测试的部分为黑箱，记录输入输出信息，同时搭建肯定正确的验证模块，将记录的输入输出在验证模块测试，看相同的输入是否得到相同的输出。

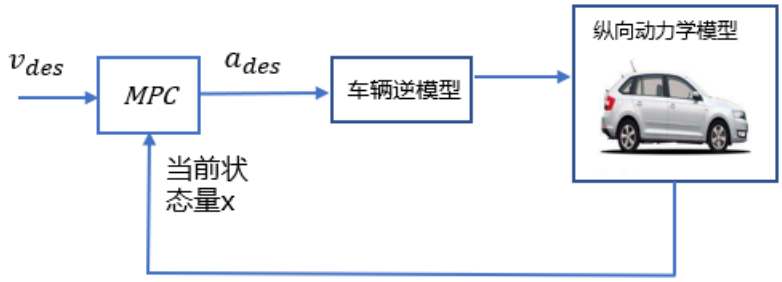
（3）不要陷入算法错误的死循环。在后期调试时，采用了将 C++算法与 Matlab 算法对比的方式，将 Simulink 和 Carsim 的联合仿真的输入输出参数记录，并对比验证了 C++算法的正确性，这时依然出现了 HIL 实验跑飞的现象。这时还认为时算法层面的错误再用更多的数据找问题也是无效的。当验证算法的正确性后，就要考虑是否为仿真环境或者物理模型等方面出现问题。在这次调试中，验证算法无误后，通过师兄的帮助找到问题是超过了轮胎力极限。由于刚度计算不合理，算法中给出的刚度估值较小，同样的方向盘转角导致了物理模型需要更大的侧向力，同时速度曲线不合理，在转弯是速度过大或者速度变化过快导致合力超过最大静摩擦力，必然出现侧滑现象。

（4）先在 PC 端完成全部的仿真测试，再进行硬件在环实验或者其他实验。因为横纵向控制器采用了分别设计的方式，在调试初期只完成了横向部分的 Carsim 和 Simulink 的联合仿真，而且只跑了双移线恒速的工况，纵向部分的完全由 Simulink 搭建，没有和 Carsim 联合仿真，PC 端的仿真还没有完整的验证就进行硬件在环实验导致到处都是问题。这次调试之前，在 PC 端完成了横纵向联合仿真，用 PC 的仿真结果来指导硬件在环调试，对比调试进度会加速很多。

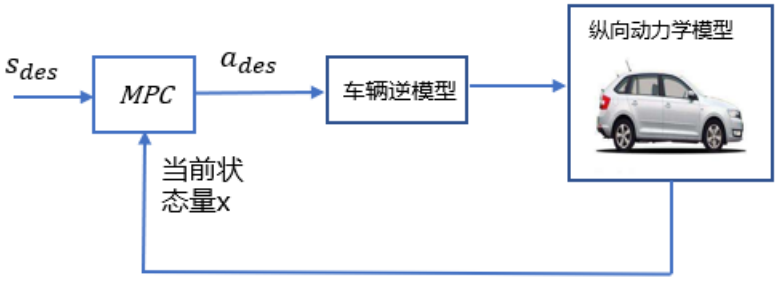
（5）每一个没想通的细节都不放过。在理论研究时总是有很多理想假设、理想工况排除了很多实际因素。在使用二自由度模型时，由于全局坐标系和车辆坐标系是相同的，没有深究横向偏差的选用是全局坐标还是车辆坐标。在二自由度的动力学模型推导过程中采用的是车辆坐标系，在实际控制的时候，计算横向偏差等都需要转换到车辆坐标系下进行。



横向 MPC



跟速度的纵向控制



跟车距的纵向控制