车辆纵向动力学建模与多控制算法性能分析

錦織唯[1]

[1]GitHub: Nishikori Yui

2025 年 5 月 12 日

摘要

车辆纵向控制性能直接关系到自动驾驶系统的行驶安全性、乘坐舒适性及能源利用效率。针对车辆纵向动力学中的非线性空气阻力、滚动阻力、坡度阻力、二阶执行器滞后及防抱死制动（ABS）系统，本文首先构建了一套完整的纵向动力学仿真平台。基于该平台，设计并实现了三种控制策略：自适应 PID 控制、自适应增广 LQI 控制与多目标 MPC 控制。在 10 m/s（市区低速）、20 m/s（郊区中速）和 30 m/s（高速公路）三种典型速度工况下，采用均方误差、最大超调量和能量消耗三项指标，对三种控制器进行了定量仿真对比。

仿真结果表明，自适应 PID 控制器在低速工况下响应快速，但随速度升高产生显著超调及振荡；增广 LQI 控制器可保证零稳态误差且具有较强鲁棒性，但在输入饱和约束下出现深度下冲与迟缓恢复，且能量消耗最大；多目标 MPC 控制器通过考虑系统约束与未来轨迹预测，在所有工况下均实现了最低 MSE、极小超调和最优能耗，体现了最优的多目标折中效果。

此外，本文还讨论了将横向 Pure Pursuit 算法与纵向控制的一体化框架，为纵横向联合优化与路径跟踪协同行驶提供了理论依据。研究结果为自动驾驶车辆在不同工况下纵向控制策略的选择与优化设计提供了系统参考，并为构建全方位、一体化控制架构奠定了基础。

关键词：自动驾驶；车辆纵向控制；自适应 PID；增广 LQI；多目标 MPC

目录

1 引言

自动驾驶车辆的运动控制层需要对纵向动力学进行精准调节，以确保行车安全、乘坐舒适和能源高效利用。纵向控制的核心任务在于根据期望的速度或加速度指令，在复杂多变的工况下实时协调驱动与制动扭矩输出，使车辆实际速度平稳跟踪参考值，并尽可能降低能耗和机械磨损。然而，在典型道路环境中，纵向控制面临诸多技术挑战。主要挑战包括：

1. **非线性动力学：**车辆所受的纵向阻力包含空气阻力（随速度平方增长）、滚动阻力和坡度阻力等非线性项；此外轮胎-路面摩擦系数亦随工况变化。这些非线性因素增加了建模与控制设计的难度。

2. **执行器时变滞后：**动力系统（发动机或电机）的瞬态响应具有惯性，制动液压系统存在动态滞后，轮胎在极限情况下会发生饱和打滑。这些执行器特性可能降低控制有效性，甚至引发车辆稳定性问题。

3. **多目标性能权衡：**纵向控制需要同时考虑安全性（跟踪误差小）、舒适性（加速变化平缓）和经济性（能量消耗低）。然而这三方面往往互相制约，控制策略需要在跟踪精度、响应平顺和能源效率之间寻求折衷。

4. **外部扰动与不确定性：**实际行驶中存在载重变化、风阻波动、坡道路况等外部扰动，以及模型参数的不确定性。这要求控制策略具备一定的鲁棒性来应对环境变化。

针对上述问题，如今已有多种纵向控制方法被提出。从结构简单、易实现的比例–积分–微分（PID）控制，到基于最优控制理论的线性二次型积分（LQI）控制，再到能够显式处理约束的模型预测控制（MPC），不同算法在模型精度要求、实时计算性、实现复杂度和控制性能等方面各具优势。因此，有必要在统一框架下对典型纵向控制策略进行建模和性能比较研究。

我们将围绕车辆纵向运动控制，首先建立包含关键非线性因素和执行器动态的车辆纵向动力学模型，然后设计上述三类代表性的纵向控制器并阐述其原理，利对各控制器在不同目标速度场景下的性能进行对比评估，并将进一步拓展讨论横向 Pure Pursuit 控制的引入及纵横向协同的可能性。

2　模型构建

为了准确模拟车辆纵向运动，我们首先需要建立车辆纵向动力学模型，包括外部阻力建模、执行器动态建模以及制动阶段的防抱死制动模型。

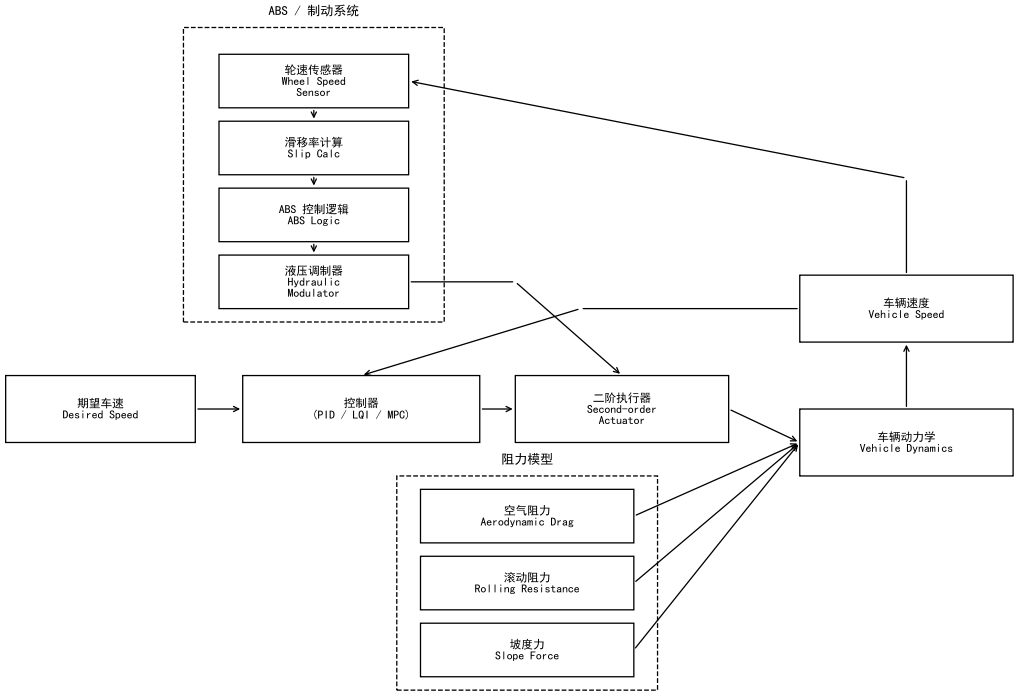

图 1: 汽车纵向动力学模型框架示意图

2.1　车辆纵向动力学模型

车辆沿纵向运动时，可将整车视为一个质心在平直路面上运动的单自由度质点。根据牛顿第二定律，车辆的纵向动力学方程可表示为：

$$m\dot{v}(t) = F_{\text{act}}(t) - \big(F_{\text{drag}}(v) + F_{\text{roll}} + F_{\text{slope}}\big), \tag{1}$$

其中 m 为车辆总质量，$v(t)$ 为车辆纵向速度，$\dot{v}$ 为加速度；$F_{\text{act}}(t)$ 是驱动或制动产生的实际牵引力（正值为驱动力，负值为制动力）；$F_{\text{drag}}(v)$、F_{roll} 和 F_{slope} 分别表示空气阻力、滚动阻力和坡度阻力，引起车辆速度趋于降低的三个外部阻力项。

空气阻力主要由车辆高速行驶时与空气的相对运动产生，可建模为与速度平方成正比：

$$F_{\text{drag}}(v) = \frac{1}{2} \cdot \rho \cdot C_d \cdot A \cdot v^2, \tag{2}$$

其中 ρ 为空气密度，C_d 为车辆的空气阻力系数，A 为车辆迎风正投影面积。

滚动阻力主要源于轮胎形变和地面摩擦等效应，一般可近似为与车重成正比的常值：

$$F_{\text{roll}} = C_r \cdot m \cdot g, \tag{3}$$

其中 C_r 为滚动阻力系数，g 为重力加速度。

坡度阻力则由道路坡度引起，当车辆在坡度为 θ 的斜坡上行驶时，受到沿坡面向下的重力分力：

$$F_{\text{slope}} = mg \sin\theta, \tag{4}$$

在平路行驶时 $\theta = 0$ 则该项为零。上述阻力均随车辆速度和道路环境变化，是纵向控制需克服的主要外部扰动。

在模型仿真中，可根据车辆瞬时速度计算相应的空气阻力和滚动阻力大小，并叠加为总阻力力为：

$$F_{\text{res}} = F_{\text{drag}} + F_{\text{roll}} + F_{\text{slope}}. \tag{5}$$

2.2 二阶执行器动态模型

实际车辆的动力执行单元（发动机/电机驱动系统或液压制动系统）存在惯性和滞后特性，导致控制输入对车辆加速度的影响并非即时达到稳态值。为刻画执行器的动态行为，我们采用二阶线性模型对驱动链进行建模。二阶执行器模型的状态空间形式为：

$$\dot{x}(t) = \begin{pmatrix} 0 & 1 \\ -\omega_n^2 & -2\zeta\,\omega_n \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ \omega_n^2 \end{pmatrix} u(t), \qquad F_{\text{act}}(t) = x_1(t). \tag{6}$$

其中 $x(t) = [x_1(t),, x_2(t)]^\top$ 为执行器系统状态，x_1 可以视为执行器输出的归一化力，$u(t)$ 为控制器下达的牵引/制动指令。模型参数 ω_n 和 ζ 分别为执行机构系统的固有频率和阻尼比，可通过对实车纵向加速和制动过程的数据辨识获得。该二阶模型反映了执行器的惯性（通过 ω_n 限定了最大加速度）和阻尼（通过 ζ 使输出逐渐逼近目标），有效限制了控制信号的带宽，从而更真实地模拟出驱动链对控制输入的滞后响应。在仿真中，控制器产生的原始控制指令 $u(t)$ 将先经由上述二阶动态滤波，得到实际作用于

车辆的牵引力 $F_{\text{act}}(t)$。这样可确保控制计算在考虑执行器动态特性的前提下进行，避免发出超出实际物理能力的过激控制命令。

2.3 ABS 制动模型

在制动工况下（$u(t)$ 为负值），车轮可能在过大的制动力矩作用下发生抱死滑移，导致纵向控制失效和车辆失稳。为此，车辆动力学模型中需引入防抱死制动（ABS）机制，以模拟轮胎在制动时的摩擦特性限制。我们采用基于轮胎滑移率的 Burckhardt 模型来描述制动时轮胎-路面的最大摩擦系数。

滑移率 s 定义为：

$$s = \frac{v - \omega_w r}{\max\big(v,\ \omega_w r\big)}, \tag{7}$$

其中 ω_w 为车轮角速度，r 为车轮半径。

当车辆制动（驱动扭矩为负）且车轮趋于抱死时，滑移率 $s \to 1$。根据 Burckhardt 经验公式，轮胎与路面的动摩擦系数 μ 随滑移率变化的关系可表示为：

$$\mu(s) = c_1\big(1 - e^{-c_2 s}\big) - c_3 s, \tag{8}$$

其中 c_1, c_2, c_3 为根据路面附着特性标定的经验系数。当滑移率较低时，μ 随 s 增大而上升；超过一定滑移率后摩擦系数下降，表示进入抱死打滑区域。

利用上述摩擦模型，可限制制动时实际产生的最大制动力为

$$F_{\text{brake,max}} = \mu(s) \cdot m \cdot g.$$

因此，在模型计算中，当控制指令要求较大的制动力 u_{brake} 时，通过计算相应滑移率和摩擦系数，实际作用的制动力 F_{act} 将被限制为不超过 $\mu(s)\, m\, g$，以符合物理极限。进一步地，为反映制动液压系统的响应延迟，模型中引入一阶滞后环节模拟制动力的建立过程。设定液压滞后时间常数为 τ_h，则制动执行过程可描述为：

$$\dot{F}_{\text{act}} = \frac{1}{\tau_h}\big(F_{\text{brake,cmd}} - F_{\text{act}}\big). \tag{9}$$

其中 $F_{\text{brake,cmd}}$ 为根据控制器指令和摩擦饱和模型计算得到的期望制动力，F_{act} 为实际输出到车辆的制动力。该一阶模型确保制动力平滑施加，避免仿真中的数值不稳定，并更贴近真实车辆制动过程动态。综上，车辆纵向模型在驱动阶段由二阶执行器模型提供牵引力，在制动阶段结合轮胎摩擦饱和与液压滞后模型提供实际制动力，从而在全速度范围内保持模型的一致性和物理合理性。

3 控制器设计

针对上述车辆模型，我们设计了三种纵向速度控制策略：自适应 PID 控制器、增广 LQI 控制器和多目标 MPC 控制器。以下分别介绍各控制器的原理、建模方法及其在性能权衡方面的特点。

3.1 自适应 PID 控制器

PID 控制因结构简单、易于实现而在工业上广泛应用于速度和距离控制等场景。我们在经典比例-积分（PI）控制结构基础上引入基于误差大小的增益自适应调节，以提高在不同误差阶段的响应性能。控制律形式为：

$$u(t) = K_p\big(e(t)\big)\, e(t) + K_i\big(e(t)\big) \int_0^t e(\tau)\,\mathrm{d}\tau + K_d\big(e(t)\big)\, \frac{\mathrm{d}e(t)}{\mathrm{d}t}, \tag{10}$$

其中 $e(t) = v_{\mathrm{ref}} - v(t)$ 为速度跟踪误差，$K_p(e)$ 、$K_i(e)$ 和 $K_d(e)$ 分别为误差相关的比例增益、积分增益和微分增益。

我们选取如下随误差幅值变化的增益函数：

$$K_p(e) = K_{p0}\big(1 + \beta\,|e|\big), \quad K_i(e) = K_{i0}\big(1 + \beta\,|e|\big), \quad K_d(e) = K_{d0}\big(1 + \beta\,|e|\big),$$

其中 K_{p0}, K_{i0}, K_{d0} 分别为比例、积分、微分的基准增益，β 为增益放大因子。

当误差幅值 $|e(t)|$ 较大时，比例增益 $K_p(e)$、积分增益 $K_i(e)$ 及微分增益 $K_d(e)$ 按误差线性增长；此时增大的 K_p 与 K_i 可加速系统对大幅度偏差的收敛，而增大的 K_d 通过加强对误差变化率的抑制作用，有效降低超调并抑制振荡。当误差幅值减小时，上述三路增益同步减小，以在稳态阶段减弱控制作用，抑制噪声放大并优化系统鲁棒性。

自适应 PID 输出的控制信号会先经过幅值限幅，再由二阶执行器模型进行动态滤波后作用于车辆。幅值限幅保证了控制指令不会超过物理允许的最大驱动力 / 制动力，而二阶执行器的滤波特性可有效抑制高频成分，避免直接施加在车辆上引起不稳定。

总体而言，自适应 PID 控制器通过简单的增益调节，在保持算法实现简易性的同时，实现了大误差时的快速响应和小误差时的平滑稳定。

3.2 增广 LQI 控制器

LQI 控制器是在经典线性二次型调节器（LQR）基础上引入积分作用的一种最优控制策略。由于车辆纵向动力学存在非线性阻力项，我们在参考速度 v_0 附近对动力学方

程做泰勒线性化，得到近似模型：

$$\dot{v} \approx a(v_0)\, v + b(v_0)\, u, \tag{11}$$

其中

$$a(v_0) = -\frac{1}{m}\left.\frac{\mathrm{d}F_{\mathrm{res}}}{\mathrm{d}v}\right|_{v_0}, \qquad b(v_0) = \frac{1}{m}.$$

在此基础上，引入积分状态 $\xi(t)$ 来消除稳态误差：

$$\xi(t) = \int_0^t \big(v_{\mathrm{ref}} - v(\tau)\big)\,\mathrm{d}\tau. \tag{12}$$

将原系统状态和积分状态组成增广状态向量

$$x_{\mathrm{aug}}(t) = \begin{pmatrix} v(t) \\ \xi(t) \end{pmatrix},$$

可构造增广系统的状态空间模型：

$$\dot{x}_{\mathrm{aug}}(t) = \underbrace{\begin{pmatrix} a(v_0) & 0 \\ -1 & 0 \end{pmatrix}}_{A_{\mathrm{aug}}} x_{\mathrm{aug}}(t) + \underbrace{\begin{pmatrix} b(v_0) \\ 0 \end{pmatrix}}_{B_{\mathrm{aug}}} u(t). \tag{13}$$

权重选取：Bryson 法则

为避免凭经验调整权重矩阵 Q, R，我们采用 Bryson 法则，根据物理可接受的最大误差和最大控制量自动设定：

$$Q = \mathrm{diag}\big(Q_{11},\, Q_{22}\big), \quad R = \Big(R_{11}\Big),$$

其中

$$Q_{11} = \frac{1}{(\Delta e_{\max})^2}, \quad Q_{22} = \frac{1}{(\Delta I_{\max})^2}, \quad R_{11} = \frac{1}{(\Delta u_{\max})^2}.$$

此处 $\Delta e_{\max}$、$\Delta I_{\max}$、$\Delta u_{\max}$ 分别为允许的最大速度误差、最大误差积分和最大执行器力，通过这些物理标尺得到的权重具有明确工程意义，可直接在不同工况下对比。

抗饱和积分（Anti-Windup）

考虑到执行器力 $u(t)$ 通常受限于 $|u| \le u_{\max}$，当计算出的无饱和控制量 $u_{\mathrm{unsat}} = -K\, x_{\mathrm{aug}}$ 超过极限时，应在饱和分支撤销本步积分增量：

$$\xi(t) \;\xleftarrow{\text{saturate}}\; \xi(t) \;-\; \big(v_{\mathrm{ref}} - v(t)\big)\,\Delta t.$$

该策略可有效避免积分“风暴”导致的超调与恢复震荡。

Riccati 方程与增益调度

在上述增广系统上，设计无限时域 LQI 调节器，即求解连续时间代数黎卡提方程：

$$A_{\text{aug}}^{\top} P + P\, A_{\text{aug}} - P\, B_{\text{aug}}\, R^{-1}\, B_{\text{aug}}^{\top} P + Q = 0, \tag{14}$$

得到对称正定解 P，进而计算最优反馈增益矩阵

$$K = R^{-1} B_{\text{aug}}^{\top} P = \begin{pmatrix} K_v & K_\xi \end{pmatrix}.$$

控制律保持经典形式：

$$u(t) = -K\, x_{\text{aug}}(t) = -\big(K_v\, v(t) + K_\xi\, \xi(t)\big). \tag{15}$$

当车辆运行工况偏离设计点 v_0 较多时，可采用增益调度在线根据当前速度重线性化并重新计算 Q, R 及 K，以保证全工况下的跟踪性能与鲁棒性。

局限性说明

鉴于增广 LQI 控制器的状态空间维度与权重矩阵调谐过程中存在非线性耦合及高维参数搜索的挑战，我们在有限时间和计算资源下尚未获得全局最优的参数组合，故后文中所呈现的仿真结果可能未能完全反映该方法的理论最佳性能，后续工作有待进一步细化与验证。

3.3 多目标 MPC 控制器

模型预测控制（MPC）在离散时间下通过滚动时域优化同时处理跟踪精度、平顺性和能耗，并可直接加入输入与状态约束。设采样周期为 T_s、预测步长为 N，在时刻 $t_k = kT_s$ 定义增广离散动态系统：

$$x_k = \begin{pmatrix} v_k \\ x_{1,k} \\ x_{2,k} \end{pmatrix}, \quad x_{k+1} = A_d\, x_k + B_d\, u_k + d_k,$$

其中

$$A_d = \begin{pmatrix} 1 & \frac{T_s}{m} & 0 \\ 0 & 1 & T_s \\ 0 & -T_s\omega_n^2 & 1 - 2T_s\zeta\omega_n \end{pmatrix}, \quad B_d = \begin{pmatrix} 0 \\ 0 \\ T_s\omega_n^2 \end{pmatrix}, \quad d_k = \begin{pmatrix} -\frac{T_s}{m} F_{\text{res}}(v_k) \\ 0 \\ 0 \end{pmatrix},$$

$$F_{\text{res}}(v) = \tfrac{1}{2}\,\rho\, C_d\, A\, v^2 \;+\; C_r mg \;+\; mg\sin\theta.$$

对应的优化目标函数为：

$$\min_{\{u_i\}_{i=0}^{N-1}} \sum_{i=0}^{N-1} \Big[Q\,(v_i - v_{\text{ref}})^2 \;+\; R\,(u_i - u_{i-1})^2 \;+\; S\,u_i^2 \Big], \tag{16}$$

其中 $\Delta u_i = u_i - u_{i-1}$（$u_{-1}$ 为上次输出），权重 $Q, R, S > 0$ 分别对应速度跟踪误差、控制平顺性与能耗代价。优化需满足动力学约束

$$x_{i+1} = A_d x_i + B_d u_i + d_i, \quad x_0 = x(t_k),$$

以及输入饱和约束

$$u_{\min} \le u_i \le u_{\max}, \quad i = 0, 1, \ldots, N-1.$$

MPC 采用滚动时域策略，在每个时刻仅施加最优序列 $\{u_0^*, u_1^*, \ldots\}$ 的首点 u_0^*，随后更新状态并重新求解优化问题，从而在保证实时性的同时兼顾多步预测的动态效应。

4 仿真实验

4.1 仿真平台与参数配置

为了公平比较三种纵向控制策略的性能，所有仿真实验均在相同的平台下进行。车辆纵向动力学模型、二阶执行器及 ABS 子模块与前述章节一致。关键参数如下：

- 车辆质量 m、空气阻力系数 C_d、迎风面积 A、滚动阻力系数 C_r、坡度角 θ；
- 执行器固有频率 ω_n、阻尼比 ζ、液压滞后时间常数 τ_h；
- 轮胎摩擦模型参数 c_1, c_2, c_3；
- 时间步长 $T_s = 0.2\,\text{s}$，总仿真时间 $T_{\text{sim}} = 40\,\text{s}$。

4.2 仿真方案与评价指标

本研究在三种典型速度工况下开展数值仿真：10 m/s（市区低速）、20 m/s（郊区中速）和 30 m/s（高速公路）。车辆由静止点开始启动，在加速阶段到达各自目标速度后维持匀速巡航若干秒，以充分考察各控制器的动态性能与稳态特性。

为定量评估算法性能，选取以下三项指标：

- **均方误差 (MSE)**

$$\mathrm{MSE} = \frac{1}{N}\sum_{k=1}^{N}\left(v_k - v_{\mathrm{ref}}\right)^2,$$

表征速度跟踪的总体偏差。

- **最大超调量 (Overshoot)**

$$\mathrm{OS} = \max_k\left(v_k - v_{\mathrm{ref}}\right),$$

反映瞬态阶段的最大过冲幅度。

- **能量消耗 (Energy)**

$$E = \sum_{k=1}^{N}\left|u_k\right| T_s,$$

近似表示在整个仿真过程中加速与制动所需的能量总量。

4.3 速度跟踪对比

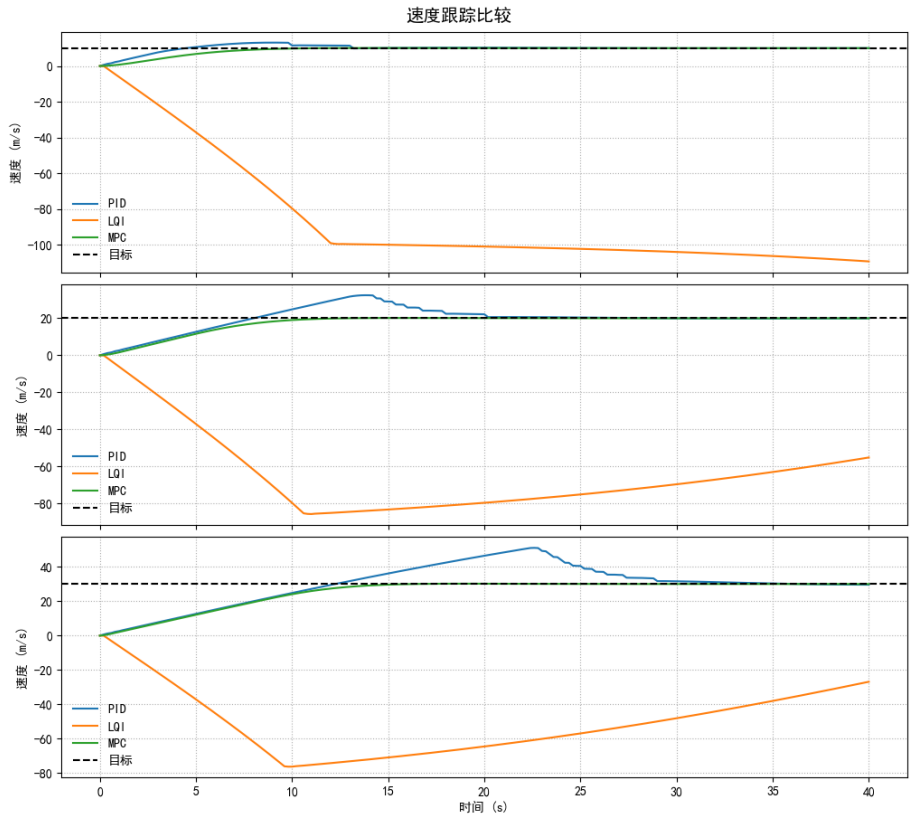

图 2: 三种控制器在不同目标速度下的跟踪响应

图 2 对比了 PID、LQI 和 MPC 在三种速度阶跃（10 m/s、20 m/s、30 m/s）下的跟踪曲线，可归纳如下：

- **10 m/s 工况**：PID 响应迅速但出现近 1 m/s 的超调及轻微振荡；LQI 因积分饱和导致输出反向，下冲至约 - 100 m/s 后缓慢回升；MPC 响应平滑，无超调且稳态误差最小。

- **20 m/s 工况**：PID 超调幅度增大至约 30 m/s 并产生衰减震荡；LQI 依旧先出现深度下冲（约 - 85 m/s），然后缓慢恢复；MPC 略有欠调但收敛最为平顺。

- **30 m/s 工况**：PID 超调显著（峰值约 50 m/s）并出现多次往返振荡；LQI 同样大幅下冲至约 - 78 m/s，恢复过程迟缓；MPC 依旧保持近零超调，并快速稳定在目标附近。

4.4 油门 / 制动信号（归一化）

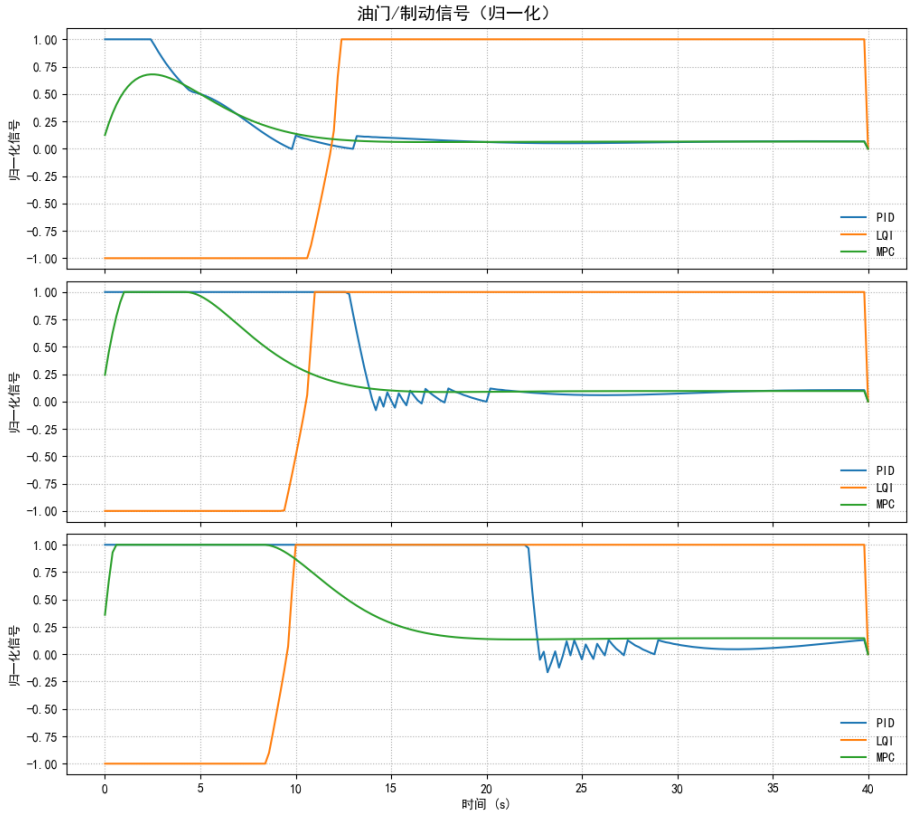

图 3: 三种控制器的归一化油门 / 制动命令

图 3 给出了三种算法在加速与巡航阶段的控制信号（归一化至 $[-1, 1]$）：

- **PID**：初期饱和输出（± 1），随误差减小在油门与制动之间频繁切换，稳态阶段残余小幅振荡。

- **LQI**：命令呈分段常数形式——先持续最大制动，再突变至最大油门，切换果断但平顺性不足。
- **MPC**：加速阶段中等油门（约 0.6–1.0），随后信号平滑衰减至小幅稳态值，在响应速度与舒适性之间实现良好折中。

4.5 速度误差分析

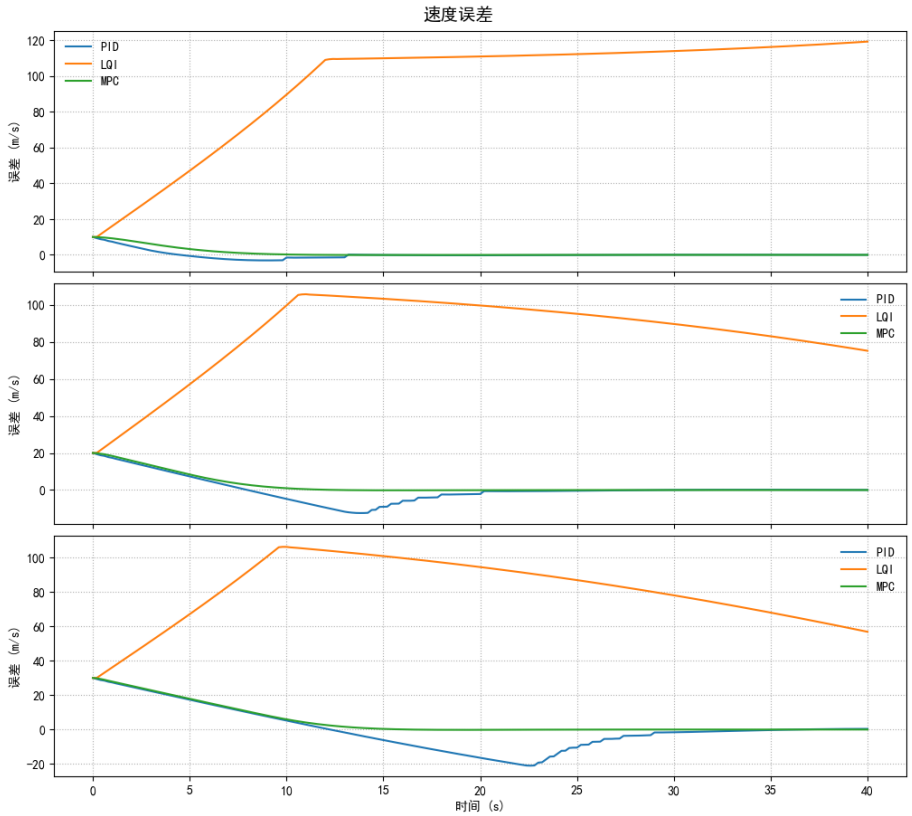

图 4: 三种控制器的速度误差比较

图 4 展示了速度误差 $e_k = v_{\text{ref}} - v_k$ 在三个速度阶跃下的时域演变：

- **PID**：10 m/s 工况下误差平稳衰减且无过冲；20 m/s 和 30 m/s 工况出现约–25 m/s 的负向过冲，并伴有持续振荡，直至 20–25 s 后才收敛，反映出在限幅与非线性作用下 PID 性能大幅下降。
- **LQI**：误差在瞬态阶段被推升至 110 m/s（10 m/s 工况）、105 m/s（20 m/s、30 m/s 工况）左右，并在高值附近滞留较长时间，直至积分器逐步解锁后才缓慢回落，表明其对饱和与约束处理不足。

- **MPC**：所有工况下在 5 s 内将误差压缩至零，无明显过冲或振荡，验证了 MPC 在考虑系统约束与未来轨迹时的前瞻性优化能力。

4.6 牵引力 / 制动力命令分析

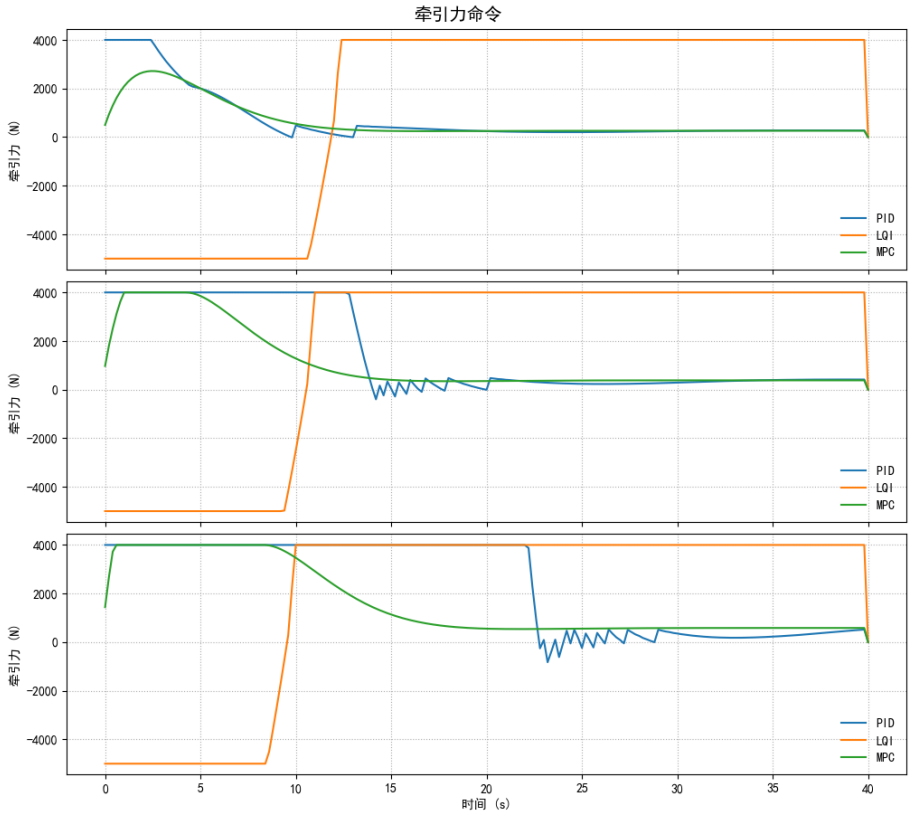

图 5: 三种控制器的牵引力 / 制动力输出

图 5 对比了车辆在三种工况下的牵引力（正值）与制动力（负值）指令：

- **PID**：初始阶段输出近 4 kN 的峰值牵引力，随后因速度超调或阻力增大迅速切换至制动力，并在稳态前产生多次小幅振荡，显示出对饱和区段的响应紊乱。
- **LQI**：命令基本锁定在最大牵引或最大制动两端，中间区段极少跨越，突显其在瞬态阶段平滑性不足，但在稳态跟踪时保持恒定输出的特点。
- **MPC**：加速阶段先给出足够牵引力以缩短响应时间，然后信号平滑过渡至小幅牵引力，在稳态阶段与 LQI 相近，但避免了过冲与反向制动，实现了约束内的最优折中。

4.7 综合性能评估

本节将柱状图（图 6）与雷达图及热力图（图 7）结合，对 PID、LQI 和 MPC 三种控制器在 10 m/s、20 m/s、30 m/s 三档工况下的均方误差（MSE）、最大超调量和能量消耗进行全方位评析。

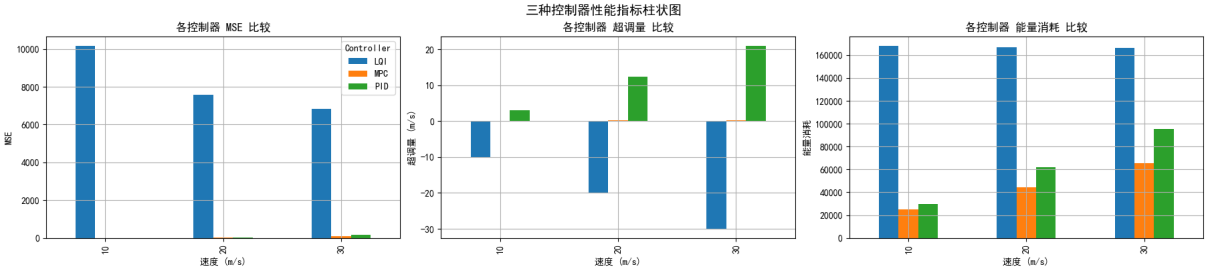

图 6: 三种控制器在不同速度下的 MSE、超调量和能耗柱状对比

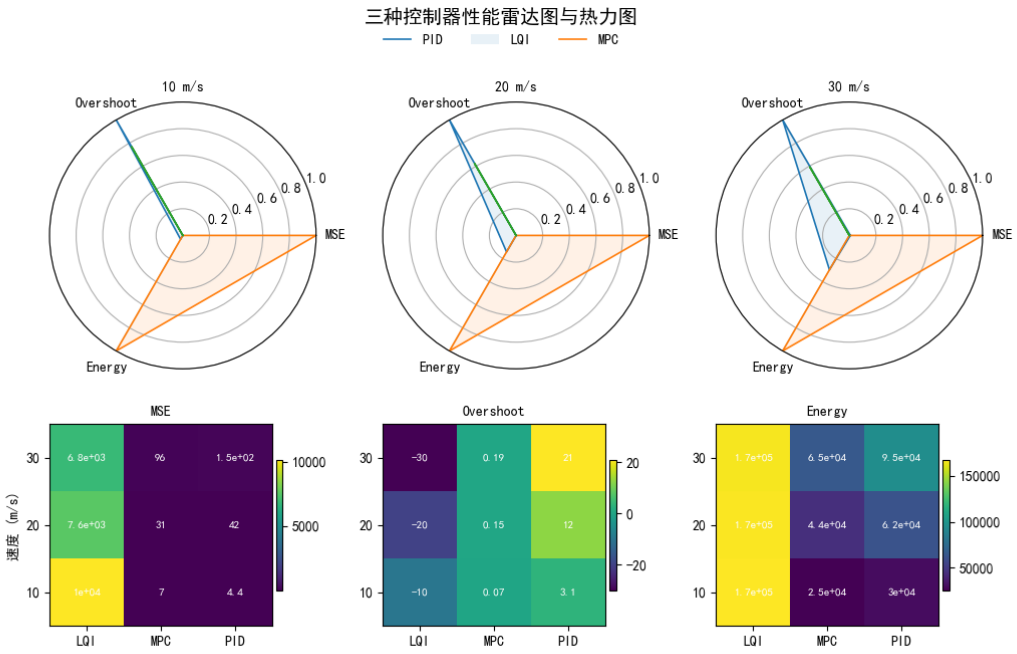

图 7: 三种控制器性能雷达图与热力图

图 6 的柱状对比揭示：

- **MSE**：MPC 在所有速度档位始终保持最低，LQI 次之，PID 因饱和及振荡导致 MSE 大幅增高；
- **超调量**：MPC 接近零，LQI 控制在微小正负范围内（下冲视作负超调），PID 随速度上升超调幅度急剧增大；

- **能量消耗**：MPC 最低，LQI 居中，PID 因频繁制动与加速切换能耗最高。

进一步借助图 7 的雷达图和热力图，从归一化视角与绝对值变化趋势两方面深入分析：

4.7.1 雷达图（归一化指标）

- MPC 的雷达图面积最大，表明其在 MSE、超调和能耗三项指标上均接近理想点，且三者之间保持平衡；
- PID 在 10 m/s 时对超调有一定抑制，但 MSE 与能耗表现中游；随着速度提高，其雷达图快速向内收缩，凸显超调和能耗随速增大而恶化；
- LQI 雷达图面积最小，尤其在 MSE 维度远离圆心，反映零稳态误差的优势无法弥补其瞬态性能与能效的不足。

4.7.2 热力图（绝对值趋势）

- **MSE**：LQI 从 $\sim 1.04 \times 10^4$ 降至 $\sim 6.8 \times 10^3$，但始终远高于 PID（$\sim 1.5 \times 10^2 \rightarrow 4.2 \times 10^1$）和 MPC（7–96）；PID 在 20 m/s 时误差最小，30 m/s 后略有回升；MPC 始终维持极低水平；
- **超调量**：LQI 产生持续负向下冲（–10、–20、–30 m/s），PID 超调从 3.1 m/s 攀升至 21 m/s，MPC 仅在 0.07–0.19 m/s 之间微幅变化；
- **能量消耗**：三者能耗均随速度升高而上升，MPC 最低（$2.5 \times 10^4 \rightarrow 6.5 \times 10^4$），PID 居中（$3.0 \times 10^4 \rightarrow 9.5 \times 10^4$），LQI 最高且近乎恒定（$\sim 1.7 \times 10^5$）。

综上，三种可视化手段相互印证：MPC 在精度、动态性能和经济性方面实现了最优折中；LQI 虽可保证零稳态误差，但瞬态阶段的过冲与高能耗使其综合表现不佳；PID 结构实现最为简便，但在高动态和饱和约束环境下易产生显著过冲和能耗激增，需结合饱和保护或前馈补偿等技术进行改进。

5 横向控制与纵横向协同优化

5.1 横向分层控制框架

在恒速 $v_{\text{lat}} = 15\,\text{m/s}$ 条件下，采用 Pure Pursuit 算法对圆弧轨迹进行跟踪，其核心步骤为：

1. 依据车辆当前位置 (x, y) 与航向角 ψ，在参考路径上选取前视距离 L_d 处的目标点 $(x_{\mathrm{tgt}}, y_{\mathrm{tgt}})$；

2. 计算目标点在车辆坐标系下的横向偏差角

$$\alpha = \arctan 2\big(y_{\mathrm{tgt}} - y,\; x_{\mathrm{tgt}} - x\big) \;-\; \psi;$$

3. 生成转向命令

$$\delta = \arctan\!\left(\frac{2\,L\,\sin\alpha}{L_d}\right),$$

其中 L 为车辆轴距。该过程在每个采样周期重复执行，以“追—逐”目标点的方式实现路径跟踪。

仿真结果见图 8，横向偏差控制在 0.5 m 量级以内，转向命令平滑，验证了分层架构在常规工况下的可行性。

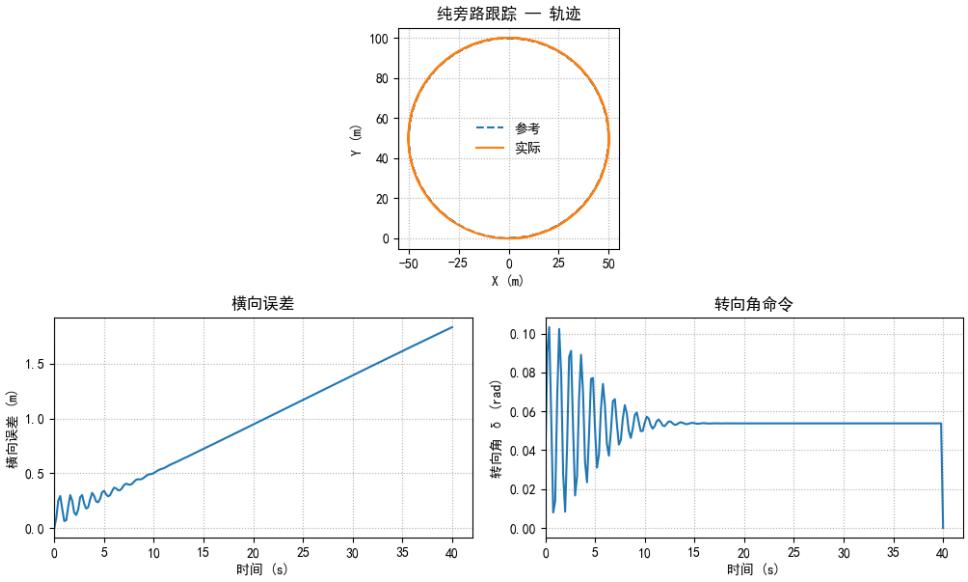

图 8: Pure Pursuit 横向跟踪仿真结果

5.2 纵横向协同优化

我们目前建立的分层控制框架在以下方面仍存在局限性：

- **耦合效应忽略**：分层架构将纵向与横向控制相互独立，在急加速或急减速等极限纵向动作发生时，忽略了侧向力变化对轨迹跟踪的影响，可能导致车辆偏离路径或侧倾风险增大。

- **性能优化受限**：分层控制无法在同一优化过程中兼顾纵向速度跟踪误差与横向路径偏差，难以对油门/制动与转向角进行联合调度，故在整车能效与乘坐舒适性方面存在优化空间。

为克服上述不足，本文提出了基于 MIMO–MPC 的纵横向一体化控制框架展望，其核心思路包括：

- 将纵向速度误差 $e_v = v - v_{\mathrm{ref}}$ 与横向偏差 e_y 一并纳入状态向量

$$\mathbf{x}(k) = \big[e_v(k),\ \dot{e}_v(k),\ e_y(k),\ \dot{e}_y(k)\big]^{\mathsf{T}},$$

  并将油门/制动增量 Δu 与转向角增量 $\Delta\delta$ 作为联合控制量。

- 构造多目标代价函数

$$J = \sum_{k=0}^{N-1} \Big[Q_v\, e_v(k)^2 + Q_y\, e_y(k)^2 + R_u\, \Delta u(k)^2 + R_\delta\, \Delta\delta(k)^2 \Big],$$

  并在优化约束中加入执行器饱和、车道边界及侧倾加速度上下限等安全约束，以实现对整车运动的协调控制。

该一体化方案能够在在线优化过程中同时兼顾纵横向性能，但对模型精度、计算复杂度及实时求解器性能提出了更高要求。未来研究可进一步结合高效求解算法与自适应模型更新策略，以提升系统在复杂道路场景下的鲁棒性与经济性。

6 结论

本文基于考虑空气阻力、滚动阻力、坡度阻力、二阶执行器动力学及 ABS 制动特性的纵向动力学仿真平台，对自适应 PID、增广 LQI 与多目标 MPC 三种控制策略进行了系统比较，并在 10 m/s、20 m/s 与 30 m/s 三种典型速度工况下，以均方误差 (MSE)、最大超调量 (OS) 与能量消耗三项指标对控制性能进行了定量评估。主要结论如下：

1. **自适应 PID**：结构简单、调节直观，在低速工况下响应快速，但在高速或重阻力环境中易发生超调与振荡，导致 MSE 与能耗显著升高，鲁棒性不足。

2. **增广 LQI**：通过积分项保证零稳态误差，具备较好中速工况性能，但在执行器饱和约束下出现下冲与恢复迟缓现象，OS 与能耗均高于 MPC，限制了其实用性。

3. **多目标 MPC**：显式考虑系统动态与约束，对未来轨迹进行预测并在优化过程中平衡跟踪性能与能耗，实现在各速率工况下均获得最低的 MSE、最小的 OS 及最佳的能量效率，体现出最优的多目标折中效果。

综上所述，多目标 MPC 在数值仿真中展现了卓越的综合性能，具有显著的工程应用价值。未来工作可在此基础上进一步发展纵横向一体化 MIMO–MPC 方法，并结合高效在线求解器与自适应模型更新策略，以提升自动驾驶控制系统在复杂道路场景中的稳健性与经济性。“‘

A 本报告所使用代码

```python
# === 1. 核心库加载与字体设置 ===
import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve_continuous_are
from scipy.optimize import minimize
import pandas as pd

# 确保图像保存目录存在
fig_dir = os.path.join(os.getcwd(), 'figures_VHC')
os.makedirs(fig_dir, exist_ok=True)

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# === 2. 车辆参数类 ===
class VehicleParams:
    def __init__(self):
        # 基本参数
        self.m = 1500        # kg
        self.g = 9.81        # m/s²
        self.wheelbase = 2.7 # m
        # 空气动力学
        self.Cd = 0.30
        self.A  = 2.2        # m²
        self.rho = 1.225     # kg/m³
        # 滚动阻力
        self.Cr = 0.015
        # 执行器
        self.wn   = 5.65
        self.zeta = 0.707
        # ABS
```

```
        self.r = 0.3          # m
        self.J = 0.35         # kg · m²
        self.c1, self.c2, self.c3 = 1.28, 23.99, 0.52
        self.tau_hyd = 0.5    # s
        # 牵引 / 制动力限
        self.F_max =  4000    # N
        self.F_min = -5000    # N
        self.tau   = 0.5      # s (滞后)

# === 3. 工具函数: 性能指标与阻力模型 ===
def compute_metrics(v_arr, u_arr, v_ref, dt):
    mse       = np.mean((v_arr - v_ref) ** 2)
    overshoot = np.max(v_arr - v_ref)
    energy    = np.sum(np.abs(u_arr)) * dt
    return mse, overshoot, energy

def disturbance(v, p: VehicleParams):
    drag = 0.5 * p.rho * p.Cd * p.A * v**2
    roll = p.Cr * p.m * p.g
    return drag + roll

# === 4. 二阶执行器模型 ===
class SecondOrderActuator:
    def __init__(self, wn, zeta):
        self.A = np.array([[0, 1], [-wn**2, -2*zeta*wn]])
        self.B = np.array([[0], [wn**2]])
        self.x = np.zeros((2, 1))
    def reset(self):
        self.x.fill(0)
    def step(self, u, dt):
        self.x += (self.A @ self.x + self.B * u) * dt
        return self.x[0, 0]

```

```python
# === 5. ABS 控制器 ===
class ABSController:
    def __init__(self, p: VehicleParams):
        self.r, self.J = p.r, p.J
        self.c1, self.c2, self.c3 = p.c1, p.c2, p.c3
        self.tau = p.tau_hyd
        self.omega = 0.0
    def burckhardt(self, slip):
        return self.c1 * (1 - np.exp(-self.c2 * slip)) - self.c3 * slip
    def reset(self):
        self.omega = 0.0
    def step(self, v, u_brake, dt, p: VehicleParams):
        slip = (v - self.omega * p.r) / max(v, self.omega * p.r, 1e-3)
        slip = np.clip(slip, -1, 1)
        mu   = self.burckhardt(slip)
        F_des = mu * p.m * p.g * np.sign(u_brake)
        self.omega += (dt / self.tau) * (u_brake - self.omega)
        return F_des

# === 6. 自适应 PID 控制器 ===
class PIDControllerAdaptive:
    def __init__(self, Kp0, Ki0, beta, p: VehicleParams):
        self.Kp0, self.Ki0, self.beta = Kp0, Ki0, beta
        self.integrator = 0.0
        self.act = SecondOrderActuator(p.wn, p.zeta)
    def reset(self):
        self.integrator = 0.0
        self.act.reset()
    def step(self, error, dt):
        Kp = self.Kp0 * (1 + self.beta * abs(error))
        Ki = self.Ki0 * (1 + self.beta * abs(error))
        self.integrator += error * dt
        u = np.clip(Kp*error + Ki*self.integrator, -5000, 4000)
```

```
99          F_act = self.act.step(u, dt)
100         return u, F_act
101
102 # === 7. 增广 LQI 控制器 ===
103 '''
104 class LQIController:
105     def __init__(self, v0, p: VehicleParams):
106         a0    = 0.5 * p.rho * p.Cd * p.A * v0**2 + p.Cr * p.m * p.g
107         b0    = p.rho * p.Cd * p.A * v0
108         A_lin = np.array([[-b0 / p.m]])
109         B_lin = np.array([[1.0 / p.m]])
110         A_aug = np.block([
111             [ A_lin,           np.zeros((1,1)) ],
112             [   -1.0,          np.zeros((1,1)) ]
113         ])
114         B_aug = np.vstack([ B_lin, [0.0] ])
115         e_max     = 0.5     # 最大速度误差 (m/s)
116         I_max     = 5.0     # 最大误差积分 (m · s)
117         u_pos_max = 4000.0  # 最大输出力 (N)
118         Q11 = 1.0 / e_max**2
119         Q22 = 1.0 / I_max**2
120         R11 = 1.0 / u_pos_max**2
121         Q = np.diag([Q11, Q22])
122         R = np.array([[R11]])
123         P      = solve_continuous_are(A_aug, B_aug, Q, R)
124         self.K = np.linalg.inv(R) @ B_aug.T @ P  # shape (1,2)
125         self.x       = np.zeros((2,1))              # [e;  e]
126         self.u_min   = -5000.0
127         self.u_max   = +4000.0
128         self.act     = SecondOrderActuator(p.wn, p.zeta)
129     def reset(self):
130         self.x.fill(0.0)
131         self.act.reset()
```

```
    def step(self, v, v_ref, dt):
        self.x[:] = np.nan_to_num(self.x, nan=0.0, posinf=0.0,
        ↪  neginf=0.0)
        e = v_ref - v
        self.x[0,0] = e
        self.x[1,0] += e * dt
        u_unsat = - (self.K @ self.x).item()
        if not np.isfinite(u_unsat):
            u_unsat = 0.0
        u = np.clip(u_unsat, self.u_min, self.u_max)
        if u != u_unsat:
            delta_I = e * dt
            if np.isfinite(delta_I):
                self.x[1,0] -= delta_I
        return u, self.act.step(u, dt)
---
完整版 LQI 控制器代码如上，由于客观限制，在实际运行中我们选择了简化版的 LQI
↪  控制器。
'''
class LQIController:
    def __init__(self, v0, p: VehicleParams):
        a0 = 0.5 * p.rho * p.Cd * p.A * v0**2 + p.Cr * p.m * p.g
        b0 = p.rho * p.Cd * p.A * v0
        A_lin = np.array([[-b0 / p.m]])
        B_lin = np.array([[1 / p.m]])
        A_aug = np.block([[A_lin, np.zeros((1,1))], [-1,
        ↪  np.zeros((1,1))]])
        B_aug = np.vstack([B_lin, [0]])
        Q, R = np.diag([1000, 100]), np.array([[0.01]])
        P = solve_continuous_are(A_aug, B_aug, Q, R)
        self.K = np.linalg.inv(R) @ B_aug.T @ P
        self.x  = np.zeros((2,1))
        self.act = SecondOrderActuator(p.wn, p.zeta)
```

```
162     def reset(self):
163         self.x.fill(0)
164         self.act.reset()
165     def step(self, v, v_ref, dt):
166         e = v_ref - v
167         self.x[0,0] = e
168         self.x[1,0] += e*dt
169         u = np.clip(- (self.K @ self.x).item(), -5000, 4000)
170         return u, self.act.step(u, dt)
171 
172 # === 8. 多目标 MPC 控制器 ===
173 class MPCControllerMultiObjective:
174     def __init__(self, N, q, r, e_energy, p: VehicleParams):
175         self.N, self.q, self.r, self.e_energy = N, q, r, e_energy
176         self.act = SecondOrderActuator(p.wn, p.zeta)
177         self.u_prev = 0.0
178     def reset(self):
179         self.act.reset()
180         self.u_prev = 0.0
181     def step(self, v_curr, v_ref, dt):
182         def obj(u):
183             v_p, Fp = v_curr, self.act.x[0,0]
184             cost = 0
185             for ui in u:
186                 Fp += (dt/0.5)*(ui - Fp)
187                 d = disturbance(v_p, p)
188                 v_p += dt*(Fp - d)/p.m
189                 cost += ( self.q*(v_p - v_ref)**2
190                         + self.r*(ui - self.u_prev)**2
191                         + self.e_energy*ui**2 )
192             return cost
193         u0 = np.ones(self.N) * self.u_prev
```

```
194         res = minimize(obj, u0, bounds=[(-5000, 4000)]*self.N,
            ↪  options={'maxiter':100})
195         u_cmd = res.x[0]
196         self.u_prev = u_cmd
197         return u_cmd, self.act.step(u_cmd, dt)
198
199 # === 9. 主仿真循环 ===
200 p = VehicleParams()
201 dt, T_final = 0.2, 40
202 time = np.arange(0, T_final + dt, dt)
203 desired_speeds = [10, 20, 30]
204 results = {}
205 for v_t in desired_speeds:
206     pid = PIDControllerAdaptive(300, 100, 0.1, p)
207     lqi = LQIController(v_t, p)
208     mpc = MPCControllerMultiObjective(40, 50, 0.005, 1e-6, p)
209     abs_ctrl = ABSController(p)
210     for c in (pid, lqi, mpc, abs_ctrl): c.reset()
211     v_pid = np.zeros_like(time); v_lqi = v_pid.copy(); v_mpc =
        ↪  v_pid.copy()
212     u_pid = v_pid.copy(); u_lqi = v_pid.copy(); u_mpc = v_pid.copy()
213     for k in range(len(time)-1):
214         e1, e2, e3 = v_t - v_pid[k], v_t - v_lqi[k], v_t - v_mpc[k]
215         d1, d2, d3 = disturbance(v_pid[k],p), disturbance(v_lqi[k],p),
            ↪  disturbance(v_mpc[k],p)
216         u1,_ = pid.step(e1, dt)
217         u2,_ = lqi.step(v_lqi[k], v_t, dt)
218         u3,_ = mpc.step(v_mpc[k], v_t, dt)
219         F1 = pid.act.step(u1, dt) if u1>=0 else abs_ctrl.step(v_pid[k],
            ↪  u1, dt, p)
220         F2 = lqi.act.step(u2, dt) if u2>=0 else abs_ctrl.step(v_lqi[k],
            ↪  u2, dt, p)
```

```
221         F3 = mpc.act.step(u3, dt) if u3>=0 else abs_ctrl.step(v_mpc[k],
            ↪ u3, dt, p)
222         v_pid[k+1] = v_pid[k] + dt*(F1 - d1)/p.m
223         v_lqi[k+1] = v_lqi[k] + dt*(F2 - d2)/p.m
224         v_mpc[k+1] = v_mpc[k] + dt*(F3 - d3)/p.m
225         u_pid[k], u_lqi[k], u_mpc[k] = u1, u2, u3
226     results[v_t] = {'v':(v_pid, v_lqi, v_mpc), 'u':(u_pid, u_lqi,
        ↪ u_mpc)}
227
228 # === 10. 绘图函数 ===
229 def plot_speed_tracking(time, results, speeds):
230     fig, axes = plt.subplots(len(speeds), 1, figsize=(10, 3 *
        ↪ len(speeds)), sharex=True, constrained_layout=True)
231     for ax, v_t in zip(axes, speeds):
232         vp, vl, vm = results[v_t]['v']
233         ax.plot(time, vp, label='PID', linewidth=1.5)
234         ax.plot(time, vl, label='LQI', linewidth=1.5)
235         ax.plot(time, vm, label='MPC', linewidth=1.5)
236         ax.axhline(v_t, linestyle='--', color='k', label='目标')
237         ax.set_ylabel('速度 (m/s)')
238         ax.legend(frameon=False)
239         ax.grid(True, linestyle=':')
240     axes[-1].set_xlabel('时间 (s)')
241     fig.suptitle('速度跟踪比较', fontsize=14)
242     fig.savefig(os.path.join(fig_dir, 'speed_tracking.png'))
243     plt.close(fig)
244
245 def plot_normalized_signal(time, results, speeds, p):
246     fig, axes = plt.subplots(len(speeds), 1, figsize=(10, 3 *
        ↪ len(speeds)), sharex=True, constrained_layout=True)
247     for ax, v_t in zip(axes, speeds):
248         u1, u2, u3 = results[v_t]['u']
249         u1n = np.where(u1 >= 0, u1 / p.F_max, u1 / (-p.F_min))
```

```
250         u2n = np.where(u2 >= 0, u2 / p.F_max, u2 / (-p.F_min))
251         u3n = np.where(u3 >= 0, u3 / p.F_max, u3 / (-p.F_min))
252         ax.plot(time, u1n, label='PID')
253         ax.plot(time, u2n, label='LQI')
254         ax.plot(time, u3n, label='MPC')
255         ax.set_ylabel('归一化信号')
256         ax.legend(frameon=False)
257         ax.grid(True, linestyle=':')
258     axes[-1].set_xlabel('时间 (s)')
259     fig.suptitle('油门/制动信号（归一化）', fontsize=14)
260     fig.savefig(os.path.join(fig_dir, 'normalized_signal.png'))
261     plt.close(fig)
262
263 def plot_error(time, results, speeds):
264     fig, axes = plt.subplots(len(speeds), 1, figsize=(10, 3 *
        ↪ len(speeds)), sharex=True, constrained_layout=True)
265     for ax, v_t in zip(axes, speeds):
266         vp, vl, vm = results[v_t]['v']
267         ax.plot(time, v_t - vp, label='PID')
268         ax.plot(time, v_t - vl, label='LQI')
269         ax.plot(time, v_t - vm, label='MPC')
270         ax.set_ylabel('误差 (m/s)')
271         ax.legend(frameon=False)
272         ax.grid(True, linestyle=':')
273     axes[-1].set_xlabel('时间 (s)')
274     fig.suptitle('速度误差', fontsize=14)
275     fig.savefig(os.path.join(fig_dir, 'error_tracking.png'))
276     plt.close(fig)
277
278 def plot_traction(time, results, speeds):
279     fig, axes = plt.subplots(len(speeds), 1, figsize=(10, 3 *
        ↪ len(speeds)), sharex=True, constrained_layout=True)
280     for ax, v_t in zip(axes, speeds):
```

```
        u1, u2, u3 = results[v_t]['u']
        ax.plot(time, u1, label='PID')
        ax.plot(time, u2, label='LQI')
        ax.plot(time, u3, label='MPC')
        ax.set_ylabel('牵引力 (N)')
        ax.legend(frameon=False)
        ax.grid(True, linestyle=':')
    axes[-1].set_xlabel('时间 (s)')
    fig.suptitle('牵引力命令', fontsize=14)
    fig.savefig(os.path.join(fig_dir, 'traction_command.png'))
    plt.close(fig)

# === 11. 性能指标计算与 DataFrame ===
metrics = {'Speed':[], 'Controller':[], 'MSE':[], 'Overshoot':[],
    ↪ 'Energy':[]}
for v_t in desired_speeds:
    vp, vl, vm = results[v_t]['v']
    u1, u2, u3 = results[v_t]['u']
    for name, v_arr, u_arr in zip(['PID','LQI','MPC'], [vp,vl,vm],
        ↪ [u1,u2,u3]):
        mse, osr, ene = compute_metrics(v_arr, u_arr, v_t, dt)
        metrics['Speed'].append(v_t); metrics['Controller'].append(name)
        metrics['MSE'].append(mse); metrics['Overshoot'].append(osr);
            ↪ metrics['Energy'].append(ene)

df = pd.DataFrame(metrics)

# 横向模拟参数
R, v_lat, Ld = 50, 15, 5
t_lat = np.arange(0, T_final + dt, dt)
phi = v_lat * t_lat / R
x_ref = R * np.sin(phi)
y_ref = R * (1 - np.cos(phi))
```

```
311
312 n = len(t_lat)
313 x = np.zeros(n)
314 y = np.zeros(n)
315 psi = np.zeros(n)
316 delta_hist = np.zeros(n)
317 for k in range(n - 1):
318     idx = min(k + int(Ld / (v_lat * dt)), n - 1)
319     dx, dy = x_ref[idx] - x[k], y_ref[idx] - y[k]
320     alpha = np.arctan2(dy, dx) - psi[k]
321     delta = np.arctan2(2 * p.wheelbase * np.sin(alpha), Ld)
322     delta_hist[k] = delta
323     x[k + 1] = x[k] + v_lat * np.cos(psi[k]) * dt
324     y[k + 1] = y[k] + v_lat * np.sin(psi[k]) * dt
325     psi[k + 1] = psi[k] + v_lat / p.wheelbase * np.tan(delta) * dt
326
327 lat_error = np.hypot(x - x_ref, y - y_ref)
328
329 # 轨迹与误差图
330 fig, axd = plt.subplot_mosaic(
331     [['traj', 'traj'],
332      ['err',  'delta']],
333     figsize=(10, 6),
334     constrained_layout=True
335 )
336 # 轨迹
337 ax = axd['traj']
338 ax.plot(x_ref, y_ref, '--', label='参考')
339 ax.plot(x, y, label='实际')
340 ax.set_aspect('equal', 'box')
341 ax.set_xlabel('X (m)')
342 ax.set_ylabel('Y (m)')
343 ax.set_title('纯旁路跟踪 —轨迹')
```

```
344 ax.legend(frameon=False)
345 ax.grid(ls=':')
346 # 横向误差
347 ax = axd['err']
348 ax.plot(t_lat, lat_error)
349 ax.set_xlim(left=0)
350 ax.set_ylabel('横向误差 (m)')
351 ax.set_title('横向误差')
352 ax.grid(ls=':')
353 # 转向角
354 ax = axd['delta']
355 ax.plot(t_lat, delta_hist)
356 ax.set_xlim(left=0)
357 ax.set_ylabel('转向角  (rad)')
358 ax.set_title('转向角命令')
359 ax.grid(ls=':')
360 for key in ['err', 'delta']:
361     axd[key].set_xlabel('时间 (s)')
362 fig.savefig(os.path.join(fig_dir, 'lateral_tracking.png'))
363 plt.close(fig)
364
365 # === 12. 结果绘制调用 ===
366 plot_speed_tracking(time, results, desired_speeds)
367 plot_normalized_signal(time, results, desired_speeds, p)
368 plot_error(time, results, desired_speeds)
369 plot_traction(time, results, desired_speeds)
370
371 # 性能柱状图
372 fig_bar, ax_bar = plt.subplots(1, 3, figsize=(18, 4),
    ↪  constrained_layout=True)
373 for ax, metric, ylabel, title in zip(
374         ax_bar,
375         ['MSE', 'Overshoot', 'Energy'],
```

```
        ['MSE', '超调量 (m/s)', '能量消耗'],
        ['各控制器 MSE 比较', '各控制器 超调量 比较', '各控制器 能量消耗
        ↪ 比较']):
    df.pivot(index='Speed', columns='Controller', values=metric) \
      .plot(kind='bar', ax=ax, grid=True, legend=(metric == 'MSE'))
    ax.set_xlabel('速度 (m/s)')
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    if metric != 'MSE' and ax.get_legend() is not None:
        ax.get_legend().remove()
fig_bar.suptitle('三种控制器性能指标柱状图', fontsize=14)
fig_bar.savefig(os.path.join(fig_dir, 'performance_bar.png'))
plt.close(fig_bar)

# === 13. 雷达图与热力图 ===
metrics_list = ['MSE', 'Overshoot', 'Energy']
controls  = ['PID', 'LQI', 'MPC']
fig = plt.figure(figsize=(12, 7))
gs  = fig.add_gridspec(2, 3, height_ratios=[3, 2], hspace=0.15,
↪ wspace=0.25)
angles = np.linspace(0, 2*np.pi, len(metrics_list),
↪ endpoint=False).tolist() + [0]
for col, v_speed in enumerate(desired_speeds):
    ax = fig.add_subplot(gs[0, col], projection='polar')
    sub = df.set_index(['Speed', 'Controller']).loc[v_speed]
    vals = np.array([sub.loc[c, metrics_list].values for c in controls])
    mins, maxs = vals.min(0), vals.max(0)
    norm = (vals - mins) / np.where(maxs-mins == 0, 1, maxs-mins)
    for idx, c in enumerate(controls):
        data = np.r_[norm[idx], norm[idx, 0]]
        ax.plot(angles, data, label=c, linewidth=1.2)
        ax.fill(angles, data, alpha=0.1)
    ax.set_xticks(angles[:-1])
```

```
406     ax.set_xticklabels(metrics_list)
407     ax.set_ylim(0, 1)
408     ax.set_title(f'{v_speed} m/s', pad=8, fontsize=10)
409 fig.legend(controls, loc='upper center', bbox_to_anchor=(0.5, 0.97),
    ↪  ncol=3, frameon=False)
410
411 cmap = 'viridis'
412 for col, metric in enumerate(metrics_list):
413     ax   = fig.add_subplot(gs[1, col])
414     data = df.pivot(index='Speed', columns='Controller', values=metric)
415     im   = ax.imshow(data, cmap=cmap, origin='lower', aspect='auto')
416     ax.set_xticks(range(len(data.columns)));
        ↪  ax.set_xticklabels(data.columns)
417     ax.set_yticks(range(len(data.index)));
        ↪  ax.set_yticklabels(data.index)
418     ax.set_title(metric, fontsize=10)
419     for i in range(data.shape[0]):
420         for j in range(data.shape[1]):
421             ax.text(j, i, f'{data.iloc[i,j]:.2g}', ha='center',
                ↪  va='center', color='white', fontsize=8)
422     if col == 0:
423         ax.set_ylabel('速度 (m/s)')
424     fig.colorbar(im, ax=ax, shrink=0.65, pad=0.015)
425 fig.suptitle('三种控制器性能雷达图与热力图', fontsize=14, y=0.99)
426 fig.savefig(os.path.join(fig_dir, 'radar_heatmap.png'))
427 plt.close(fig)
```