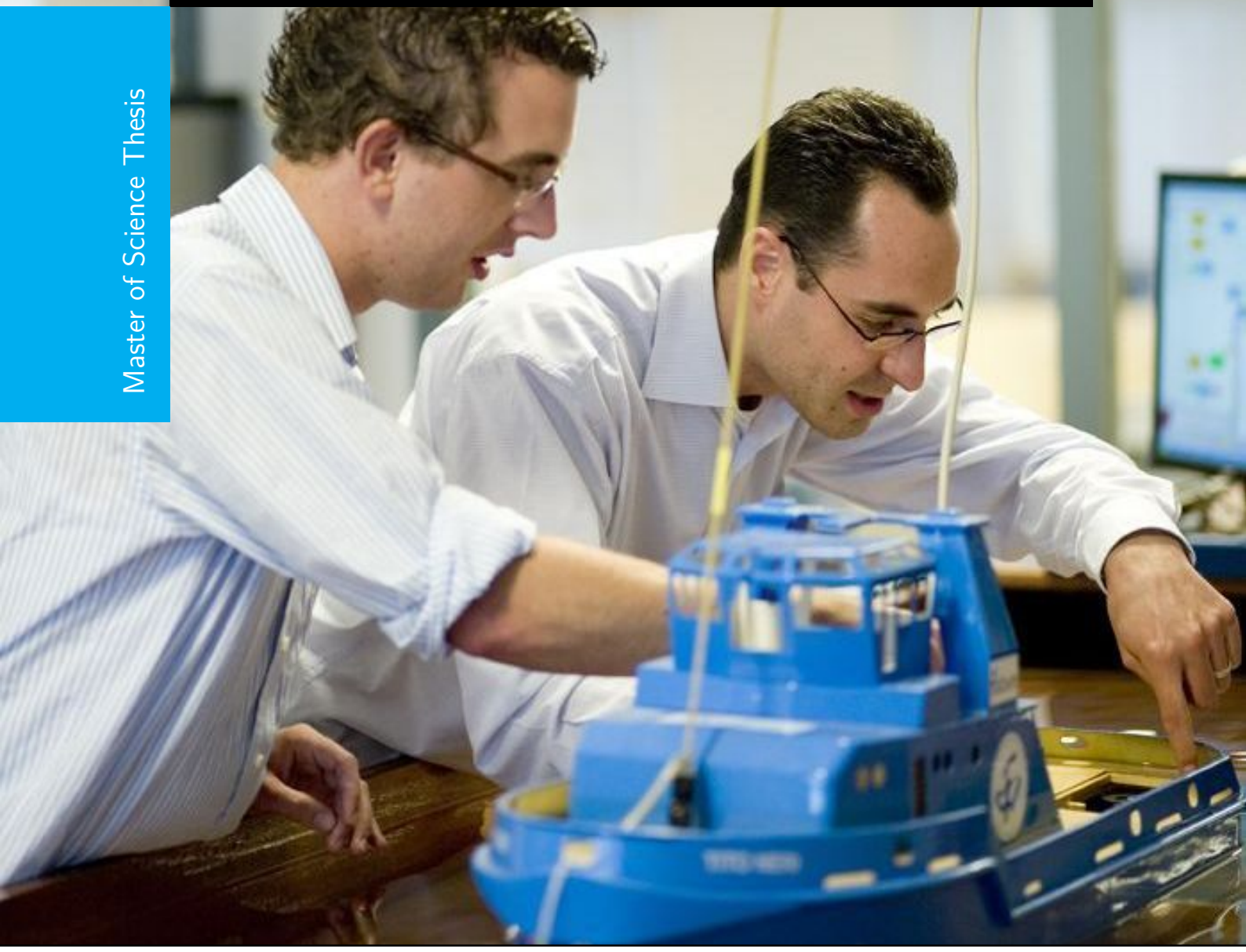
Integrating MPC and RL for Efficient Control of Autonomous Vehicles

Qizhang Dong

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Integrating MPC and RL for Efficient Control of Autonomous Vehicles

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Qizhang Dong

January 9, 2025

Faculty of Mechanical Engineering · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical Engineering for acceptance a thesis entitled

Integrating MPC and RL for Efficient Control of Autonomous Vehicles

by

Qizhang Dong

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: January 9, 2025

Supervisor(s):

prof.dr.ir. Bart de Schutter

dr.ir. Sam Mallick

dr.ir. Gianpietro Battocletti

Reader(s):

dr.ir. Luca Laurenti

Abstract

Autonomous vehicles offer significant potential for improving traffic efficiency and reducing fuel consumption, with Model Predictive Control (MPC) being widely used due to its ability to guarantee constraint satisfaction and safety while providing optimal control performance. However, car models traditionally used in MPC approaches for vehicle control often overlooks discrete dynamics like gear changes, which are critical for optimizing vehicle fuel consumption. Advancements have incorporated these discrete dynamics into MPC, resulting in a hybrid model that considers both continuous and discrete dynamics. The incorporation of the fuel model, along with these discrete dynamics, significantly increases the computational complexity of the MPC problem, making real-time implementation challenging. To address this issue, Reinforcement Learning (RL) can be leveraged to simplify the optimization problem by learning policies that determine key discrete components, such as gear selection. This allows the MPC controller to handle a simpler optimization problem, thereby reducing the computational burden and enabling real-time control. This research aims to propose a new approach to integrate RL and MPC for vehicle control, where RL is used to manage gear transitions and MPC controls the overall vehicle dynamics, offering a computationally efficient solution, while achieving near optimal performance comparable to the conventional MPC approach.

Table of Contents

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, prod.dr.ir. Bart de Schutter, for his guidance and support throughout my research journey. Special thanks go to Sam Mallick and Gianpietro Battocletti, whose help and advice in my daily research and thesis writing have been truly invaluable. Their patience and encouragement made this process much smoother and more enjoyable.

I am also deeply thankful to my classmates and friends, whose support and companionship have meant so much to me during this journey.

Delft, University of Technology — Qizhang Dong
January 9, 2025

Chapter 1

Introduction

Autonomous driving technology has garnered significant attention due to its potential to enhance traffic flow, reduce congestion, and optimize energy consumption. As society increasingly demands safer and more sustainable modes of transportation, autonomous driving solutions are being rigorously researched and developed. Key elements of autonomous driving, including vehicle control, inter-vehicle spacing, and energy consumption, have been extensively studied. Traditional control methods such as Proportional-Integral-Derivative (PID) control and sliding mode control are often employed to ensure stable vehicle performance across varying conditions.

Early research on autonomous driving often utilized simplified vehicle models that focused exclusively on continuous dynamics, overlooking discrete factors such as gear shifts. In these approaches, gear control was typically delegated to low-level controllers, such as rule-based systems, which limited opportunities for optimizing driving performance and fuel efficiency. Recent studies, however, have demonstrated that explicitly incorporating gear dynamics and optimizing gear selection can enhance vehicle driving performance and fuel efficiency, enabling the development of more effective and realistic control strategies.

Model Predictive Control (MPC) is a efficient approach capable of managing both continuous and discrete variables, as discussed earlier. In addition, it can simultaneously address multiple objectives while incorporating system constraints, such as safety limits and physical boundaries, directly into its optimization process. By predicting future system behavior, MPC optimizes control actions in real time, making it particularly effective for complex driving tasks such as car-following and trajectory planning. Its adaptability to changing conditions offers a significant advantage over traditional controllers, ensuring stable and reliable performance in dynamic environments. This versatility makes MPC a powerful tool for achieving efficient and safe autonomous driving in diverse scenarios.

The inclusion of gear dynamics introduces multiple discrete variables into the optimization problem, resulting in a hybrid MPC formulation. When combined with the fuel consumption model, which is nonlinear and non-convex, the problem becomes a mixed-integer nonlinear optimization problem. A key challenge associated with this problem is the resulting computational complexity. In practice, its complexity can grow exponentially with the problem size

due to the interplay of continuous and discrete variables. This growth in complexity leads to prolonged solution times, making it difficult for standard MPC methods to achieve real-time performance. This challenge is particularly critical in dynamic driving environments, where rapid responses are essential for maintaining safety and achieving effective control.

Reinforcement learning (RL) presents a promising solution to mitigate this challenge. RL enables systems to learn a policy through direct interaction with the environment, making it well-suited for complex, real-time decision-making tasks. In this study, we use RL to determine the vehicle's gear choice, replacing most of the discrete components in the hybrid MPC optimization problem. Besides, by addressing the fuel consumption cost within the RL framework, we remove the fuel consumption model in MPC optimization problem, further reducing the problem's complexity. This decomposition significantly reduces the computational complexity of the MPC optimization problem, enabling a real-time implementation while achieving near-optimality compared to the original formulation.

The remainder of this thesis is organized as follows: Chapter 2 reviews the relevant literature and introduces foundational concepts in reinforcement learning and vehicle modeling. Chapter 3 details the implementation of the proposed control methodology. Chapter 4 presents the evaluation of the proposed method, comparing its performance with existing approaches. Chapter 5 discusses broader implications of the results and outlines directions for future research. Finally, Chapter 6 concludes the study.

Chapter 2

Background and Relevant Literature

This chapter provides the background of the study and reviews relevant literature. It includes an overview of the vehicle model used in this work, a review of existing control methods, and an explanation of the reinforcement learning technique utilized in this work.

2-1 Vehicle Model

In this study, we employ a point mass vehicle model that explicitly accounts for gear dynamics[9]. The model used is described here. The dynamics of a forward-moving vehicle are represented by:

$$m\ddot{p}(t) + c\dot{p}(t)^2 + \mu mg = b(j, \dot{p})u(t), \tag{2-1}$$

where $p(t)$ represents the position at time t, c is the aerodynamic drag coefficient, μ is the friction coefficient, $j \in 1, \ldots, 6$ indicates the selected gear, and $b(j, \dot{p})u(t)$ represents the traction force, which is proportional to the normalized throttle input $u(t)$. The parameters used are listed in Table 4-1.

Defining the state vector as position and velocity, i.e., $x = \begin{bmatrix} p & \dot{p} \end{bmatrix}^\top$, the state-space representation can be expressed as:

$$\dot{x} = A(x) + B(j, x)u, \tag{2-2}$$

where:

$$A(x) = \begin{bmatrix} x_2 \\ -(c/m)x_2^2 - \mu g \end{bmatrix}, \quad B(j, x) = \begin{bmatrix} 0 \\ \frac{b(j,x_2)}{m} \end{bmatrix} \tag{2-3}$$

The system dynamics is nonlinear due to the quadratic term in the friction component of $A(x)$, and the dynamics is hybrid owing to the discrete gear selection j, which affects $B(j, x_2)$.

A common approach to managing the nonlinear characteristics of the model is to use a piecewise affine (PWA) approximation. This method divides the nonlinear state space into multiple regions, each represented by an affine segment, simplifying the system's representation. For the quadratic friction term in $A(x)$, we approximate the nonlinear friction using two PWA regions. The decision to use two PWA regions strikes a balance between model complexity and accuracy. Defining the quadratic friction term as $f(x_2) = cx_2^2$, the PWA approximation $\hat{f}$ is represented as:

$$\hat{f}(x_2) = \begin{cases} a_1 x_2 + c_1 & x_2 \leq \alpha \\ a_2 x_2 + c_2 & x_2 > \alpha \end{cases}, \tag{2-4}$$

where $a = \dot{s}_{\text{MAX}}/2$, and $\dot{s}_{\text{MAX}}$ denotes the maximum vehicle velocity. Using this PWA approximation, $A(x)$ becomes:

$$A_{\text{PWA}}(x) = \begin{bmatrix} x_2 \\ -\hat{f}(x_2)/m - \mu g \end{bmatrix}. \tag{2-5}$$

Figure 2-1 illustrates the friction approximation in $A(x)$.

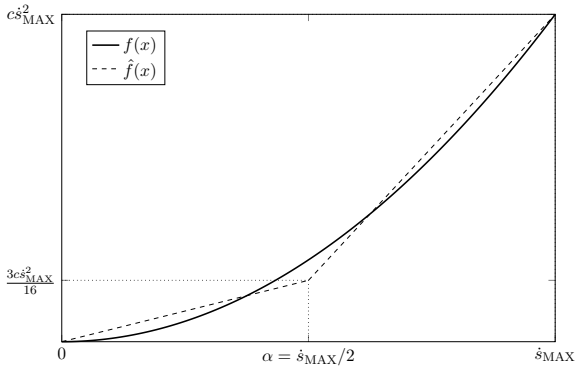
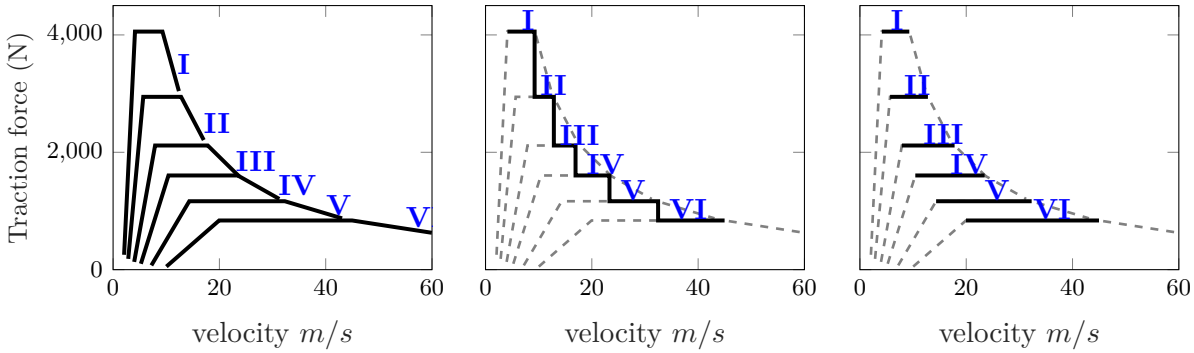


Figure 2-1: PWA friction approximation. The solid line represents the true dynamics and the dashed line represents the piecewise approximation.

Figure 2-2a illustrates the complete gear model, while Table 4-2 presents the maximum traction force for each gear along with the velocity ranges within which the maximum traction remains constant. For the hybrid gear dynamics $B(j, x)$, we consider two approaches for approximation. In the first approach, gears correspond one-to-one with the vehicle velocity, resulting in a PWA approximation for $B(j, x)$ (Figure 2-2b). In the second approach, the gear choice is treated as a discrete decision variable, formulated using a mixed-logical-dynamical (MLD) representation (Figure 2-2c).

2-1-1 PWA gear approximation

To approximate the gear dynamics with a PWA approach, we focus on the regions of the traction curves where traction remains constant. The velocity range is divided into segments, and a specific gear is assigned to each segment, creating a one-to-one mapping between velocity



(a) Full **(b)** PWA **(c)** Discrete input

Figure 2-2: Gear Approximations

segments and gear selection. The mid-point of each velocity range is used as the lower bound for the corresponding PWA region associated with that gear. The PWA gear approximation is then $B_{\text{PWA}}(x) = \begin{bmatrix} 0 & \hat{b}(x_2)/m \end{bmatrix}^\top$ where

$$\hat{b}(x_2) = \begin{cases} b_{1,\text{H}} & v_{1,\text{ L}} \leq x_2 < \frac{v_{2,\text{H}}+v_{2,\text{ L}}}{2} \\ b_{2,\text{H}} & \frac{v_{2,\text{ L}}+v_{2,\text{ L}}}{2} \leq x_2 < \frac{v_{3,\text{H}}+v_{3,\text{ L}}}{2} \\ \vdots & \\ b_{6,\text{H}} & \frac{v_{6,\text{H}}+v_{6,\text{H}}}{2} \leq x_2 < v_{6,\text{H}} \end{cases}, \tag{2-6}$$

and $b_{j,\text{H}}, v_{j,\text{ L}}$, and $v_{j,\text{H}}$, are the maximum traction, minimum velocity, and maximum velocity bounds for gear j, as given in Table 4-2. The PWA gear approximation no longer includes a discrete decision variable j, and is a function only of the state x. Figure 2-2b depicts the PWA gear approximation. Combining $B_{\text{PWA}}(x)$ with $A_{\text{PWA}}(x)$ gives the PWA model

$$\dot{x} = A_{\text{PWA}}(x) + B_{\text{PWA}}(x)u. \tag{2-7}$$

2-1-2 Discrete input gear approximation

The second gear model treats gear selection as a discrete input. Similar to the PWA approach, the traction curves are restricted to regions of constant traction for each gear, and each gear is limited to operate only within its defined region. However, unlike the PWA model, the mapping between velocity and gear is not one-to-one. Figure 2-2c depicts this gear approximation.

To represent gear selection, we introduce six binary variables, $\delta_1, \delta_2, \ldots, \delta_6$, with each variable corresponding to a specific gear. When $\delta_j = 1$, gear j is selected. The following constraint ensures that only one gear is active at any given time:

$$\sum_{j=1}^{6} \delta_j = 1. \tag{2-8}$$

The discrete input approximation is then expressed as

$$b\left(\delta_1, \ldots, \delta_6\right) = \sum_{j=1}^{6} \delta_j b_{j,\mathrm{H}}, \tag{2-9}$$

where $b_{j,\mathrm{H}}$ is the maximum traction for gear j. The B matrix is then approximated with

$$B_{\mathrm{DISC}}\left(\delta_1, \ldots, \delta_6\right) = \begin{bmatrix} 0 \\ b\left(\delta_1, \ldots, \delta_6\right)/m \end{bmatrix}. \tag{2-10}$$

The nonlinearity arising from the multiplication of binary and continuous decision variables in $B_{\mathrm{DISC}}(\delta_1, \ldots, \delta_6)u$ can be reformulated into a linear expression, $B_{\mathrm{MLD}}(\delta_1, \ldots, \delta_6, u)$, by introducing auxiliary variables and mixed-integer linear constraints [1]. To ensure that each gear operates only within its constant traction velocity region, additional mixed-integer constraints are added to encode the necessary logical conditions, such as:

$$\left(\delta_j = 1 \Longrightarrow x_2 \leq v_{j,\mathrm{H}}\right) \Longleftrightarrow x_2 - v_{j,\mathrm{H}} \leq M_{\mathrm{H}}\left(1 - \delta_j\right), \tag{2-11}$$

where $M_{\mathrm{H}} = \max_{x_2}\left(x_2 - v_{1,\mathrm{H}}\right)$, and

$$\left(\delta_j = 1 \Longrightarrow x_2 \geq v_{j,\ \mathrm{L}}\right) \Longleftrightarrow v_{j,\ \mathrm{L}} - x_2 \leq M_{\mathrm{L}}\left(1 - \delta_j\right), \tag{2-12}$$

where $M_{\mathrm{L}} = \max_{x_2}\left(v_{1,\ \mathrm{L}} - x_2, u\right)$.

Converting $A_{\mathrm{PWA}}(x)$ into MLD form [1] and combining it with $B_{\mathrm{MLD}}\left(\delta_1, \ldots, \delta_6\right)$ results in the MLD model

$$\dot{x} = A_{\mathrm{MLD}}(x) + B_{\mathrm{MLD}}\left(\delta_1, \ldots, \delta_6, u\right). \tag{2-13}$$

2-2 Current Methods

In this chapter we review the current methods for the autonomous vehicle control problem, which are categorized by controller type, including linear control, sliding mode control (SMC), H_∞ control, distributed MPC (DMPC), RL, and composite control which combines multiple control methods.

2-2-1 Linear Controller

Shaw *et al.* [14] conducted a comprehensive study on string stability in heterogeneous vehicle strings, utilizing a leader-predecessor following control strategy with a constant spacing policy. Their analysis highlighted the challenges posed by heterogeneity in vehicle dynamics and demonstrated that stability can be achieved without requiring system reconfiguration. This work advanced the understanding of multi-vehicle control strategies under diverse dynamics, with implications for more efficient and reliable transportation systems.

Naus *et al.* [12] proposed a decentralized Cooperative Adaptive Cruise Control (CACC) framework combining feedforward and feedback control mechanisms. The design ensures robust string stability and smooth operation even under communication failures, emphasizing adaptability through frequency-domain analysis. Their work validated the system's effectiveness in mitigating disruptions and maintaining safe inter-vehicle spacing.

2-2-2 Sliding Mode Controller

Wu *et al.* [17] developed a distributed SMC framework for vehicular platoons with nonlinear node dynamics and positive definite topologies. Their approach is notable for addressing topological diversity using a two-phase design: topological sliding surface design and topologically structured reaching law design. The control method ensures stability through local feedback and information from neighboring nodes, leveraging the Lyapunov method for stability proofs. This work highlights the importance of topology matrices in influencing stability and convergence rates, validated through numerical simulations.

2-2-3 H_∞ Controller

Zheng *et al.* [20] analyzed the robustness and developed a scalable distributed H_∞ controller synthesis for homogeneous platoons under undirected communication topologies. This approach utilizes spectral decomposition to decouple the collective dynamics of the platoon into simpler subsystems, significantly reducing computational complexity. Additionally, they highlighted the impact of communication topology on robustness performance, introducing scalable optimization techniques for topology selection.

Similarly, Ploeg *et al.* [13] proposed a frequency-domain-based H_∞ controller to ensure string stability in vehicle platoons with linear dynamics. By solving linear matrix inequalities (LMIs), they derived stabilizing controller parameters to explicitly maintain the string stability condition under various spacing policies and topological constraints.

These methods emphasize robustness to disturbances and scalability to large platoon sizes, aligning well with theoretical guarantees. Their contributions lie in the ability to address external disturbances and system uncertainties while ensuring system stability.

2-2-4 DMPC Controller

He *et al.* [5] proposed a DMPC framework aimed at improving fuel efficiency in vehicle platoons. Their methodology integrates an energy-based control strategy while leveraging vehicle-to-vehicle (V2V) communication to ensure coordination among platoon members. A key aspect of their work is the incorporation of stability and string stability, which are essential for safe and efficient platooning. By embedding fuel consumption optimization directly into the control framework, their approach achieves significant fuel savings while maintaining robust control performance under dynamic conditions.

Zheng *et al.* [19] addressed the challenge of coordinating heterogeneous vehicle platoons under unidirectional communication constraints. They proposed a DMPC algorithm, modeling the communication network as a directed graph to ensure scalability and practicality in real-world

scenarios. The use of a coupled cost function enables effective local optimization while maintaining global coordination objectives, such as desired inter-vehicle spacing and tracking the leader's trajectory. Their work demonstrates the feasibility of achieving robust coordination even with limited communication, highlighting its potential for scalable implementation in autonomous driving systems.

Hu *et al.* [6] developed a distributed economic model predictive control (DEMPC) framework that integrates switching feedback control to optimize fuel consumption in vehicle platoons. Their method combines economic MPC with switching control, introducing a lower bound on the dwell time of the switched system to ensure asymptotic stability and string stability. This approach reduces computational complexity compared to traditional economic MPC methods while achieving robust performance and significant fuel savings. Numerical simulations validate their framework, demonstrating its effectiveness in managing fuel efficiency without compromising system stability.

2-2-5 RL Controller

Li and Cao [8] introduce Communication Proximal Policy Optimization (CommPPO), a reinforcement learning approach for optimizing vehicle platoon control. The method enhances inter-vehicle communication by improving the exchange of state and reward information, addressing issues like redundant data and spurious rewards in traditional multi-agent frameworks. A curriculum learning strategy further improves training efficiency, enabling the model to handle complex scenarios such as vehicle merging and splitting. CommPPO demonstrates improved coordination and operational efficiency, making it a promising method for managing dynamic platoon operations.

Li and Görges [7] propose an ecological adaptive cruise control strategy for vehicles with step-gear transmissions, based on a novel actor-gear-critic reinforcement learning framework. This approach integrates continuous traction force control with discrete gear shift optimization, ensuring safe inter-vehicle distances while improving fuel efficiency. The method is model-free and considers nonlinear vehicle dynamics and transmission efficiency, making it robust to diverse driving scenarios. Simulations validate its adaptability and effectiveness, offering a significant advancement in ecological and adaptive vehicle technologies.

2-2-6 Composite Controllers

Yin *et al.* [18] propose a hierarchical model predictive control (HMPC) strategy for managing platoons of hybrid electric vehicles (HEVs) to improve energy efficiency and safety. The upper-level control employs MPC to optimize longitudinal movements, leveraging V2V communication for maintaining spacing and synchronized speeds. The lower-level Q-Learning controller optimizes energy distribution based on power demands and battery states, with PID control ensuring accurate tracking of vehicle speed. This framework balances computational efficiency and robust performance, making it practical for dynamic real-world platooning scenarios.

Turri *et al.* [15] address fuel efficiency in heavy-duty vehicle platooning by integrating gear management into a layered control architecture. The high-level platoon coordinator generates

fuel-optimal speed trajectories, while the lower-level vehicle controllers ensure adherence to these trajectories. A key contribution is the gear management layer, which uses dynamic programming to optimize gear shifts, minimizing fuel consumption and ensuring smooth operation without disrupting platoon coherence. This approach effectively handles the dynamic impacts of gear shifts, enhancing platoon efficiency and stability under varying road conditions.

2-3 Reinforcement Learning Basics

In this chapter we review the relevant RL theory needed in this research, including key elements of RL, Q-learning, and deep Q-learning (DQN).

2-3-1 Key Elements of RL

RL is a branch of machine learning where an agent learns to make decisions by interacting with its environment. The agent aims to maximize cumulative rewards over time by exploring various actions to understand their outcomes and exploiting the most rewarding strategies it discovers. Understanding RL requires familiarity with several key concepts:

Agent and Environment RL operates through the interaction between two key entities: the agent and the environment. The definitions are listed below:

- **Agent**: The decision-maker that interacts with the environment.
- **Environment**: The system with which the agent interacts, comprising everything outside the agent.

States, Actions, and Rewards These elements define the interaction between the agent and the environment, forming the basis for the agent's learning process:

- **State (s)**: Representation of the environment at a particular time.
- **Action (a)**: A decision or move taken by the agent that affects the environment.
- **Reward (r)**: A scalar feedback signal received by the agent after taking an action in a state.

Policy (π) A policy is a strategy used by the agent to decide which action to take in a given state. It can be deterministic or stochastic:

- **Deterministic Policy**: Maps states to actions directly, $\pi(s) = a$.
- **Stochastic Policy**: Maps states to probabilities of actions, $\pi(a|s) = P(a|s)$.

The goal of the agent is to find a policy to maximize the total reward over time:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \pi\right] \tag{2-14}$$

where γ is the discount factor controling the importance of future rewards.

Value Functions Value functions estimate the expected return (cumulative reward) of states or state-action pairs under a specific policy:

- **State Value Function** $V^\pi(s)$: The expected return when starting in state s and following policy π:

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right] \tag{2-15}$$

- **Action Value Function** $Q^\pi(s,a)$: The expected return when starting in state s, taking action a, and thereafter following policy π:

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \tag{2-16}$$

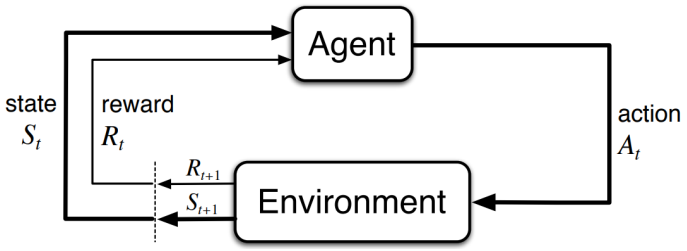


Figure 2-3: Markov Decision Process

Markov Decision Process (MDP) Represented schematically in Figure 2-3, an MDP is a mathematical framework that provides a formalization for the RL problem. It is defined by:

- A set of states S that satisfy the Markov property, that is, the future state of the process depends only on the present state.
- A set of actions, A
- A transition function, $P(s'|s,a)$, which defines the probability of moving from state s to state s' under action a.
- A reward function, $R(s,a)$, which specifies the immediate reward received after transitioning from state s to state s' due to action a.

2-3-2 Q-Learning

Q-learning [16] is a model-free reinforcement learning algorithm that learns the optimal action-value function $Q^*(s, a)$ through direct interaction with the environment, without requiring prior knowledge of its dynamics. It is an off-policy method, meaning the target Q-value is determined by taking the maximum action-value, which differs from the actual behavior policy followed by the agent during exploration.

The Q-value is updated iteratively using the rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right], \tag{2-17}$$

where α is the learning rate and γ is the discount factor.

This approach ensures convergence to the optimal policy under appropriate conditions, even while exploring the environment. It is well-suited for discrete state and action spaces and serves as the foundation for deep Q-networks.

2-3-3 Deep Q-Networks

Deep Q-Networks (DQN), proposed by Mnih *et al.* [10, 11], combine Q-learning with deep neural networks to handle high-dimensional state spaces. Below is a detailed introduction to its key components.

Q-Network In DQN, the Q-function is approximated using a neural network, referred to as the Q-network. The Q-network can be structured in two ways, depending on the nature of the action space:

1. For continuous or discrete action spaces, the input to the network is a state-action pair, and the output is a scalar value representing the Q-value for that pair.

2. For discrete action spaces, the input is a state, and the outputs are Q-values for all possible actions.

The policy is implicitly derived from the Q-network by selecting the action with the highest Q-value for a given state:

$$\pi(s) = \arg\max_a Q(s, a). \tag{2-18}$$

This setup allows the Q-network to approximate the optimal Q-function, enabling the agent to make decisions effectively in complex environments.

Note that when the action space is continuous, performing the arg max operation becomes computationally expensive and challenging.

Target Network The target network is a separate neural network used to compute stable target Q-values during training. The target value is defined as:

$$y_t = r_t + \gamma Q_{\text{target}}(s_{t+1}, \pi(s_{t+1}); \theta_{target}), \tag{2-19}$$

where Q_{target} is the Q-value predicted by the target network, and θ_{target} represents the parameters of the target network.

To train the Q-network, the following loss function is minimized:

$$\mathcal{L} = \mathbb{E}\left[(y_t - Q(s_t, a_t; \theta))^2\right], \tag{2-20}$$

where θ represents the parameters of the Q-network.

During training, the target network parameters are fixed to stabilize the targets. After a certain number of updates, the target network parameters are synchronized with the Q-network parameters:

$$\theta_{\text{target}} \leftarrow \theta. \tag{2-21}$$

This approach reduces instability caused by constantly changing targets.

Exploration To balance exploitation (choosing actions that maximize Q-values) and exploration (trying new actions), DQN uses strategies such as:

- ϵ**-greedy**: The agent selects a random action with probability ϵ and the action that maximizes the Q-value with probability $1-\epsilon$. The value of ϵ typically decays over time.
- **Boltzmann exploration**: The agent selects an action based on a probability distribution derived from Q-values, ensuring a higher probability for actions with higher Q-values.

Experience Replay DQN employs a replay buffer to store experiences (s_t, a_t, r_t, s_{t+1}) gathered during training. For each update, a random batch of experiences is sampled from the buffer. This mechanism provides the following advantages:

- **Reduced Sample Correlation**: Random sampling breaks the temporal correlation between consecutive experiences, leading to more stable training.
- **Increased Data Efficiency**: By reusing past experiences, the agent learns more effectively from the data, reducing the need for continuous environment interaction.
- **Batch Learning**: Using batches for training leverages the computational efficiency of modern deep learning frameworks, enabling faster and more stable optimization.

2-4 Summary

In this chapter, we provided a comprehensive review on background and relevant literature essential for understanding our work on autonomous vehicle control. This chapter is organized into three main sections.

First, we presented the vehicle model employed in our research, outlining its nonlinear and hybrid dynamics and how these dynamics are approximated using PWA and MLD representations. The use of these approximations aims to simplify the complex vehicle dynamics while retaining a reasonable level of accuracy, thus facilitating effective control approaches.

Second, we reviewed current methods and approaches for autonomous vehicle control, categorized by control approach type. This included discussions on various controllers such as linear, sliding mode, H_∞, DMPC, reinforcement learning, and composite controllers that integrate multiple control methods. Each of these approaches was explored in the context of their applications to vehicle platooning and energy management, highlighting their advantages, limitations, and scenarios in which they are most effective.

Finally, we provided an overview of fundamental reinforcement learning concepts relevant to our work, including key elements of RL and DQN. The discussion also covered practical aspects of RL training such as exploration strategies and experience replay, which are critical for developing effective RL-based control strategies.

In the next chapter, we will address the problem formulation in detail, analyze the challenge associated with the traditional standard MPC approach, and then then propose our integrated MPC-RL solution, which aims to overcome this challenge by leveraging reinforcement learning for efficient gear selection in hybrid vehicle dynamics.

Chapter 3

Proposed MPC-RL Approach

In this chapter, we present the control methodology proposed in this research. The chapter is organized into sections detailing the problem formulation, the design of the controller, and the training framework for the RL agent.

3-1 Problem Formulation

This work addresses the control of a single vehicle tasked with following a given reference trajectory, denoted as ξ. The objective is to minimize a cost that considers tracking accuracy and fuel consumption, while satisfying constraints related to vehicle dynamics and safety requirements. This strategy is particularly suited for highway scenarios, including applications such as car-following and vehicle platooning.

3-1-1 Vehicle States and Inputs

The vehicle's states, denoted as x, include position (x_1) and velocity (x_2). The control inputs comprise the throttle input (u) and gear choice (j).

3-1-2 Constraints

To ensure safety and comfort, the states and inputs of the vehicle are subject to the following constraints:

Velocity and Acceleration Constraints The vehicle's velocity and acceleration are constrained as:

$$\begin{aligned} &v_{\text{MIN}} \leq x_2(k) \leq v_{\text{MAX}}, \\ &a_{\text{MIN}} T \leq x_2(k+1) - x_2(k) \leq a_{\text{MAX}} T, \end{aligned} \tag{3-1}$$

where v_{MIN}, v_{MAX}, a_{MIN}, and a_{MAX} are the bounds on velocity and acceleration. The minimum velocity constraint ensures the validity of constant traction approximations and applies to highway scenarios like car-following or platooning, where forward motion is continuous.

Throttle Input Constraints The normalized throttle input is constrained as:

$$u_{\text{MIN}} \leq u(k) \leq u_{\text{MAX}} \tag{3-2}$$

3-1-3 Performance Cost

The performance cost, J_{total}, is defined as a weighted sum of two components:

$$J_{\text{total}} = w_1 \cdot J_{\text{tracking}} + w_2 \cdot J_{\text{fuel}} \tag{3-3}$$

The weights w_1 and w_2 balance the importance of tracking accuracy (J_{tracking}) and fuel efficiency (J_{fuel}) in the overall cost.

J_{tracking} quantifies tracking accuracy:

$$J_{\text{tracking}} = \|x(k) - \xi(k)\|_{Q_x} \tag{3-4}$$

where $\|\cdot\|_{Q_x}$ denotes the 2-norm weighted by the matrix Q_x, which emphasizes the importance of each state variable.

J_{fuel} represents the fuel consumption cost, calculated using a polynomial fuel consumption model that depends on the vehicle velocity [5]. Specifically, the fuel consumption model is expressed as:

$$J_{\text{fuel}} = Q_f \left(\sum_{m=0}^{3} b_m x_2^m(k) + \frac{x_2(k+1) - x_2(k)}{T} \sum_{l=0}^{2} c_l x_2^l(k) \right) \tag{3-5}$$

Here Q_f is a scaling factor for fuel cost, T is the sampling time, b_m and c_l are coefficients derived from experimental data to model fuel consumption. Note that during braking when $u \leq 0$, it is assumed that no fuel is consumed due to the energy recycle technique.

3-1-4 Challenges of Standard MPC Approach

Currently, a standard MPC approach (see later Section 4-3-1) can solve the optimization problem using mixed-integer nonlinear optimization [9]. However, the inclusion of integer variables and the fuel consumption model make the problem non-convex and hybrid, thereby significantly increasing the complexity of the problem. This heightened complexity results in long solution times, making real-time implementation infeasible for practical applications.

3-1-5 Objective of This Study

This study aims to develop a control strategy that simplifies the standard hybrid MPC problem, addresses its computational challenges, and enables real-time implementation while maintaining performance comparable to the original formulation.

3-2 Controller Design

To overcome the challenges outlined in section 3-1-4, we employ an RL-based approach to handle most of the discrete components, such as gear selection, and to directly manage the fuel consumption cost. This allows the MPC to focus solely on computing the continuous throttle input and optimizing the tracking cost, significantly simplifying the optimization problem. As a result, the computational time can be significantly reduced, enabling real-time implementation.

Illustrated as figure 3-1, in this study we propose that an RL agent determines the gear selection, thereby simplifying the original mixed-integer optimization problem. Although some discrete variables remain, the reduction achieved enables a much faster and more efficient MPC solution for the remaining control problem.

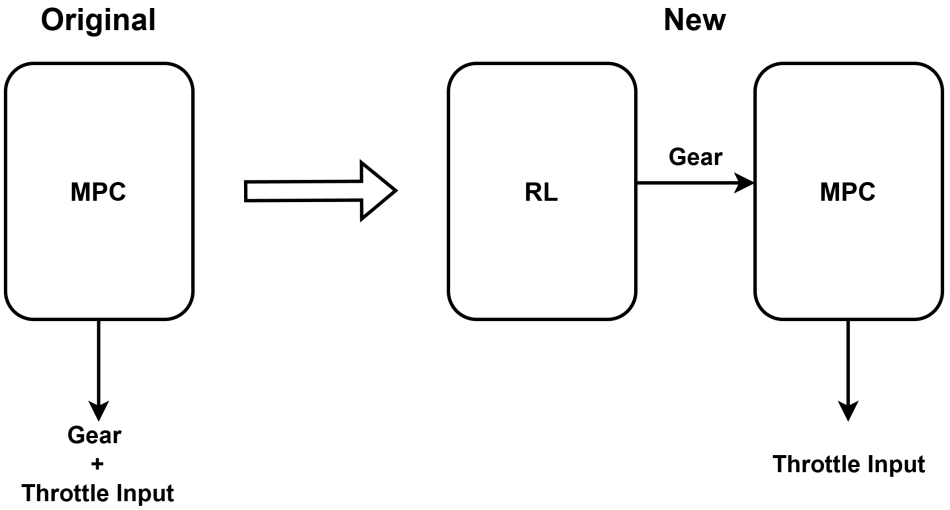


Figure 3-1: Controller Design. The left panel depicts the standard MPC approach, where the controller determines both the discrete gear selection and continuous throttle input. The right panel illustrates the proposed MPC-RL approach, where MPC determines the throttle input and RL handles the gear selection.

The integration of RL for discrete decision-making (gear selection) and MPC for continuous control (throttle management) leverages the strengths of each approach: RL brings adaptability and efficiency to gear selection, while MPC ensures optimal and safe control actions. The subsequent sections will detail the design of the gear controller, the vehicle throttle controller, and the structure of the overall control loop.

3-2-1 Vehicle Throttle MPC Controller Design

In the MPC controller of our approach, the discrete input gear approximation, as described in Section 2-1-2, is utilized to represent the gear model. The states and input constraints, as outlined in Section 3-1, define the operational limits of the system. The MPC optimization objective includes both tracking and control costs, evaluated using norm penalties. The

optimization problem is formulated as follows:

$$\begin{aligned} J(x, \mathbf{j}) = \min_{\mathbf{u},\mathbf{x}} & \underbrace{\sum_{k=0}^{N+1} \|x(k) - \xi(k)\|_{Q_x}}_{\text{Tracking Cost}} \\ & + \underbrace{\sum_{k=0}^{N} \|u(k)\|_{Q_u}}_{\text{Control Cost}} \\ & \text{s.t. (3-1) - (3-2)} \\ & \text{vehicle model (2-13)} \\ & x(0) = x \end{aligned} \tag{3-6}$$

where the decision variables $\mathbf{u}$ and $\mathbf{x}$ represent the control and state trajectories over the prediction horizon. Notably, the gear selection sequence $\mathbf{j}$ is not included as a decision variable in the optimization problem but is instead treated as an input. It is determined by the RL controller and fixed for the vehicle model during the optimization process.

At each time step, the MPC generates a control sequence $\mathbf{u} = (u(0), \cdots, u(N-1))$ by solving the optimization problem defined over the prediction horizon N, adhering to the given constraints and vehicle model. The initial element of this control sequence, $u(0)$, is then applied to the system, and the optimization process is repeated at the next time step in a receding horizon manner.

3-2-2 Vehicle Gear RL Controller Design

State and Action Space

State Space At each time step, the MPC controller provides the RL controller with the predicted states and control sequences over a horizon of length N. Consequently, the raw outputs from the MPC controller consists of the vehicle states—position (x_1) and velocity (x_2)—as well as the control inputs—throttle input (u) and gear choice (j). Denoted as $\mathbf{s}$, the predicted state and control sequences are structured as follows:

$$\mathbf{s}(k) = \begin{bmatrix} x_1(k) & x_1(k+1) & \dots & x_1(k+N-1) \\ x_2(k) & x_2(k+1) & \dots & x_2(k+N-1) \\ u(k) & u(k+1) & \dots & u(k+N-1) \\ j(k) & j(k+1) & \dots & j(k+N-1) \end{bmatrix} \tag{3-7}$$

Note that the original length of the predicted state sequence is $N+1$, while the control input sequence has a length of N. To maintain consistency between the state and control sequences, only the first N terms of the predicted state sequence are used.

The RL controller does not directly utilize the raw outputs from the MPC controller. Instead, pre-processing is applied to the state sequences to make them more suitable for decision-making. Specifically, the absolute position of the vehicle does not provide meaningful information for the RL controller. Therefore, the position tracking error between the current

position x_{1_k} and the reference position r_k is computed and used as input. This is expressed as:

$$e(k)) = x_1(k) - \xi(k) \tag{3-8}$$

For the velocity state x_2, normalization is applied to scale the values into a standardized range. This ensures consistent scaling of input features, which enhances the stability and efficiency of the RL training process. The normalized velocity is calculated as:

$$v^{\text{norm}}(k) = \frac{x_2(k) - v_{\text{MIN}}}{v_{\text{MAX}} - v_{\text{MIN}}} \tag{3-9}$$

As a result, the states utilized by the RL controller is structured as follows:

$$\mathbf{s}'(k) = \begin{bmatrix} e(k) & e_(k+1) & \dots & e_(k+N-1) \\ v^{\text{norm}}(k) & v^{\text{norm}}(k+1) & \dots & v^{\text{norm}}(k+N-1) \\ u(k) & u(k+1) & \dots & u(k+N-1) \\ j(k) & j(k+1) & \dots & j(k+N-1) \end{bmatrix} \tag{3-10}$$

Action Space The action set consists of three possible gear operations: up-shift, down-shift, and no shift. This formulation simplifies the decision-making process for the RL controller by focusing on relative gear changes rather than explicit gear choices. When considering a prediction horizon N, the action space expands to include all possible sequences of actions over the horizon, represented as the Cartesian product

$$\mathcal{A}^N = \underbrace{\mathcal{A} \times \mathcal{A} \times \cdots \times \mathcal{A}}_{N \text{ times}} \tag{3-11}$$

where

$$\mathcal{A} = \{\text{upshift}, \text{downshift}, \text{no shift}\} \tag{3-12}$$

Using gear operations instead of explicit gear levels offers several advantages. First, it reduces the size of the action space, which is particularly beneficial for reinforcement learning, as smaller action spaces typically lead to faster convergence and more stable learning. Second, this approach inherently accounts for sequential dependencies in gear changes, ensuring that transitions occur only between adjacent gears.

Reward Function

The reward function should be designed to prioritize ensuring that the gear sequence selected by the RL controller is feasible for the MPC controller. Once feasibility is guaranteed, it should further guide the agent to select gears that minimize fuel consumption cost and tracking error. During training, there are two scenarios where the selected gear choices may result in infeasibility:

1. Exceeding Gear Limits: The RL controller selects a gear shift that exceeds the defined limits of the gear range. For instance, it suggests an upshift when the current gear is already at the maximum (e.g., gear 6) or a downshift when the current gear is at the minimum (e.g., gear 1).

2. Velocity-Gear Mismatch: While the gear choice stays within the allowable range (e.g., between 1 and 6), it may still be infeasible if the MPC controller cannot adjust the vehicle dynamics to achieve a velocity compatible with the selected gear (Figure 2-2c). This constraint forces the MPC to keep the velocity within the range corresponding to the chosen gear, and any mismatch can result in an unsolvable optimization problem.

Therefore, a penalty for infeasibility must be incorporated into the reward function and assigned the highest priority. Feasibility is foundational, as it ensures the solvability of the MPC problem and the stability of the training process. Without feasible gear choices, the MPC cannot operate effectively, undermining the entire control framework.

Since gear selection can impact fuel consumption, and the RL controller is responsible for managing it, the fuel consumption cost is included in the reward function. Additionally, while gear selection also affects tracking performance, its influence is less pronounced compared to the MPC controller. Therefore, the tracking cost is included in the reward function but assigned a relatively smaller weight.

This weighting strategy ensures that the RL controller focuses primarily on feasibility, while still contributing to tracking and fuel optimization. By maintaining this balance, the RL controller supports the MPC, ensuring stable training and desirable performance of the hybrid control system.

RL Algorithm Selection

Given the discrete nature of the action space, DQN was selected for its efficiency, stability, and suitability for this application. DQN is specifically designed for discrete action spaces by learning a Q-value function $Q(s, a)$, which estimates the expected cumulative reward for each action given a state. This makes DQN an ideal choice for the gear shift problem, where the set of actions is fixed and finite. Furthermore, DQN includes mechanisms such as experience replay and target networks that enhance sample efficiency and stabilize training. Experience replay allows DQN to reuse past transitions, improving training efficiency and stability by breaking the correlation between consecutive samples. Target networks prevent large fluctuations in Q-value updates, ensuring stable training.

Network Architecture

Since the input states $\mathbf{s}^{'}$ of the RL controller are sequences over a horizon of N, the RL controller must provide a corresponding sequence of gear choices $\mathbf{j}$, one for each predicted state. The predicted states from the MPC controller exhibit strong temporal relationships that inherently capture information about the vehicle's dynamics. Therefore, it is crucial for the RL controller to account for these temporal dependencies to ensure more accurate and effective gear selection.

To model these temporal dependencies effectively, a recurrent neural network (RNN) is integrated into the network architecture of the DQN. RNNs are well-suited for sequential data, allowing the RL controller to capture dependencies across time steps and make more informed gear selection decisions. In this setup, the RNN follows a many-to-many architecture, meaning it takes a sequence of inputs (predicted states over multiple time steps) and generates a

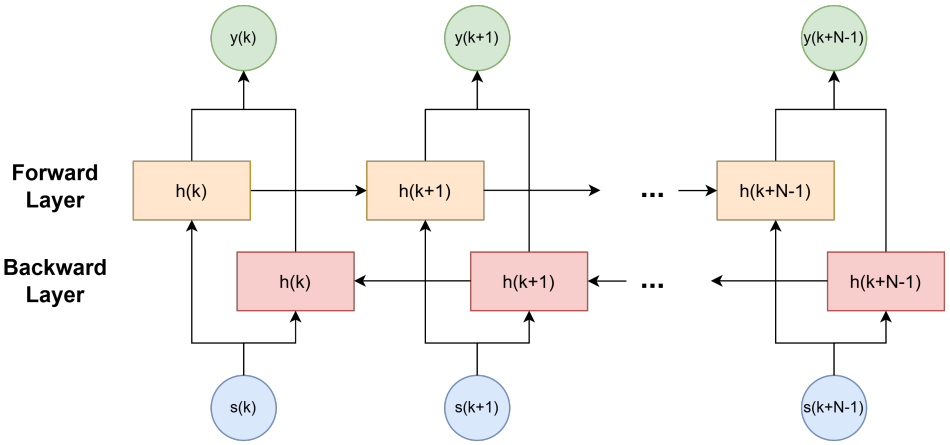


Figure 3-2: BiRNN Diagram

corresponding sequence of outputs. This ensures that the RNN provides a gear shift decision for every time step in the sequence of predicted states.

At each time step k, the RNN takes the current input $s(k)$ and the hidden state $h(k-1)$ from the previous step to compute the current hidden state $h(k)$. The hidden state is then used to produce the output $y(k)$, which corresponds to the gear shift at that time step. This process is repeated across all N time steps, allowing the network to model temporal dependencies in the sequence of predicted states effectively.

Additionally, the gear selection problem in this context requires that each gear choice not only depends on the predicted states preceding it but also on the predicted states following it within the horizon N. To address this, a bidirectional RNN (BiRNN) is used, which processes the sequence in both forward and backward directions. Figure 3-2 illustrates the BiRNN architecture, where the network's outputs are derived by combining the hidden states from both directions. This architecture allows the RL controller to simultaneously consider both earlier and later predicted states within the horizon, leading to better-informed gear choices that can potentially enhance their feasibility and optimality. By capturing temporal patterns in both directions, the BiRNN enables the RL controller to optimize gear selection for both immediate feasibility and overall system performance objectives, such as tracking accuracy and fuel efficiency. This combination of DQN with BiRNN provides a efficient framework for handling the complexities of sequential decision-making in hybrid control systems.

As shown in Figure 3-3, the network architecture in our approach takes an input vector of dimension 4 (3-10), which is fed into the Bi-RNN layer. The outputs from the Bi-RNN layer (represented by black arrows) are processed by a fully connected layer, which maps them to an output size of 3 corresponding to the action set defined in Equation (3-12). Simultaneously, the hidden states are updated through time, as indicated by the red arrows. The specific sizes of the hidden layers will be introduced in the next chapter. The network processes a sequence of predicted states at each time step and outputs a corresponding gear choice for each predicted state, operating in a many-to-many configuration.

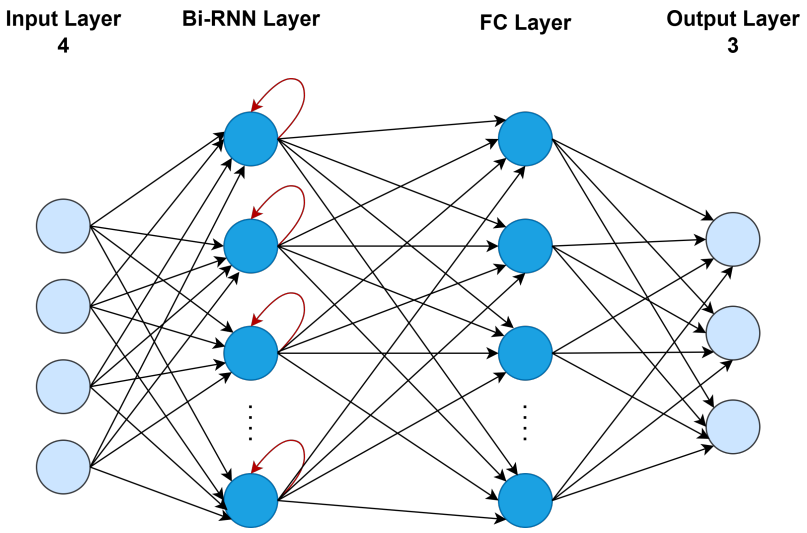


Figure 3-3: Network architecture. The Bi-RNN layer includes propagation of hidden states through time, represented by the red arrows.

3-2-3 Control Loop

Figure 3-4 illustrates the control loop of the proposed method. At each time step k, the predicted sequences $\mathbf{s}(k-1)$ from the MPC controller at the previous time step is preprocessed by the preprocessing block to generate the input $\mathbf{s}'(k-1)$ for the RL controller. The RL controller takes $\mathbf{s}'(k-1)$ as input and outputs a sequence of gear shift decisions $\Delta\mathbf{j}(k)$.

The postprocessing block converts $\Delta\mathbf{j}(k)$ into a specific gear sequence $\mathbf{j}(k)$, which serves as an input to the MPC controller. At the same time, the MPC controller receives the current observation $x(k)$ from the environment along with the reference input. Using $\mathbf{j}(k)$ and $x(k)$, the MPC controller computes the optimal throttle input $\mathbf{u}(k)$.

Finally, the first elements of the predicted control sequences, $u(k)$ and $j(k)$, are applied to the plant. The updated prediction sequences $\mathbf{s}(k)$ from the MPC controller is then used as input for the next time step, ensuring the closed-loop control operation.

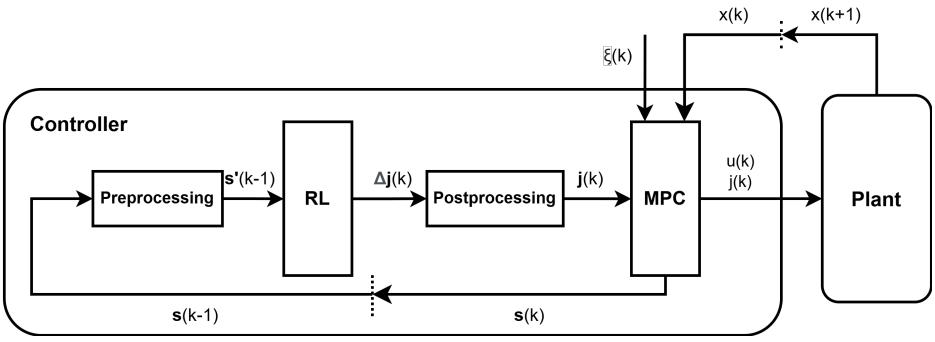


Figure 3-4: Control loop at time step k, illustrating the interaction between the RL and MPC controllers for gear shift decision-making and throttle optimization.

3-3 RL Training Scheme

3-3-1 Environment Setup

We developed a simulation environment that simulates vehicle movement based on its true dynamics. It is important to note that the vehicle model used here is the 'true' nonlinear model (2-1), rather than the approximated model used by the MPC controller.

To simulate multiple scenarios, we randomized the reference trajectory for each episode, which helps improve the robustness and adaptability of the RL agent by training it with diverse driving conditions. The total length of the reference trajectory is N_{T} time steps. The initial position of the reference trajectory is given by:

$$\xi_1(0) = \xi_{1,0} \tag{3-13}$$

and the initial reference velocity is randomized as

$$\xi_2(0) = \mathrm{rand}(\xi_{2,\min}, \xi_{2,\max}) \tag{3-14}$$

We divided the velocity profile into N_{R} piecewise regions, with switching moments determined by fixed time steps combined with random offset values. These offset values are denoted as:

$$o_i = \mathrm{rand}(o_{\min}, o_{\max}), \quad i = 1, 2, \ldots, N_{\mathrm{R}} - 1 \tag{3-15}$$

In the first and last piecewise regions, the velocity remains constant. For the remaining regions, the velocity changes at a certain rate in each region. The rates of change are also random, expressed as:

$$\lambda_i = \mathrm{rand}(\lambda_{\min}, \lambda_{\max}), \quad i = 1, 2, \ldots, N_{\mathrm{R}} - 2 \tag{3-16}$$

Thus, the reference velocity can be described as the following piecewise function:

$$\xi_2(k+1) = \begin{cases} \xi_2(k), & k \in [0, t_1 + o_1) \\ \max(\min(v_{\max}, \xi_2(k) + \lambda_1), v_{\min}), & k \in [t_1 + o_1, t_2 + o_2) \\ \vdots & \\ \xi_2(k), & k \in [t_{N_{\mathrm{R}}-1} + o_{N_{\mathrm{R}}-1}, N_{\mathrm{T}}) \end{cases} \tag{3-17}$$

The reference position is then given by

$$\xi_1(k+1) = \xi_1(k) + T \cdot \xi_2(k), \quad k \in [0, N_{\mathrm{T}}) \tag{3-18}$$

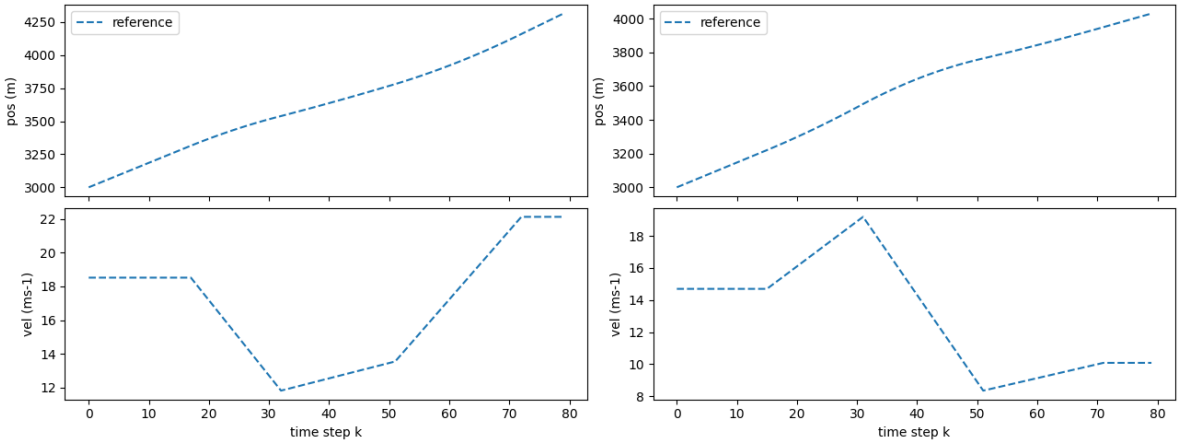
Figure 3-5 shows two examples of randomly generated reference trajectories used for training, which highlights the variations in reference trajectories used to train the RL agent under different conditions.

The initial position of the vehicle is randomized as:

$$x_1(0) = \mathrm{rand}(x_{1,\min}, x_{1,\max}) \tag{3-19}$$

and the initial vehicle velocity is given by:

$$x_2(0) = \mathrm{rand}(x_{2,\min}, x_{2,\max}) \tag{3-20}$$



(a) Example 1 **(b)** Example 2

Figure 3-5: Two Examples of Reference Trajectories

3-3-2 Training Procedure

The training is performed episode by episode, where each episode contains N_{E} time steps. The detailed training loop within an episode will be introduced in the following.

Step 0: Initialization At the beginning of each training episode, the reference trajectory and vehicle states are initialized as outlined in Section 3-3-1. At the first time step, the MPC controller is tasked with solving the optimization problem using the gear sequence provided by the RL controller. However, the RL controller relies on the predicted states generated by the MPC, which are unavailable at the initial time step due to the lack of prior predictions. To address this issue, the initial gear is determined using the PWA approximation based on the initial velocity. This initial gear, denoted as $j(0)$, is then uniformly applied across the entire prediction horizon. The gear sequence for time step 0 is defined as:

$$\mathbf{j}(0) = \begin{bmatrix} j(0) & j(0) & \cdots & j(0) \end{bmatrix}_{1\times N} \tag{3-21}$$

This gear sequence is fed into the MPC controller to solve the optimization problem $J(x, \mathbf{j})$, resulting in the predicted state sequence $\hat{\mathbf{x}}(0)$ and control sequence $\hat{\mathbf{u}}(0)$. Subsequently, the environment is stepped forward using the first element of the control sequence, yielding the new observation $x(1)$.

While the initial gear sequence may not be optimal, it facilitates the MPC controller generates the necessary predicted sequences for the RL controller. This facilitates a smooth start to the training process and ensures consistent interaction between the MPC and RL controllers from the beginning.

Step 1: Gear Selection With the predicted states and control sequence at time step $k-1$, the RL controller determines the gear sequence for time step k. The network outputs Q-values

for the three possible gear shifts, and the gear shift is selected based on the index with the highest Q-value. The selected gear shift as Δj is calculated as:

$$\Delta j(k) = \arg \max_{i \in \{0,1,2\}} Q(s^{'}(k-1), i) - 1 \tag{3-22}$$

where $Q_i(s^{'}(k-1))$ represents the Q-value corresponding to gear shift option i, as predicted by the Q-network. The selected gear shift Δj is determined by finding the index i that yields the maximum Q-value and then subtracting 1 to map it to the appropriate gear shift value. The gear shift is defined as follows:

$$\Delta j = \begin{cases} -1, & \text{downshift} \\ 0, & \text{no shift} \\ 1, & \text{upshift} \end{cases} \tag{3-23}$$

This formulation ensures that the selected gear shift accurately reflects the intended action: downshift, no shift, or upshift, providing a clear and consistent mapping between the Q-network outputs and the actual gear control decisions. The gear shift sequence $\Delta\hat{\mathbf{j}}$ at time step k is defined as:

$$\Delta\hat{\mathbf{j}}(k) = \begin{bmatrix} \Delta\hat{j}(k) & \Delta\hat{j}(k+1) & \cdots & \Delta\hat{j}(k+N-1) \end{bmatrix} \tag{3-24}$$

Then, we calculated the explicit gear sequence $\hat{\mathbf{j}}(k)$ using the gear shift sequence $\Delta\hat{\mathbf{j}}(k)$. Within the prediction horizon, each future gear is determined sequentially based on the previous gear value and the corresponding gear shift at that time step. The predicted gear sequence is then expressed as

$$\hat{\mathbf{j}}(k) = \begin{bmatrix} \hat{j}(k) & \hat{j}(k+1) & \cdots & \hat{j}(k+N-1) \end{bmatrix} \tag{3-25}$$

The individual gears in the sequence are calculated iteratively as follows:

$$\hat{j}(k+i) = \begin{cases} j(k-1) + \Delta\hat{j}(k+i), & i = 0 \\ \hat{j}(k+i-1) + \Delta\hat{j}(k+i), & i = 1, \cdots, N-1 \end{cases} \tag{3-26}$$

To incorporate exploration in the decision-making process, an ϵ-greedy exploration mechanism is applied during training. The exploration probability ϵ decays over time to balance exploration and exploitation. The exploration decay is defined as:

$$\epsilon = \epsilon_{\text{end}} + (\epsilon_{\text{start}} - \epsilon_{\text{end}}) \cdot \exp\left(-\frac{n_{\text{steps}}}{\epsilon_{\text{decay}}}\right) \tag{3-27}$$

where:

- ϵ_{start}: Initial exploration rate.
- ϵ_{end}: Minimum exploration rate.
- n_{steps}: Number of training steps completed.

- ϵ_{decay}: Decay factor controlling the speed of decay.

At each time step, a random number $\zeta \in [0, 1]$ is sampled. If $\zeta < \epsilon$, a random gear shift sequence is selected for exploration. Otherwise, the gear shift is selected greedily based on the Q-values:

$$\Delta j(k) = \begin{cases} \text{random gear shift}, & \zeta < \epsilon \\ \arg\max_{i \in \{0,1,2\}} Q_i(s^{'}(k-1)) - 1, & \zeta \geq \epsilon \end{cases} \tag{3-28}$$

This exploration mechanism ensures a trade-off between exploring new actions and exploiting the learned policy, enabling the RL controller to efficiently learn gear control decisions.

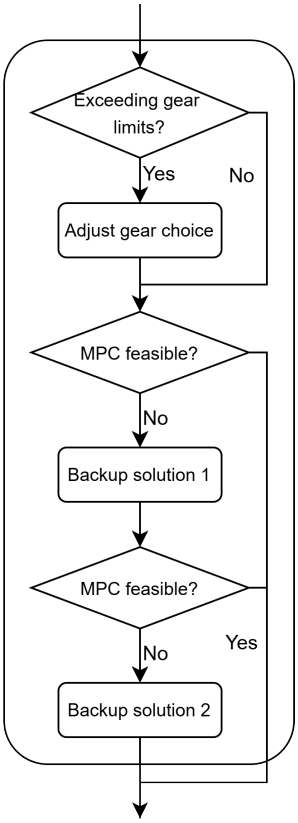


Figure 3-6: Detailed Flowchart of Step 2

Step 2: Feasibility Check & Solving MPC Problem After calculating the predicted gear sequence, we check if the sequence is feasible for the vehicle dynamics. As discussed in 3-2-2, the gear sequence may be infeasible due to exceeding gear limits or velocity-gear mismatch, causing the failure of solving the MPC problem.

We first check if some gear choices in the sequence exceed the gear limits. Since the vehicle has 6 gears, we set the gear choice to 6 when it exceeds 6, and to 1 when its lower than 1.

Specifically, it is expressed as:

$$\hat{j}(k+i) = \begin{cases} 6, & \hat{j}(k+i) > 6 \\ 1, & \hat{j}(k+i) < 1 \\ \hat{j}(k+i), & \text{else} \end{cases} \qquad i \in [0, N-1] \tag{3-29}$$

The adjusted gear sequence is provided to the MPC solver to solve the MPC problem. If the solver returns an infeasibility result, we revert to the feasible gear sequence from previous time step. Specifically, we use the gear sequence at time step $k-1$ to replace the infeasible gear sequence at time step k.

At time step k, the previously predicted gear sequence from time step $k-1$, denoted as $\hat{\mathbf{j}}_{\text{old}}(k-1)$, is expressed as:

$$\hat{\mathbf{j}}_{\text{old}}(k-1) = \begin{bmatrix} \hat{j}_{\text{old}}(k-1) & \hat{j}_{\text{old}}(k) & \cdots & \hat{j}_{\text{old}}(k+N-2) \end{bmatrix} \tag{3-30}$$

To generate the new predicted gear sequence $\hat{\mathbf{j}}(k)$ at time step k, we take the second to the N-th elements of $\hat{\mathbf{j}}_{\text{old}}(k-1)$ and assign them as the first $N-1$ elements of $\hat{\mathbf{j}}(k)$. The last element of $\hat{\mathbf{j}}(k)$ is set as the final gear choice from $\hat{\mathbf{j}}_{\text{old}}(k-1)$. Mathematically, $\hat{\mathbf{j}}(k)$ is represented as:

$$\hat{\mathbf{j}}(k) = \begin{bmatrix} \hat{j}_{\text{old}}(k) & \cdots & \hat{j}_{\text{old}}(k+N-2) & \hat{j}_{\text{old}}(k+N-2) \end{bmatrix} \tag{3-31}$$

This approach provides a backup solution for the gear sequence when the original gear sequence is infeasible, which is refereed to **Backup solution 1**. The predicted sequence $\hat{\mathbf{j}}(k)$ is then fed again to the MPC solver. However, this backup solution is not always feasible because the last gear choice of $\hat{\mathbf{j}}(k)$ is simply replicated from the last gear choice of $\hat{\mathbf{j}}_{\text{old}}(k-1)$. If the solver returns an infeasibility result again, a second backup solution is required.

In the second backup solution, the gear sequence is determined in the same way as during initialization. First, the gear corresponding to the current velocity $v(k)$ is obtained using the PWA gear approximation. Then, this gear is applied uniformly across the entire prediction horizon. Denoting the gear determined by the PWA model as $\hat{j}_{\text{pwa}}(k)$, the second backup gear sequence is expressed as:

$$\hat{\mathbf{j}}(k) = \begin{bmatrix} \hat{j}_{\text{pwa}}(k) & \hat{j}_{\text{pwa}}(k) & \cdots & \hat{j}_{\text{pwa}}(k) \end{bmatrix}_{1\times N} \tag{3-32}$$

This second backup solution, referred to as **Backup Solution 2**, is the most conservative yet effective approach, although it does not ensure the feasibility of the MPC problem.

Figure 3-6 provides a detailed flowchart of **Step 2**, showing how the feasibility check of the gear interacts with the MPC controller.

The adjusted gear sequence is then provided to the MPC controller, which solves the optimization problem to generate the predicted state sequence $\hat{\mathbf{x}}(k)$ and control sequence $\hat{\mathbf{u}}(k)$. These predictions are subsequently used for the next steps in the decision-making process.

Step 3: Interaction with Environment After the MPC controller solves the optimization problem, the first throttle input $u(k)$ of the predicted control sequence $\mathbf{u}(k)$, combined with the first gear choice $j(k)$ of the gear sequence $\mathbf{j}(k)$, is applied to control the vehicle in the environment.

The environment then transitions to a new state $x(k+1)$ based on this control input, following the system dynamics introduced in (2-2).

Step 4: Reward Calculation As discussed in Section 3-2-2, the reward function is composed of three components: the tracking cost, the fuel cost, and the penalty cost associated with gear infeasibility. At time step k, these components are calculated as follows:

1. Tracking Cost (J_{tracking}): The tracking cost quantifies the deviation of the current state $x(k)$ from the reference trajectory $\xi(k)$, which is calculated as (3-4).
2. Fuel Cost (J_{fuel}): The fuel cost is computed based on the vehicle's velocity $x_2(k)$. It is given by (3-5).
3. Penalty Cost (J_{penalty}): The penalty cost accounts for violations in the gear sequence and MPC infeasibility. At each time step, the number of gear limit violations is denoted as n_{penalty}, with each violation penalized by ϕ_1. Additionally, a binary variable λ_{penalty} indicates whether the MPC problem is infeasible, incurring a penalty of ϕ_2 if set to 1. The variable λ_{penalty} is defined as:

$$\lambda_{\text{penalty}} = \begin{cases} 1, & \text{if MPC is infeasible} \\ 0, & \text{if MPC is feasible} \end{cases} \tag{3-33}$$

   The total penalty P is then calculated as:

$$J_{\text{penalty}} = \phi_1 \cdot n_{\text{penalty}} + \phi_2 \cdot \lambda_{\text{penalty}} \tag{3-34}$$

Finally, the reward at time step k is computed as:

$$R(k) = -\big(l_1 \cdot J_{\text{tracking}}(k) + l_2 \cdot J_{\text{fuel}}(k) + l_3 \cdot J_{\text{penalty}}(k)\big), \tag{3-35}$$

where l_1, l_2, and l_3 are weights used to balance the contributions of tracking cost, fuel cost, and penalty cost in the reward function. Note that the negative sign in the reward function is used because DQN aims to maximize the reward. By negating the costs, minimizing the costs becomes equivalent to maximizing the reward, aligning with the DQN optimization framework.

Step 5: Experience Storage At the end of each time step, the experience tuple is stored in the replay buffer. The experience is expressed as:

$$\Big(\hat{\mathbf{s}}(k-1), \Delta\hat{\mathbf{j}}(k), \hat{\mathbf{s}}(k), R(k)\Big) \tag{3-36}$$

Since the RL agent generates a gear sequence at each time step, the transitions are stored in the replay buffer as time sequences. Given that the network architecture is an RNN, this storage method allows the RL network to effectively capture temporal dependencies across multiple time steps, thereby learning the system dynamics and improving the gear selection process.

During training, we perform a feasibility check on the selected actions, introduced in **Step 2**, to ensure that the MPC problem can be solved efficiently, thereby facilitating the training process. However, when storing transitions in the replay buffer, the action sequences are the original outputs of the network, not the adjusted ones. This approach is taken because the penalty costs are calculated based on the original network outputs, which accurately reflect the agent's decision-making.

Step 6: Batch Training and Network Update

Batch Sampling During training, a batch of transitions is randomly sampled from the replay buffer at each time step. This random sampling ensures that the experiences used for training come from different episodes, which helps to break correlations between consecutive samples. By training on diverse experiences, the RL agent is better able to generalize its policy across a variety of situations.

In our training, the states and actions are sequences, while the reward is a scalar, as the sequences are generated by the MPC controller rather than through direct interaction with the environment. At each time step, only the first control input from the MPC sequence is applied to the environment, resulting in a single scalar reward.

To align this scalar reward with the sequence data, we expand the reward to a sequence to facilitate training. We broadcast it across the entire prediction horizon, giving same reward for each time step. This ensures consistency in training by associating the reward with the entire sequence.

$$\mathbf{R}(k) = \begin{bmatrix} R(k) & R(k) & \cdots & R(k) \end{bmatrix}_{1\times N} \tag{3-37}$$

Network Update With the sampled batch, the target Q-value is computed as follows:

$$Q_{\text{target}} = \mathbf{R}(k) + \gamma \cdot \max_{\Delta j} Q^{'}(\hat{\mathbf{s}}(k)), \tag{3-38}$$

where $Q^{'}$ represents the target network, γ is the discount factor, the current Q-value is given by:

$$Q_{\text{current}} = Q(\hat{\mathbf{s}}(k-1)), \tag{3-39}$$

where Q represents the policy network. Huber loss is used as the loss function, defined as:

$$L_\delta(Q_{\text{target}}, Q_{\text{current}}) = \begin{cases} \frac{1}{2}(Q_{\text{target}} - Q_{\text{current}})^2, & \text{if } |Q_{\text{target}} - Q_{\text{current}}| \le \delta, \\ \delta \cdot (|Q_{\text{target}} - Q_{\text{current}}| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases} \tag{3-40}$$

Here, δ is a threshold parameter that determines the transition between the squared loss and absolute loss regions. This ensures robustness against outliers in the value difference $|Q_{\text{target}} - Q_{\text{current}}|$. The Adam optimizer is used to minimize the loss function.

Target Network Update The target network is updated to stabilize the training process by providing a more consistent target for the Q-value estimation. Instead of directly copying the weights of the policy network to the target network at every update, a soft update is applied. The soft update is performed as follows:

$$\theta^{'} \leftarrow \tau\theta + (1-\tau)\theta^{'}, \tag{3-41}$$

where θ represents the parameters of the policy network, $\theta^{'}$ represents the parameters of the target network, and $\tau \in [0, 1]$ is a hyperparameter that controls the update rate.

The use of soft updates ensures that the target network evolves slowly over time, which reduces the risk of divergence and helps maintain the stability of Q-value updates. By gradually

blending the weights of the policy network into the target network, the training process benefits from both consistent targets and the ability to adapt to the latest policy improvements.

For each episode, after initialization, the training process iterates from **Step 1** to **Step 6** until the episode concludes. A new episode then begins from **Step 0**. Figure 3-7 illustrates the overall training process in the form of a flowchart.

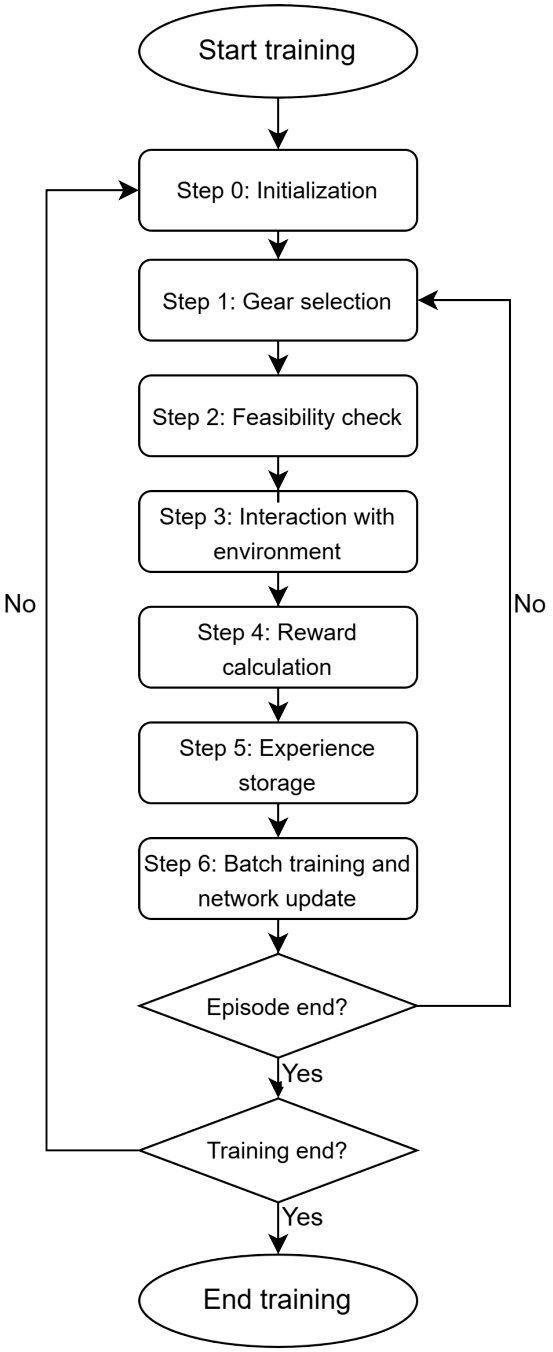


Figure 3-7: Training Flowchart Diagram

3-4 Summary

In this chapter, we presented the control methodology proposed in this research, focusing on integrating RL with MPC for vehicle control. The chapter began by formulating the problem of a single vehicle tasked with following a trajectory while minimizing a performance cost that considers both tracking accuracy and fuel consumption. We introduced the state and input definitions, along with the relevant constraints, ensuring safety and vehicle dynamics adherence.

We then discussed the challenges associated with the mixed-integer nature of the standard MPC approach, highlighting the increased computational complexity due to discrete components like gear choice. To address these challenges, we proposed an RL-based strategy to manage the gear transitions, which effectively reduces the complexity of the remaining MPC optimization problem to only a few discrete variables, thereby making real-time implementation feasible.

The design of the RL and MPC controllers was elaborated, showcasing the hybrid control architecture that divides decision-making between RL (for gear selection) and MPC (for throttle control and trajectory tracking). Finally, the RL training scheme, including the environment setup, training procedure, and network architecture, was detailed.

This chapter sets the foundation for the proposed integrated control solution, demonstrating how the combined RL and MPC approach addresses computational complexity challenges in vehicle control. The subsequent chapters will provide an evaluation of the proposed method, detailing simulation results and performance analyses to validate its effectiveness.

Chapter 4

Training and Evaluation Results

In this chapter, we present the results of our research, including the training outcomes and evaluation results for both individual episodes and a large-scale test. Additionally, we compare our approach with the standard MPC and PID approaches, focusing on performance and runtime.

4-1 Training Setup & Results

The training comprises 50,000 episodes, each lasting 60 time steps. This setting was chosen based on observed convergence after multiple tests. A new reference trajectory is generated for every episode, with the vehicle starting from randomized states. The specific training parameters are presented below, followed by the corresponding training results.

4-1-1 Training Setup

The parameters of the vehicle model are provided in Table 4-1, while the maximum traction force and velocity range for each gear are presented in Table 4-2. The details of the network is outlined in Table 4-3, while Table 4-4 specifies the parameters for the reference trajectories. Additionally, Table 4-5 summarizes the parameters employed by the MPC controller in our approach (3-6), and Table 4-6 lists the parameters used to train the RL agent.

Table 4-1: Parameter values used in 2-1

Parameter	Value	Units
m	800	kg
c	0.5	kg/m
μ	0.01	-
g	9.8	m/s^2

Table 4-2: Maximum Traction Force and Velocity Range for Each Gear.

Gear j	Traction force $b(j)$ (N)	Min. vel. (m/s)	Max. vel. (m/s)
I	4057	3.94	9.46
II	2945	5.43	13.04
III	2116	7.56	18.15
IV	1607	9.96	23.90
V	1166	13.70	32.93
VI	838	19.10	45.84

Table 4-3: Network Architecture Details

Layer	Number of Nodes
Input Layer	4
Bi-RNN Layer	2×64
Fully Connected Layer	128
Output Layer	3

Table 4-4: Parameters for Reference Trajectory and Initial Vehicle States

Parameter	Value	Description
$\xi_{1,0}$	3000	Initial reference position
$\xi_{2,\min}$, $\xi_{2,\max}$	15, 25	Range for initial reference velocity
N_{R}	5	Number of piecewise affine region
N_{T}	80	Total time steps for the reference trajectory
t_1, t_2, t_3, t_4	20, 35, 50, 70	Fixed time steps for switching regions
$o_{\min}$, $o_{\max}$	-5, 5	Range for random offsets
$\lambda_{\min}$, $\lambda_{\max}$	-0.6, 0.6	Range for velocity change rates
$x_{1,\min}$, $x_{1,\max}$	2900, 3100	Range for initial vehicle position
$x_{2,\min}$, $x_{2,\max}$	5, 25	Range for initial vehicle velocity

Table 4-5: MPC Controller Parameters

Parameter	Value
Prediction Horizon (N)	5
Sampling Time (T)	1 s
State Weight Matrix (Q_x)	diag(1, 0.1)
Control Weight Matrix (Q_u)	1
Throttle Bounds ($u_{\mathrm{MIN}}, u_{\mathrm{MAX}}$)	[-1, 1]
Velocity Limits ($v_{\mathrm{MIN}}, v_{\mathrm{MAX}}$)	[3.94, 45.84]
Acceleration Limits ($a_{\mathrm{MIN}}, a_{\mathrm{MAX}}$)	[-2, 2.5]

4-1-2 Training Results

Several key metrics are tracked over 50,000 training episodes, as depicted in Figure 4-1, including total cost J_{total} (3-3), tracking cost J_{tracking} (3-4), fuel consumption cost J_{fuel} (3-5), penalty cost J_{penalty} (3-34), and the counts of gear limit violations and MPC infeasibilities.

Table 4-6: RL Training Parameters

Parameter	**Value**
Learning Rate	0.001
Discount Factor (γ)	0.9
Batch Size	128
Number of Episodes	50000
Steps per Episode (N_{E})	60
Starting Exploration Rate ($\epsilon_{\mathrm{start}}$)	0.999
Ending Exploration Rate (ϵ_{end})	0
Exploration Decay Factor ($\epsilon_{\mathrm{decay}}$)	70000
Replay Buffer Size	100000
Target Network Soft Update Rate (τ)	0.001
Threshold of Huber Loss Function (δ)	1
Optimizer	Adam
Tracking Cost Weight (l_1)	0.001
Fuel Cost Weight (l_2)	1
Penalty Cost Weight (l_3)	1
Weights for Performance Cost (w_1, w_2)	0.0025, 1
Penalties for Gear Infeasibilities (ϕ_1, ϕ_2)	100, 50

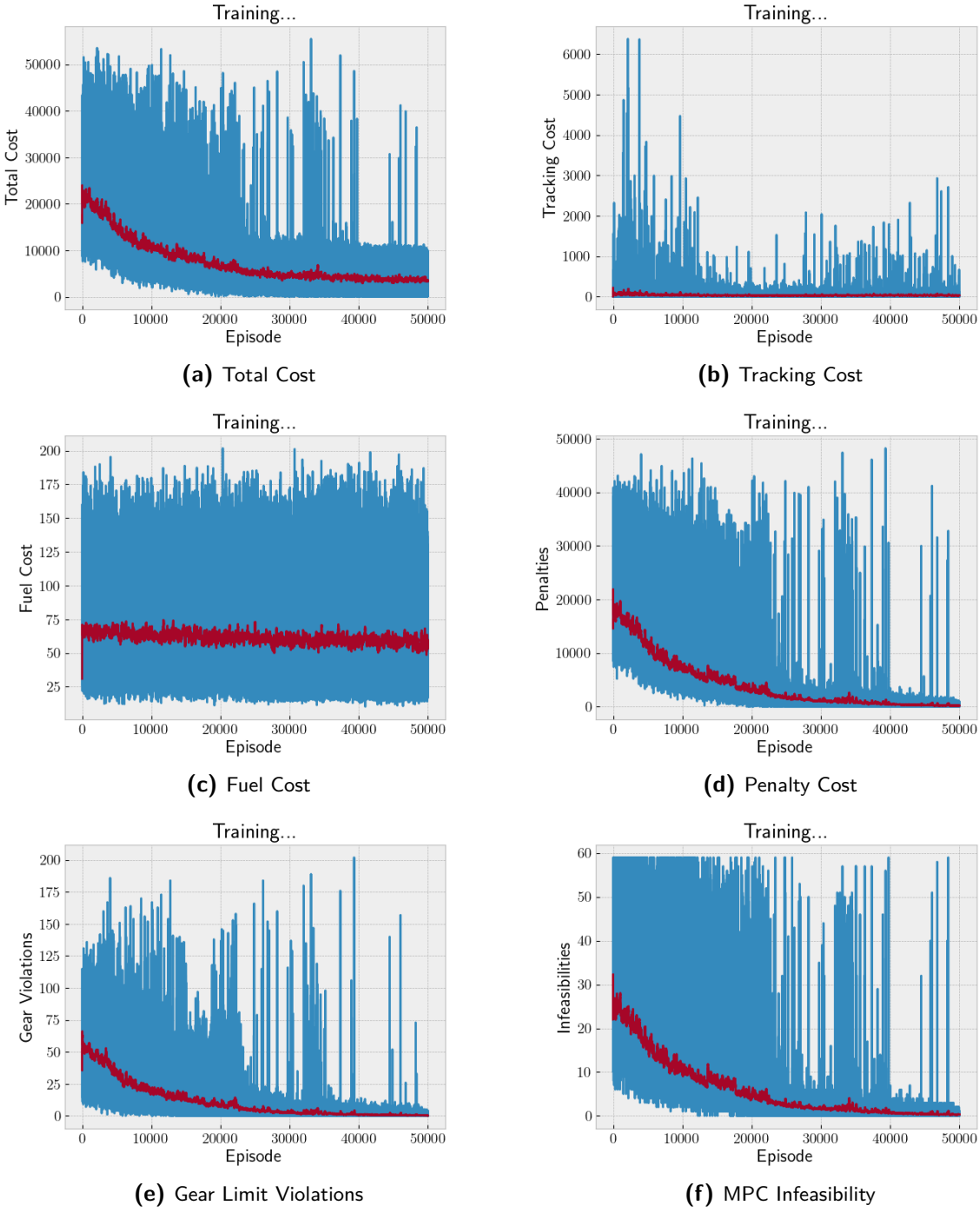
Figure 4-1a displays the total cost during training. In the early stages, the average cost is high and exhibits significant variability due to the agent's exploration of different actions to discover an effective policy. As training progresses, the average cost decreases and begins to stabilize around 40,000 episodes, with fluctuations diminishing in magnitude. These residual fluctuations are expected and are primarily caused by variations in initial conditions at the start of each episode. Although some high-cost spikes persist after 40,000 episodes, their frequency is notably reduced, indicating that the agent is effectively generalizing to most scenarios while occasionally encountering challenging conditions that result in higher costs.

A similar trend is observed in Figure 4-1d, which illustrates the penalty cost. Since the penalty cost constitutes a major portion of the total cost, reductions in the penalty cost closely align with improvements in the total cost metric. Figures 4-1e and 4-1f show the counts of gear limit violations and MPC infeasibilities, respectively. Both metrics decrease alongside the penalty cost. Their running averages approach zero, indicating that the agent is increasingly learning to select feasible gear shifts for the given prediction horizon.

Figure 4-1b illustrates the tracking cost during training. The running average remains consistently low, with occasional outliers. Unlike the penalty cost, the tracking cost does not show a significant declining trend. The gear selections have a limited direct influence on vehicle tracking performance because the MPC controller exhibits a degree of robustness to adjust vehicle dynamics effectively with feasible gear sequences. Even with infeasible gear sequences, the backup solutions enables the MPC controller to minimize the tracking cost as much as possible.

Finally, Figure 4-1c depicts the fuel cost during training. The plot shows a slight declining trend with fluctuations of consistent magnitudes. These fluctuations arise from the varying velocity profiles of different reference trajectories, which influences fuel consumption. Although

better gear selections can improve fuel efficiency, their impact in our approach is limited due to the characteristics of the selected fuel model. In this model, fuel consumption is primarily governed by vehicle velocity, which is largely controlled by the throttle input rather than gear selections.



(a) Total Cost **(b)** Tracking Cost

(c) Fuel Cost **(d)** Penalty Cost

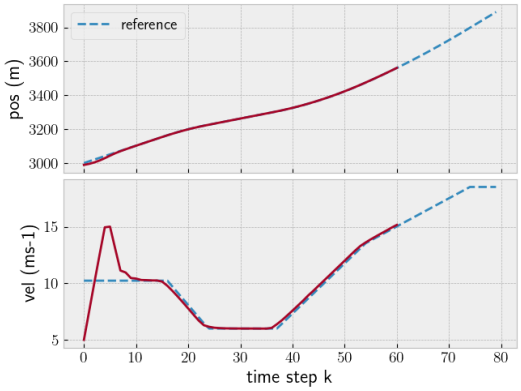
(e) Gear Limit Violations **(f)** MPC Infeasibility

Figure 4-1: Training results. In each plot, the blue line represents the metric's value for each individual episode, while the red line shows its 100-episode running average.

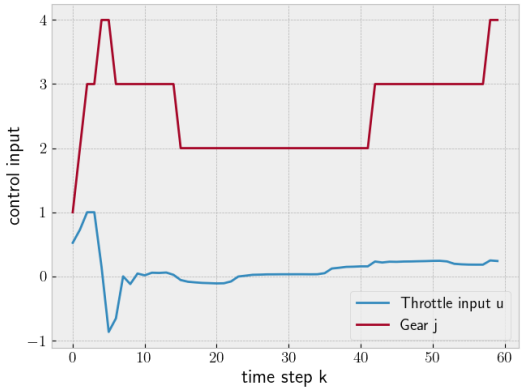
4-2 Evaluation Results

To evaluate our approach, we first conducted two individual simulations with different reference trajectories to demonstrate the RL's capability in managing gear selections across varying scenarios. Following this, we performed a larger test consisting of 100 episodes to analyze the performance metrics comprehensively. For each episode, the length of the simulation is 60 time steps.

4-2-1 Individual Tests

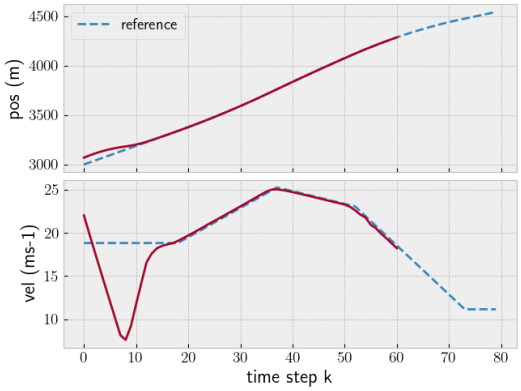


(a) Test 1: Tracking Results

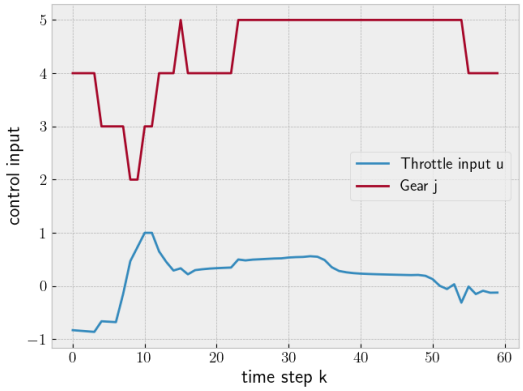


(b) Test 1: Control Inputs

Figure 4-2: Test 1 Results



(a) Test 2: Tracking Results



(b) Test 2: Control Inputs

Figure 4-3: Test 2 Results

The results for the tests are presented in Figures 4-2 and 4-3, with the left panels illustrating the tracking results. The upper subplots represent the position, and the lower subplots represent the velocity. The dotted blue line indicates the reference trajectory, while the solid red line shows the vehicle's actual trajectory. In both tests, the vehicle begins with initial states that deviate from the reference trajectory. However, it quickly converges to the reference within a few steps, demonstrating the effectiveness of the MPC-RL controller in achieving accurate tracking.

The right panels illustrate the control inputs during the tests. The red line represents the gear selected by the RL controller, while the blue line shows the throttle input provided by the MPC controller. Analyzing the control inputs alongside the tracking results reveals that the RL agent effectively selects appropriate gears to align with the vehicle's velocity, ensuring smooth transitions and maintaining tracking accuracy. In Test 1, where the reference velocity ranges between 5-15 m/s, the RL controller predominantly operates in lower gears (1 to 3). Conversely, in Test 2, where the reference velocity starts around 20 m/s and increases to approximately 25 m/s, the RL controller selects higher gears (4 and 5), showcasing its ability to return appropriate gear choices for different velocity profiles effectively.

Table 4-7 summarizes the performance metrics for both tests. Notably, there are no gear limit violations or MPC infeasibilities, indicating that the RL controller respects the defined constraints and provides feasible and optimal gear selections. Test 1 achieves a significantly lower tracking cost compared to Test 2, primarily due to the initial states being closer to the reference trajectory. Furthermore, the higher velocities in Test 2 lead to increased fuel consumption, as expected.

The runtime of both tests validates the real-time feasibility of the proposed MPC-RL controller. Test 1 achieves a total runtime of 0.38 seconds, while Test 2 takes 0.47 seconds. For Test 1, the average runtime per time step is approximately 0.0063 seconds, corresponding to a control frequency of around 159 Hz. This high control frequency is more than sufficient for high-precision autonomous vehicle control, ensuring responsive and stable operation even in dynamic environments.

Table 4-7: Performance metrics for Test 1 and Test 2

Metric	Test 1	Test 2
Runtime (s)	0.38	0.47
Performance Cost	41.57	153.35
Tracking Cost	1121.75	34269.11
Fuel Cost	38.77	67.68
Gear Limit Violations	0	0
MPC Infeasibilities	0	0

4-2-2 Large Test

We conducted a large-scale test consisting of 100 episodes, and the average performance metrics are summarized in Table 4-8.

Table 4-8: Average Performance Metrics Over 100 Test Episodes

Metric	Value
Average Runtime	0.35
Average Performance	99.62
Average Tracking Cost	19764.77
Average Fuel Cost	50.20

We analyzed the occurrence of gear limit violations and MPC infeasibilities across 100 test episodes, recording the number of such events per episode. A single gear limit violation was

observed in episode 85, and one MPC infeasibility occurred in episode 23. These isolated events demonstrate that the RL controller effectively learned and applied a policy that generalizes well to diverse scenarios, providing feasible gear selections for the MPC controller in nearly all episodes. Furthermore, in the rare episodes where infeasibilities occurred, the backup solution successfully addressed the issues, ensuring the solvability of the MPC problem and maintaining the continuity of the control process.

4-3 Comparative Analysis

In this section, we compare our approach with the standard MPC and PID methods by performing the same individual and large-scale tests.

4-3-1 Standard MPC Approach

The standard MPC approach optimizes all objectives simultaneously by solving for all decision variables within the model. For this comparison, we utilized the PWA model for the gear model (2-1-1). As demonstrated in [9], it shows that while the PWA model's performance is only slightly worse compared to the discrete input model, it significantly reduces computation time.

Optimization Problem

The standard MPC optimization objective consists of three components: tracking cost, fuel cost, and control cost. The decision variables include both the vehicle's throttle input **u** and gear selection **j**, with the gear selection implicitly incorporated into the optimization problem through the vehicle model, which includes the gear model. The MPC optimization problem is formulated as:

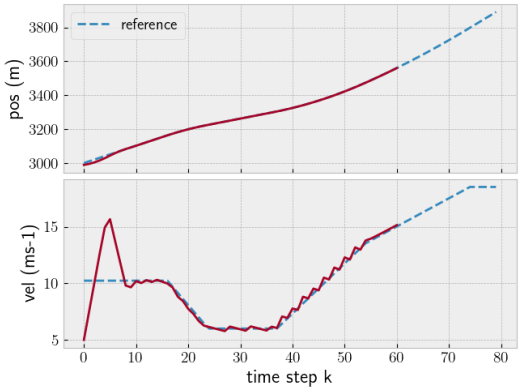
$$\begin{aligned} J(x) = \min_{\mathbf{u},\mathbf{j},\mathbf{x}} & \underbrace{\sum_{k=0}^{N+1} \|x(k) - r(k)\|_{Q_x}}_{\text{Tracking Cost}} \\ & + \underbrace{Q_f \sum_{k=0}^{N} \left(\sum_{m=0}^{3} b_m x_2^m(k) + \frac{x_2(k+1) - x_2(k)}{T} \sum_{l=0}^{2} c_l x_2^l(k) \right)}_{\text{Fuel Cost}} \\ & + \underbrace{\sum_{k=0}^{N} \|u(k)\|_{Q_u}}_{\text{Control Cost}} \\ \text{s.t. } & \text{(3-1) - (3-2)} \\ & \text{vehicle model (2-7)} \\ & x(0) = x \end{aligned} \tag{4-1}$$

The parameters for standard MPC approach are the same as Table 4-5, with an additional fuel cost weight Q_f due to the inclusion of fuel consumption cost into the MPC problem. The coefficients of the polynomial in fuel consumption are listed in Table 4-9.

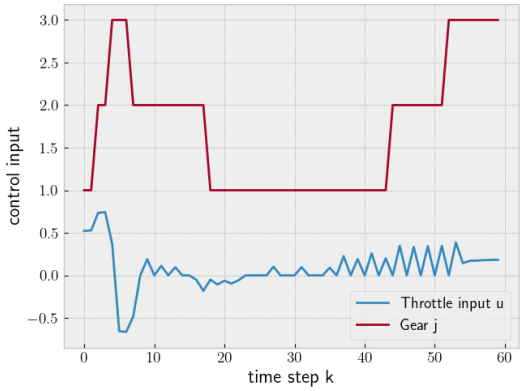
Table 4-9: Coefficients for Fuel Consumption

Parameter	Value
b_0, b_1, b_2, b_3	0.1569, $2.450 \cdot 10^{-2}$, $-7.415 \cdot 10^{-4}$, $5.975 \cdot 10^{-5}$
c_0, c_1, c_2	0.0724, $9.681 \cdot 10^{-2}$, $1.075 \cdot 10^{-3}$

Individual Tests

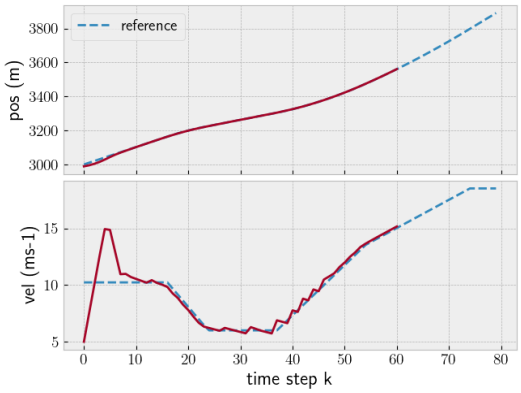


(a) Test 1 Tracking Results

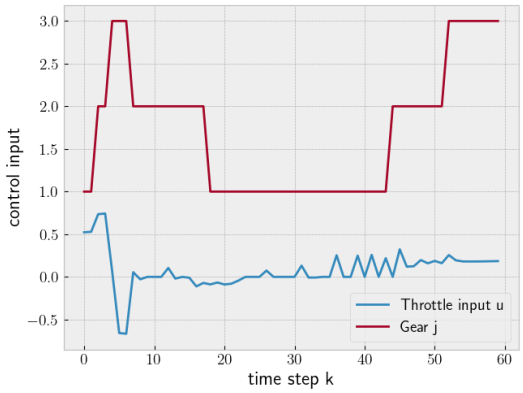


(b) Test 1: Control Inputs

Figure 4-4: Standard MPC Test 1 Results ($N = 2$)



(a) Test 1: Tracking Results



(b) Test 1: Control Inputs

Figure 4-5: Standard MPC Test 1 Results ($N = 5$)

We conducted Test 1 using the standard MPC approach with prediction horizons $N = 2$ and $N = 5$, setting the fuel cost weight $Q_f = 1$. The tracking results and control inputs are shown in Figures 4-4 and 4-5.

In both tests with different prediction horizons, jagged shapes are observed in the throttle input u and velocity x_2. This behavior arises from the MPC's attempt to balance tracking accuracy and fuel cost, as the fuel model assumes no fuel is consumed when $u \leq 0$. Figure 4-5, with a prediction horizon of 5, shows fewer jagged shapes because the extended prediction horizon enables the MPC to anticipate future states more effectively. This allows for smoother

transitions of the control inputs over time, reducing abrupt changes in both throttle input and velocity.

As shown in Table 4-10, for $N = 2$, the performance cost is higher than that of our approach, with a runtime of 165.33 seconds to complete the test, which is significantly longer than the runtime of our method. For $N = 5$, the performance cost is slightly lower than our approach, but the runtime increases dramatically, requiring approximately 27.81 seconds to solve a single time step. This highlights the computational inefficiency of the standard MPC approach.

Table 4-10: Standard MPC Performance Metrics for Test 1 with Different Prediction Horizons ($N = 2$ and $N = 5$)

Metric	$N = 2$	$N = 5$
Runtime (s)	165.33	1668.88
Performance	43.79	40.99
Tracking Cost	1107.95	1120.115
Fuel Cost	41.02	38.19

Large Test

Large-scale test was conducted similar to those described in Section 4-2-2. To achieve the lowest average performance cost, the standard MPC was optimized by exploring various combinations of prediction horizon (N) and fuel cost weight (Q_f), with N ranging from 2 to 5 and Q_f ranging from 1 to 5. Each combination was evaluated over 100 episodes.

Table 4-11: Standard MPC Large Tests ($N = 2$)

Q_f	1	2	3
Average Runtime	356.29	373.62	382.03
Average Performance	101.85	101.43	101.31
Average Tracking Cost	18867.31	18945.19	19093.79
Average Fuel Cost	54.68	54.07	53.57

Table 4-12: Standard MPC Large Tests ($N = 3$)

Q_f	1	2	3
Average Runtime	744.03	775.04	783.17
Average Performance	98.53	98.36	98.22
Average Tracking Cost	18353.05	18354.14	18364.11
Average Fuel Cost	52.65	52.48	52.31

Tables 4-11, 4-12, and 4-13 present the results for different N and Q_f combinations. However, due to the increasing complexity of the optimization problem with larger N and Q_f values, the MPC solver failed to converge for certain configurations. Consequently, results are only available for a subset of the tested combinations. Table 4-14 lists the specific configurations that led to MPC solver failures. This includes the parameter combinations (N and Q_f), the episode and time step number at which the failure occurred, and the total runtime of the test.

Table 4-13: Standard MPC Large Tests ($N = 4$)

Q_f	1
Average Runtime	1926.66
Average Performance	97.28
Average Tracking Cost	18238.11
Average Fuel Cost	51.68

From the results in Tables 4-11, 4-12, and 4-13, it is evident that increasing the prediction horizon N results in a noticeable reduction in the performance cost. However, this improvement comes at the cost of significantly higher average runtimes per episode. Furthermore, increasing the fuel cost weight Q_f slightly reduces the performance cost across all prediction horizons but also leads to a minor increase in runtime, likely due to the added complexity of the optimization problem.

Table 4-14: Records of Standard MPC Failures

Prediction Horizon N	2	3	4	5
Q_f Number	4	4	2	1
Episode Number	8	81	53	84
Time Step Number	2	2	27	37
Test Total Runtime	2843.45	63163.45	106240.21	315835.19

4-3-2 PID Approach

The PID approach uses proportional, integral, and derivative components to compute control outputs based on the current error, the accumulated error over time, and the rate of change of error, respectively. Its simplicity and ability to enable fast control make it an effective baseline for comparison.

Controller Design

The control strategy employs two separate PID controllers to regulate the vehicle's acceleration based on position and velocity errors. The desired acceleration is computed by combining the outputs of these controllers.

- **Position PID Controller:** The position error is calculated as the difference between the actual and desired positions. The controller output, $a_{\text{des,pos}}$, is determined as:

$$a_{\text{des,pos}} = k_p^{\text{pos}} e_{\text{pos}} + k_i^{\text{pos}} \int e_{\text{pos}} dt + k_d^{\text{pos}} \frac{de_{\text{pos}}}{dt}, \tag{4-2}$$

  where e_{pos} is the position error, and k_p^{pos}, k_i^{pos}, and k_d^{pos} are the proportional, integral, and derivative gains for position control.

- **Velocity PID Controller:** The velocity error is calculated as the difference between the desired velocity and the actual velocity. The controller output, $a_{\text{des,vel}}$, is given by:

$$a_{\text{des,vel}} = k_p^{\text{vel}} e_{\text{vel}} + k_i^{\text{vel}} \int e_{\text{vel}} dt + k_d^{\text{vel}} \frac{de_{\text{vel}}}{dt}, \tag{4-3}$$

  where e_{vel} is the velocity error, and k_p^{vel}, k_i^{vel}, and k_d^{vel} are the proportional, integral, and derivative gains for velocity control.

The outputs of the two controllers are combined to compute the desired acceleration, a_{des}, using a weighted sum:

$$a_{\text{des}} = \text{clip}\left(w_{\text{pos}} \cdot a_{\text{des,pos}} + w_{\text{vel}} \cdot a_{\text{des,vel}}, a_{\text{MIN}}, a_{\text{MAX}}\right), \tag{4-4}$$

where w_{pos} and w_{vel} are the weights assigned to the position and velocity controller outputs, respectively.

The `clip` function restricts the value of the computed desired acceleration a_{des} to fall within a specific range, bounded by a_{MIN} and a_{MAX}. It ensures that the resulting value remains within these predefined limits, regardless of the weighted sum of the acceleration components.

Mathematically, the `clip` function operates as:

$$\text{clip}(x, x_{\min}, x_{\max}) = \begin{cases} x_{\min}, & \text{if } x < x_{\min} \\ x, & \text{if } x_{\min} \leq x \leq x_{\max} \\ x_{\max}, & \text{if } x > x_{\max} \end{cases} \tag{4-5}$$

Finally, the gear choice is decided by the PWA gear model (Figure 2-2b), and the desired throttle input is determined using vehicle dynamics, which is calculated as:

$$u = \text{clip}\left(\frac{1}{b(j, x_2)}(ma_{\text{des}} + cx_2^2 + \mu mg), u_{\text{MIN}}, u_{\text{MAX}}\right) \tag{4-6}$$

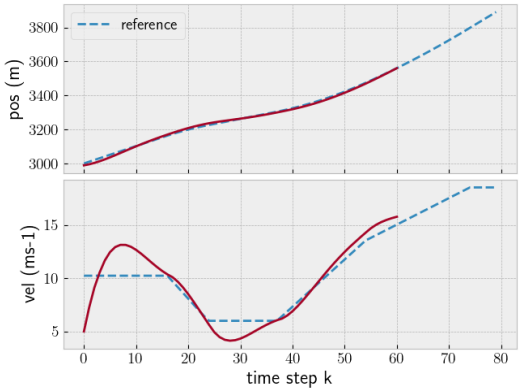
The specific values of the parameters used in the following tests are listed in Table 4-15. Notably, during the tuning process, certain parameter values resulted in violations of the vehicle velocity limit.

Table 4-15: PID Controller Parameters

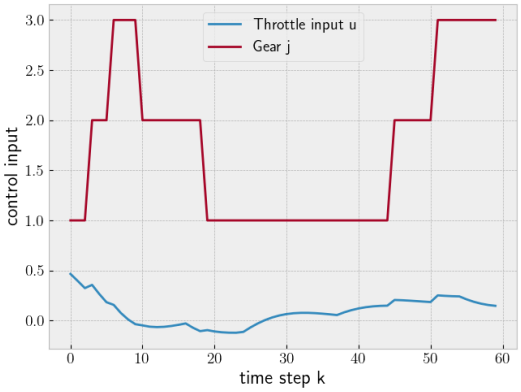
Parameter	Value
k_p^{pos}	0.05
k_i^{pos}	0.01
k_d^{pos}	0.0
k_p^{vel}	0.7
k_i^{vel}	0.1
k_d^{vel}	0.0
w_{pos}	0.55
w_{vel}	0.45

Individual Test

We conducted Test 1 using the PID control approach, with the results presented in Figure 4-6 and Table 4-16. We prioritize position tracking so it is assigned to a higher weight when calculating tracking cost, so the PID approach exhibits less strict reference tracking for velocity compared to both our proposed method and the standard MPC approach. This trade-off is evident in the velocity profile, where the deviation from the reference is more noticeable. The PID controller has a minimal runtime as it avoids solving optimization problems. Despite this advantage, it shows the highest performance cost among all approaches, as reflected in the metrics.



(a) Test 1: Tracking Results



(b) Test 1: Control Inputs

Figure 4-6: PID Test 1 Results

Table 4-16: PID Performance metrics for Test 1

Runtime	Performance	Tracking Cost	Fuel Cost
0.02	46.53	3745.52	37.16

Large Test

The large-scale test using the PID controller were conducted, with the results presented in Table 4-17. It is worth noting that the PID approach demonstrates a favorable trade-off between runtime and performance cost, with an average runtime of only 0.02 seconds per episode and a slightly worse performance cost compared to standard MPC and our MPC-RL approach. The average fuel cost is notably lower compared to the standard MPC and our MPC-RL approaches. This is likely attributed to the PID controller's lower emphasis on velocity control, resulting in reduced vehicle velocities and consequently lower fuel consumption.

Although the PID approach demonstrates reasonable performance, its limitations in flexibility and handling constraints restrict its applicability. These limitations will be further discussed in the next section.

Table 4-17: PID Controller Large Tests

Metric	Value
Average Runtime	0.02
Average Performance	100.46
Average Tracking Cost	21456.30
Average Fuel Cost	46.82

4-3-3 Comparison & Discussion

We compared the MPC-RL approach, the standard MPC approach with $N = 4$ and $Q_f = 1$, and the PID approach under large-scale testing. The results are summarized in Table 4-18.

The standard MPC approach achieves the lowest performance cost, showcasing its strong optimization capability by explicitly incorporating all decision variables within a single optimization problem. However, this comes at the expense of an extremely high runtime, averaging 1926.66 seconds per episode, making it impractical for real-time applications. The excessive computational demand severely limits its feasibility in dynamic scenarios requiring rapid decision making.

In contrast, the PID controller achieves the fastest runtime, averaging just 0.02 seconds per episode. While this makes it computationally efficient, its performance is slightly worse than both MPC-RL and standard MPC approaches. Additionally, the parameters may need to be retuned to achieve optimal performance when encountering different reference trajectories, highlighting the lack of flexibility in the PID approach compared to our method. Moreover, the PID controller lacks the capability to handle constraints effectively, which can result in unsafe or suboptimal behavior. While input constraints can be managed through clipping, state constraints, such as the vehicle velocity limit, are not inherently guaranteed. In more complex scenarios, such as vehicle platooning, hard constraints like maintaining a safe inter-vehicle distance cannot be reliably enforced using the PID approach. This limitation in flexibility and constraint management restricts its applicability.

The MPC-RL approach effectively overcomes the drawbacks of both PID and standard MPC approaches. It provides a near-optimal solution while achieving a fast average runtime of 0.35 seconds per episode, making it highly suitable for real-time control applications. Unlike PID, which lacks the ability to handle constraints, MPC-RL incorporates constraint-handling capabilities, ensuring safe and feasible control actions. At the same time, it avoids the excessive computational burden of standard MPC by excluding gear selection and fuel optimization from the optimization problem, offering a practical balance between performance and efficiency for autonomous vehicles in dynamic environments.

Table 4-18: Comparison of MPC-RL, Standard MPC, and PID Approaches

Metric	MPC-RL	Standard MPC ($N = 4$, $Q_f = 1$)	PID
Average Runtime (s)	0.35	1926.66	0.02
Average Performance	99.62	97.28	100.46
Average Tracking Cost	19764.77	18238.11	21456.30
Average Fuel Cost	50.20	51.68	46.82

4-4 Summary

This chapter presented the training and evaluation results for the proposed MPC-RL approach and compared it with the standard MPC and PID controllers. During training, the MPC-RL approach demonstrated consistent improvements, with total and penalty costs converging over 50,000 episodes. Gear limit violations and MPC infeasibilities were effectively minimized, demonstrating the reliability of the learned policy. The results highlighted the MPC-RL controller's ability to optimize performance cost while maintaining good generalizability across varying scenarios.

In the evaluation phase, MPC-RL was tested under individual scenarios with different reference trajectories and in a large-scale test of 100 episodes. The individual tests revealed that MPC-RL achieved near-optimal performance, accurately managing gear selections while adhering to system constraints. Furthermore, the low runtime in these tests validated the real-time feasibility of the MPC-RL approach. In the large-scale tests, MPC-RL maintained strong performance, achieving an average runtime of 0.35 seconds per episode and an average performance score of 99.62. The MPC-RL policy generalized effectively across diverse scenarios, with only one gear limit violation and one MPC infeasibility recorded across all episodes.

The comparative analysis highlighted the strengths and limitations of each approach. The standard MPC method achieved the lowest performance cost due to its strong optimization capabilities. However, its prohibitively high runtime, such as 1926.66 seconds per episode for $N = 4$, rendered it unsuitable for real-time applications. The PID controller offered the fastest runtime of just 0.02 seconds per episode but exhibited the highest performance cost and lacked robust constraint-handling capabilities. In contrast, MPC-RL effectively balanced the strengths of both methods, achieving real-time feasibility, robust constraint handling, and near-optimal performance. While its performance cost was slightly higher than that of the standard MPC approach, the significant reduction in runtime makes MPC-RL a practical choice for real-world implementations.

In conclusion, MPC-RL provides a practical solution for autonomous vehicle control. It efficiently balances performance, constraint adherence, and computational demands, making it highly suitable for dynamic, real-time environments.

Chapter 5

Discussion

In this chapter we present and discuss the key findings of our research, examine its limitations, and provide recommendations for future work.

5-1 Key Findings of the Research

5-1-1 Integration of RL and MPC

In this research, we propose a combined RL and MPC framework to manage the high complexity of the mixed-integer optimization problem in autonomous vehicle control. By employing RL to handle the discrete components that make the optimization problem more complex, we significantly reduce computational demands, while MPC ensures overall performance and safety by governing the key control inputs and enforcing constraints. Similar to existing works [3] [2] [4], this approach leverages the complementary strengths of learning-based and model-based techniques: the learning-based controller (e.g., RL) addresses the more complex components of the optimization problem, while the model-based controller (e.g., MPC) ensures performance and enforces constraint satisfaction.

5-1-2 Addressing the Prediction Horizon Challenge

A key challenge in integrating RL and MPC lies in handling the prediction horizon. Typically, an RL controller produces a single action at each time step, but in our framework, it is designed to generate a sequence of actions that align with the MPC controller's prediction horizon. To achieve this, we incorporate an RNN into the RL controller's architecture. A variation of DQN known as Deep Recurrent Q-Network (DRQN) is often employed for environments modeled as partially observable Markov decision processes (POMDPs). DRQN uses an RNN to encode sequential observations, allowing it to infer hidden states and make better decisions in cases where the agent has limited access to the complete state of the environment. These implementations often follow a many-to-one pattern, where a sequence of past states

yields a single action. In contrast, our research adopts a many-to-many scheme, enabling the RL controller to produce an entire action sequence at once. Moreover, unlike traditional applications, in this study, the input sequence represents future predicted states rather than historical data. This novel adaptation ensures a seamless integration between the RL and MPC frameworks, offering an effective method for coordinating multi-step decision-making in complex optimization scenarios.

5-1-3 Scalability to Vehicle Platooning

Vehicle platooning refers to the coordinated control of multiple autonomous vehicles, leveraging precise measurements and vehicle-to-vehicle communication to maintain tight formations, enhance efficiency, and improve traffic flow. Although our research focuses on a single vehicle, the approach is readily scalable to platoon scenarios. This scalability arises from the RL contoller's ability to operate independently without requiring coordination at the RL level, treating the preceding vehicle as a dynamic reference. In the meantime, the MPC controller can be extended to a distributed MPC (DMPC) framework, leveraging inter-vehicle communication for improved coordination. For example, the Alternating Direction Method of Multipliers (ADMM) is an efficient DMPC algorithm that iteratively exchanges plans to enhance performance. While standard MPC combined with ADMM may be computationally intensive due to the problem's complexity and iterative nature, our integrated MPC-RL approach creates new opportunities for real-time DMPC implementation. When discrete components are present in the MPC problem, the optimization becomes non-convex, and ADMM provides no guarantees of convergence to the global optimum. However, if all discrete components are fixed—such as through the use of a more capable RL agent to determine these components—the remaining MPC formulation becomes convex. In this convex setting, ADMM can reliably converge to the global optimum, enabling its effective application in vehicle platooning scenarios.

5-2 Limitations

5-2-1 Limited Theoretical Guarantees

Although the RL controller can converge during training, it does not inherently ensure that the learned policy always produces feasible gear selections, in contrast to a standard MPC framework that is explicitly designed to respect constraints. While backup solutions have been devised to handle gear infeasibility most of the time, there remains a risk that these remedies might fail under certain conditions. In such cases, the vehicle's ability to maintain desirable or even acceptable performance can be compromised. This lack of formal feasibility guarantees not only introduces operational uncertainties but also poses challenges for safety-critical applications where reliability and adherence to constraints are crucial.

5-2-2 Limited Generalizability

The RL controller, during training, becomes closely tailored to the specific vehicle and gear dynamics it encounters. Consequently, any significant change in the underlying dynam-

ics—such as variations in vehicle mass, powertrain characteristics, or environmental conditions—necessitates retraining the RL controller. This retraining process is likely to be resource-intensive, time-consuming, and may require extensive hyperparameter tuning to achieve acceptable performance. As a result, the current approach lacks robust generalizability, limiting its adaptability to new scenarios or altered system configurations without substantial additional effort.

5-2-3 Residual Discrete Components

While the RL controller effectively manages the critical discrete element—gear selection—from the original optimization problem, residual discrete components may still pose risks. In scenarios demanding improved performance, extending the prediction horizon could be necessary, which in turn increases the computational complexity. As the dimension of discrete variables grows, the computation time can once again become infeasible for real-time implementation.

5-3 Recommendations

5-3-1 More Advanced RL Controller Design

While the current study employs a relatively straightforward RL controller, future work could improve its design in several key areas. First, more sophisticated techniques in DQN such as Double DQN, or other algorithms such as Proximal Policy Optimization (PPO) may yield more stable learning and better sample efficiency. Second, the network architecture could be enhanced beyond a basic recurrent structure, for instance by integrating LSTM or other advanced recurrent units that capture longer-term dependencies and improve decision-making under uncertainty. Finally, although current hyperparameter selection relies on empirical trials, a more systematic approach such as grid search could further refine model performance.

5-3-2 Extend to Distributed MPC and Vehicle Platoons

While the proposed MPC-RL framework was developed and tested on a single-vehicle scenario, the same principles can be applied to more complex setups involving multiple vehicles operating cooperatively. Future work could integrate DMPC scheme that leverages inter-vehicle communication to coordinate actions and improve overall traffic flow, safety, and efficiency. By combining RL controllers that handle discrete gear decisions with DMPC methods designed to enforce constraints and optimize global objectives, it may be possible to achieve real-time performance even in large-scale platoons.

5-3-3 Incorporating Advanced Simulations for Validation

The current framework was validated using a simplified simulation environment. To enhance the applicability of the proposed methods, future work should leverage more advanced simulation platforms, such as CARLA, or conduct real-world testing. Advanced simulations can provide a more realistic representation of vehicle dynamics, complex traffic scenarios, and

environmental factors. These improvements would help bridge the gap between simulation-based results and real-world deployment, ensuring that the framework performs reliably under practical conditions.

5-4 Summary

This chapter discussed the key findings, limitations, and recommendations of the proposed MPC-RL framework. By integrating RL to handle discrete components and MPC to ensure performance and constraint satisfaction, the framework effectively reduces computational complexity and enables multi-step decision-making. The use of a recurrent RL approach allows the generation of entire gear sequences, addressing the prediction horizon challenge in the combination of MPC and RL.

However, limitations include a lack of theoretical guarantees for constraint satisfaction, limited generalizability to new system dynamics, and challenges with residual discrete components under extended prediction horizons. Recommendations for future work include adopting advanced RL techniques like Double DQN or PPO, extending the framework to distributed MPC for vehicle platooning, and validating it through advanced simulations or real-world testing.

Chapter 6

Conclusion

This research presents a novel control framework that integrates RL with MPC to address the computational challenges in vehicle control for autonomous driving. The proposed approach decomposes the hybrid control problem by assigning discrete decision-making, specifically the gear selection, to the RL controller, while MPC focuses on optimizing continuous throttle control. By doing so, the computational complexity of solving mixed-integer nonlinear optimization problems is significantly reduced, enabling real-time implementation while maintaining near-optimal performance.

The RL controller leverages a DQN integrated with a bidirectional RNN to process temporal sequences from MPC predictions and generate optimal gear selections. This innovative architecture allows the RL controller to capture temporal dependencies and produce gear decisions that align with the predicted states over the MPC horizon. Meanwhile, the MPC controller ensures accurate trajectory tracking, and constraint satisfaction. Extensive training and evaluation demonstrated the effectiveness of the proposed MPC-RL framework, achieving real-time feasibility with an average runtime of 0.35 seconds per episode.

Compared to the standard MPC approach, the MPC-RL method achieves significantly better computational efficiency. While the standard MPC delivers slightly better performance, its prohibitive runtime limits its practical applicability for real-time systems. In contrast, the MPC-RL framework balances computational efficiency and performance, making it suitable for real-world dynamic environments. Additionally, when compared to the PID controller, MPC-RL demonstrates stronger flexibility and constraints handling.

Although the proposed method achieves strong performance, certain limitations remain. The RL controller does not provide formal guarantees of feasibility, and occasional retraining may be required for significant changes in vehicle dynamics or environmental conditions. Future work can address these challenges by exploring more advanced RL algorithms and simulation scenarios. Moreover, the control framework can also be extended to vehicle platoons through DMPC.

In conclusion, this research successfully demonstrates that the integration of RL and MPC provides an effective and practical solution to the challenges of vehicle control in autonomous

driving. The MPC-RL framework achieves real-time feasibility, reliable constraint handling, and near-optimal performance, offering a promising direction for future developments in hybrid control strategies for autonomous vehicles.

Bibliography

[1] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[2] Abhishek Cauligi, Ankush Chakrabarty, Stefano Di Cairano, and Rien Quirynen. Prism: Recurrent neural networks and presolve methods for fast mixed-integer optimal control. In *Learning for Dynamics and Control Conference*, pages 34–46. PMLR, 2022.

[3] Abhishek Cauligi, Preston Culbertson, Edward Schmerling, Mac Schwager, Bartolomeo Stellato, and Marco Pavone. Coco: Online mixed-integer control via supervised learning. *IEEE Robotics and Automation Letters*, 7(2):1447–1454, 2021.

[4] Caio Fabio Oliveira da Silva, Azita Dabiri, and Bart De Schutter. Integrating reinforcement learning and model predictive control with applications to microgrids. *arXiv preprint arXiv:2409.11267*, 2024.

[5] Defeng He, Tianxiang Qiu, and Renshi Luo. Fuel efficiency-oriented platooning control of connected nonlinear vehicles: a distributed economic MPC approach. *Asian Journal of Control*, 22(4):1628–1638, 2020.

[6] Manjiang Hu, Chongkang Li, Yougang Bian, Hui Zhang, Zhaobo Qin, and Biao Xu. Fuel economy-oriented vehicle platoon control using economic model predictive control. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):20836–20849, 2022.

[7] Guoqiang Li and Daniel Görges. Ecological adaptive cruise control for vehicles with step-gear transmission based on reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4895–4905, 2019.

[8] Meng Li, Zehong Cao, and Zhibin Li. A reinforcement learning-based vehicle platoon control strategy for reducing energy consumption in traffic oscillations. *IEEE Transactions on Neural Networks and Learning Systems*, 32(12):5309–5322, 2021.

[9] Samuel Mallick, Azita Dabiri, and Bart De Schutter. A comparison benchmark for distributed hybrid MPC control methods: Distributed vehicle platooning. *arXiv preprint arXiv:2401.09878*, 2024.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015.

[12] Gerrit JL Naus, Rene PA Vugts, Jeroen Ploeg, Marinus JG van De Molengraft, and Maarten Steinbuch. String-stable CACC design and experimental validation: A frequency-domain approach. *IEEE Transactions on Vehicular Technology*, 59(9):4268–4279, 2010.

[13] Jeroen Ploeg, Dipan P Shukla, Nathan Van De Wouw, and Henk Nijmeijer. Controller synthesis for string stability of vehicle platoons. *IEEE Transactions on Intelligent Transportation Systems*, 15(2):854–865, 2013.

[14] Elaine Shaw and J Karl Hedrick. String stability analysis for heterogeneous vehicle strings. In *2007 American Control Conference*, pages 3118–3125. IEEE, 2007.

[15] Valerio Turri, Bart Besselink, and Karl H Johansson. Gear management for fuel-efficient heavy-duty vehicle platooning. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 1687–1694. IEEE, 2016.

[16] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[17] Yujia Wu, Shengbo Eben Li, Yang Zheng, and J Karl Hedrick. Distributed sliding mode control for multi-vehicle systems with positive definite topologies. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 5213–5219. IEEE, 2016.

[18] Yanli Yin, Xuejiang Huang, Sen Zhan, Xinxin Zhang, and Fuzhen Wang. Hierarchical model predictive control strategy based on Q-learning algorithm for hybrid electric vehicle platoon. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 238(2-3):385–402, 2024.

[19] Yang Zheng, Shengbo Eben Li, Keqiang Li, Francesco Borrelli, and J Karl Hedrick. Distributed model predictive control for heterogeneous vehicle platoons under unidirectional topologies. *IEEE Transactions on Control Systems Technology*, 25(3):899–910, 2016.

[20] Yang Zheng, Shengbo Eben Li, Keqiang Li, and Wei Ren. Platooning of connected vehicles with undirected topologies: Robustness analysis and distributed H-infinity controller synthesis. *IEEE Transactions on Intelligent Transportation Systems*, 19(5):1353–1364, 2017.

Integrating MPC and RL for Efficient Control of Autonomous Vehicles

1st Qizhang Dong
Faculty of Mechanical Engineering
Delft University of Technology
Delft, the Netherlands

***Abstract*—Autonomous vehicles have great potential to improve traffic efficiency and reduce fuel consumption. Model Predictive Control (MPC) is widely used for its ability to ensure constraint satisfaction and optimal control. However, incorporating discrete dynamics like gear changes into MPC increases the computational complexity of the optimization problem, which makes a real-time implementation of the control algorithm challenging. A possible approach to address this issue is to leverage reinforcement learning (RL) to manage discrete decisions such as gear selection, simplifying the MPC optimization problem. This research integrates RL and MPC for vehicle control, where RL handles gear transitions, and MPC focuses on overall vehicle dynamics, achieving computational efficiency and near-optimal performance comparable to conventional MPC approaches.**

***Index Terms*—Autonomous vehicles, model predictive control, reinforcement learning, hybrid control systems, gear selection optimization**

I. Introduction

Autonomous driving technology has attracted significant attention for its potential to improve traffic flow, reduce congestion, and optimize energy consumption. As the demand for safer and more sustainable transportation grows, autonomous driving solutions are being rigorously researched and developed. Core aspects of autonomous driving, such as vehicle control, inter-vehicle spacing, and energy optimization, have been extensively studied. Traditional control methods, including linear control [1] [2], sliding mode control [3] and H_∞ control [4] [5] are commonly used to ensure stable vehicle performance under varying conditions.

Early research in this field often relied on simplified vehicle models focusing solely on continuous dynamics, neglecting discrete components like gear shifts. In these approaches, gear control was typically handled by low-level rule-based systems, limiting optimization of driving performance and fuel efficiency. While these models facilitated theoretical analysis, they failed to capture the complexities of real-world vehicle dynamics involving both continuous and discrete actions. Recent studies have shown that explicitly optimizing gear selection can enhance driving performance and fuel efficiency, enabling more realistic and effective control strategies [6] [7].

Model Predictive Control (MPC) is an effective approach for managing both continuous and discrete variables, optimizing control actions in real time while addressing multiple objectives and system constraints. This makes it particularly suitable for complex tasks like car-following and trajectory planning, offering stable and reliable performance in dynamic conditions [6] [8] [9] [10]. However, incorporating gear dynamics introduces discrete variables, resulting in a hybrid MPC formulation. When combined with the nonlinear and non-convex fuel consumption model, this creates a mixed-integer nonlinear optimization problem. The high computational complexity of such problems makes real-time performance a significant challenge in dynamic driving environments where rapid responses are critical for safety and control.

Reinforcement learning (RL) offers a promising solution to address this challenge. By learning policies through interaction with the environment, RL excels at real-time decision-making tasks [11] [12]. In this work, RL is used to determine the vehicle's gear choice, replacing most discrete components in the hybrid MPC optimization problem. Additionally, incorporating the fuel consumption cost into the RL framework allows to remove the fuel consumption model from the MPC, further reducing complexity and enabling real-time implementation.

The remainder of this paper is organized as follows: Section II introduces the vehicle model and the problem formulation. Section III details the proposed control methodology. Section IV introduces the training scheme of RL controller. Section V presents the training and evaluation results. Section VI compares our approach against other methods. Finally, Section VII concludes the study. Appendix A provides the numerical values of the parameters used for the controller design and simulation.

II. Model and Problem Formulation

This section presents an overview of the vehicle model and outlines the problem formulation.

A. Vehicle Model

In this study, we employ a point mass vehicle model that explicitly accounts for gear dynamics. The dynamics of a forward-moving vehicle are represented by:

$$m\ddot{p}(t) + c\dot{p}(t)^2 + \mu mg = b(j, \dot{p})u(t), \tag{1}$$

where $p(t)$ represents the position at time t, c is the aerodynamic drag coefficient, μ is the friction coefficient, $j \in 1, \ldots, 6$ indicates the selected gear, and $b(j, \dot{p})u(t)$ represents the traction force, which is proportional to the normalized

throttle input $u(t)$. The parameters used are listed in Table VIII.

Defining the state vector as position and velocity, i.e., $x = \begin{bmatrix} p & \dot{p} \end{bmatrix}^\top$, the state-space representation can be expressed as:

$$\dot{x} = A(x) + B(j, x)u, \tag{2}$$

where:

$$A(x) = \begin{bmatrix} x_2 \\ -(c/m)x_2^2 - \mu g \end{bmatrix}, \quad B(j, x) = \begin{bmatrix} 0 \\ \frac{b(j,x_2)}{m} \end{bmatrix} \tag{3}$$

The system dynamics is nonlinear due to the quadratic term in the friction component of $A(x)$, and it is hybrid owing to the discrete gear selection j, which affects $B(j, x_2)$.

A common approach to managing the nonlinear characteristics of the model is to use a piecewise affine (PWA) approximation. This method divides the nonlinear state space into multiple regions, each represented by an affine segment, simplifying the system's representation. For the quadratic friction term in $A(x)$, we approximate the nonlinear friction using two PWA regions as shown in Figure 1.

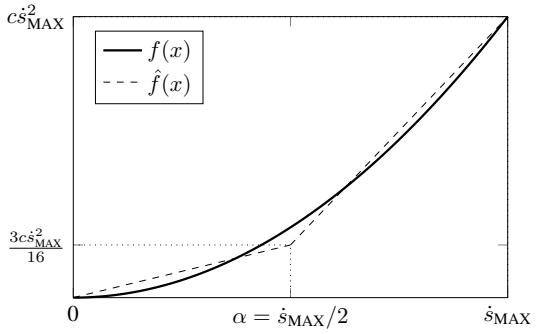


Fig. 1: PWA friction approximation. The solid line represents the true dynamics and the dashed line represents the piecewise approximation.

For the hybrid gear dynamics $B(j, x)$, we consider two approaches to approximation. In the first approach, gears are modeled as a function of vehicle velocity, resulting in a PWA approximation for $B(j, x)$. In the second approach, the gear choice is treated as a discrete decision variable, formulated using a mixed-logical-dynamical (MLD) representation. Figure 2a illustrates the complete gear model, while Table IX lists the maximum traction force for each gear along with the velocity ranges within which the maximum traction remains constant.

1) PWA gear approximation: To approximate the gear dynamics with a PWA approach, the velocity range is divided into segments, with each segment assigned a specific gear. This creates a one-to-one mapping between velocity segments and gear selection. The PWA gear approximation eliminates the discrete decision variable j, making it a function of the state x. Figure 2b dipicts this gear approximation.

2) Discrete input gear approximation: The second gear model treats gear selection as a discrete input. Similar to the PWA approach, the traction curves are restricted to regions of constant traction for each gear, and each gear is limited to operate only within its defined region. However, unlike the PWA model, the mapping between velocity segments and gear is not one-to-one. Figure 2c depicts this gear approximation.

A detailed representation of the approximations is explained in [6].

B. Problem Formulation

This work addresses the control of a single vehicle tasked with following a given reference trajectory, denoted as r. The objective is to minimize a cost that considers tracking accuracy and fuel consumption while satisfying constraints related to vehicle dynamics and safety requirements. The vehicle's states, denoted as x, include position x_1 and velocity x_2. The control inputs comprise the throttle input u and gear choice j.

To ensure safety and comfort, the states and inputs of the vehicle are subject to the following constraints. The velocity and acceleration are constrained as:

$$\begin{aligned} &v_{\text{MIN}} \leq x_2(k) \leq v_{\text{MAX}}, \\ &a_{\text{MIN}}T \leq x_2(k+1) - x_2(k) \leq a_{\text{MAX}}T, \end{aligned} \tag{4}$$

where v_{MIN}, v_{MAX}, a_{MIN}, and a_{MAX} are the bounds on velocity and acceleration. The normalized throttle input is constrained as:

$$u_{\text{MIN}} \leq u(k) \leq u_{\text{MAX}}. \tag{5}$$

The performance cost, J_{total}, is defined as a weighted sum of tracking accuracy and fuel consumption:

$$J_{\text{total}} = w_1 \cdot J_{\text{tracking}} + w_2 \cdot J_{\text{fuel}}, \tag{6}$$

where J_{tracking} quantifies tracking accuracy as:

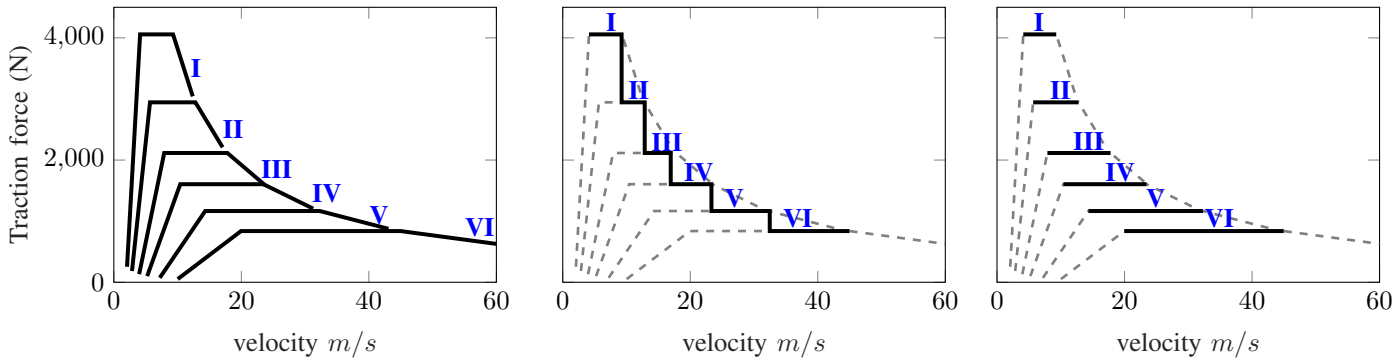
$$J_{\text{tracking}} = \|x(k) - r(k)\|_{Q_x}, \tag{7}$$

and J_{fuel} represents fuel consumption, calculated using a polynomial fuel consumption model:

$$J_{\text{fuel}} = Q_f \left(\sum_{m=0}^{3} b_m x_2^m(k) + \frac{x_2(k+1) - x_2(k)}{T} \sum_{l=0}^{2} c_l x_2^l(k) \right), \tag{8}$$

where b_m and c_l are coefficients derived from experimental data. Note that during braking ($u < 0$), it is assumed that no fuel is consumed due to energy recycling.

While standard MPC can solve this optimization problem using mixed-integer nonlinear optimization, the inclusion of integer variables and the fuel consumption model makes the problem non-convex and hybrid, significantly increasing computational complexity. This study aims to simplify the hybrid MPC problem and enable real-time implementation while maintaining performance comparable to the original formulation.



(a) Full (b) PWA (c) Discrete input

Fig. 2: Gear Approximations

III. Proposed MPC-RL Approach

To overcome the challenges outlined in section II-B, we employ an RL-based approach to handle most of the discrete components, such as gear selection, and to directly manage the fuel consumption cost. This allows the MPC to focus solely on computing the continuous throttle input and optimizing the tracking cost, significantly simplifying the optimization problem. As a result, the computational time can be significantly reduced, enabling real-time implementation.

Illustrated as Figure 3, in this study we propose a composite control framework where an RL agent determines the gear of the vehicle, thereby simplifying the original mixed-integer optimization problem. Although some discrete variables remain, the reduction achieved enables a much faster and more efficient MPC solution for the remaining control problem.

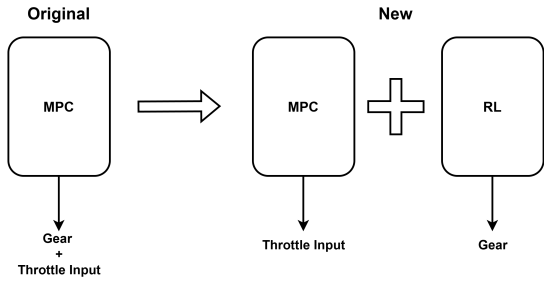


Fig. 3: Controller Design. The left panel depicts the standard MPC approach, where the controller determines both the discrete gear selection and continuous throttle input. The right panel illustrates the proposed MPC-RL approach, where MPC determines the throttle input and RL handles the gear selection.

The integration of RL for discrete decision-making (gear selection) and MPC for continuous control (throttle management) leverages the strengths of each approach: RL brings fast inference and efficiency to gear selection, while MPC ensures optimal and safe control actions. The subsequent sections will detail the design of the gear controller, the vehicle throttle controller, and the structure of the overall control loop.

A. Vehicle Throttle MPC Controller Design

In the MPC controller of our approach, the discrete input gear approximation, as described in Section II-A2, is utilized to represent the gear model. The states and input constraints, as outlined in Section II-B, define the operational limits of the system. The MPC optimization objective includes both tracking and control costs, evaluated using norm penalties. The optimization problem is formulated as follows:

$$
\begin{aligned}
J(x,\mathbf{j}) = \min_{\mathbf{u},\mathbf{x}} & \underbrace{\sum_{k=0}^{N+1} \|x(k) - r(k)\|_{Q_x}}_{\text{Tracking Cost}} \\
& + \underbrace{\sum_{k=0}^{N} \|u(k)\|_{Q_u}}_{\text{Control Cost}} \\
& \text{s.t. (4) - (5)} \\
& \text{vehicle model} \\
& x(0) = x
\end{aligned}
\tag{9}
$$

where the decision variables $\mathbf{u}$ and $\mathbf{x}$ represent the control and state trajectories over the prediction horizon. Notably, the gear selection sequence $\mathbf{j}$ is not included as a decision variable in the optimization problem but is instead treated as an input. It is determined by the RL controller and fixed for the vehicle model during the optimization process.

At each time step, the MPC generates a control sequence $\mathbf{u} = (u(0), \cdots, u(N-1))$ by solving the optimization problem defined over the prediction horizon N, adhering to the given constraints and vehicle model. The initial element of this control sequence, $u(0)$, is then applied to the system, and the optimization process is repeated at the next time step in a receding horizon manner.

B. Vehicle Gear RL Controller Design

1) State and Action Space:

a) State Space: At each time step, the MPC controller provides the RL controller with predicted states and control inputs over a horizon of length N, including position x_1,

velocity x_2, throttle input u, and gear choice j. To enhance decision-making, the absolute position is replaced with the tracking error:

$$e(k) = x_1(k) - r(k), \tag{10}$$

while the velocity x_2 is normalized to:

$$v^{\text{norm}}(k) = \frac{x_2(k) - v_{\text{MIN}}}{v_{\text{MAX}} - v_{\text{MIN}}}. \tag{11}$$

The pre-processed state matrix utilized by the RL controller is:

$$\mathbf{s}'(k) = \begin{bmatrix} e(k) & \dots & e(k+N-1) \\ v^{\text{norm}}(k) & \dots & v^{\text{norm}}(k+N-1) \\ u(k) & \dots & u(k+N-1) \\ j(k) & \dots & j(k+N-1) \end{bmatrix}. \tag{12}$$

b) Action Space: The action space consists of three possible gear operations: up-shift, down-shift, and no shift:

$$\mathcal{A} = \{\text{upshift}, \text{downshift}, \text{no shift}\}. \tag{13}$$

This formulation reduces the size of the action space, improving learning efficiency and ensuring smoother transitions between gears. By focusing on relative gear changes, the RL controller handles sequential dependencies more effectively.

2) Reward Function: The reward function is designed to prioritize feasibility of the gear sequence selected by the RL controller, ensuring compatibility with the MPC controller. Feasibility is critical for maintaining MPC solvability and system stability. Two scenarios can lead to infeasibility:

1) **Exceeding Gear Limits**: The RL controller selects a gear shift beyond the allowable range (e.g., upshift at maximum gear or downshift at minimum gear).
2) **Velocity-Gear Mismatch**: The selected gear choice results in a velocity incompatible with the chosen gear, making the MPC optimization unsolvable (Figure 2c).

To address these issues, infeasible gear choices are penalized heavily in the reward function. In addition to feasibility, the reward function incorporates fuel consumption cost and tracking error. Fuel consumption is impacted by gear selection, while tracking error plays a secondary role, given its stronger dependence on the MPC.

The weighting strategy ensures that feasibility is the highest priority, followed by fuel optimization and tracking performance. This balance enables the RL controller to support the MPC effectively, achieving stable training and desirable system performance.

3) RL Algorithm and Network Architecture: To handle the discrete nature of the action space, DQN is selected for its efficiency and stability in discrete decision-making. DQN learns a Q-value function $Q(s, a)$ that estimates the expected cumulative reward for each action given a state, making it ideal for the gear shift problem with a fixed and finite action set. Mechanisms like experience replay and target networks enhance sample efficiency and stabilize training by reducing correlations in updates and preventing large fluctuations in Q-values.

Given that the input states $\mathbf{s}'$ are sequential data over a horizon N, capturing temporal dependencies is essential for accurate gear selection. To achieve this, a bidirectional recurrent neural network (BiRNN) is integrated into the DQN architecture. BiRNN processes predicted states in both forward and backward directions, enabling the RL controller to consider both preceding and subsequent states within the horizon simultaneously. This is particularly important for the gear shift problem, where decisions depend on both immediate and future system dynamics.

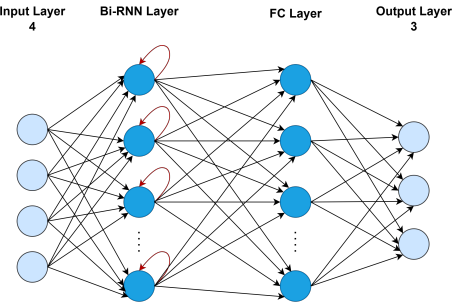


Fig. 4: Network Architecture

As illustrated in Figure 4, the network architecture consists of:

- **Input Layer**: Takes a vector of dimension 4 (tracking error, normalized velocity, throttle, and gear) as input.
- **BiRNN Layer**: Captures temporal dependencies in the input sequence over horizon N. The red arrows in the diagram indicate the recurrent connections within the Bi-RNN layer, which propagate the hidden states through time.
- **Fully Connected Layer**: Maps hidden states from the BiRNN to discrete action outputs (up-shift, down-shift, no shift) for each time step.

The BiRNN operates in a many-to-many configuration, generating a sequence of gear shift decisions $\mathbf{j}$ corresponding to the MPC-predicted states. By capturing temporal patterns in both directions, the architecture ensures informed and efficient gear selection, optimizing for feasibility, tracking accuracy, and fuel efficiency. This combination of DQN with BiRNN provides a efficient framework for sequential decision-making in hybrid control systems.

C. Control Loop

Figure 5 illustrates the control loop of the proposed method. At each time step k, the predicted sequences $\mathbf{s}(k-1)$ from the MPC controller at the previous time step is preprocessed by the preprocessing block to generate the input $\mathbf{s}'(k-1)$ for the RL controller. The RL controller takes $\mathbf{s}'(k-1)$ as input and outputs a sequence of gear shift decisions $\Delta\mathbf{j}(k)$.

The postprocessing block converts $\Delta\mathbf{j}(k)$ into a specific gear sequence $\mathbf{j}(k)$, which serves as an input to the MPC controller. At the same time, the MPC controller receives the current observation $x(k)$ from the environment along with

the reference input. Using $\mathbf{j}(k)$ and $x(k)$, the MPC controller computes the optimal throttle input $\mathbf{u}(k)$.

Finally, the first elements of the predicted control sequences, $u(k)$ and $j(k)$, are applied to the plant. The updated prediction sequences $\mathbf{s}(k)$ from the MPC controller is then used as input for the next time step, ensuring the closed-loop control operation.

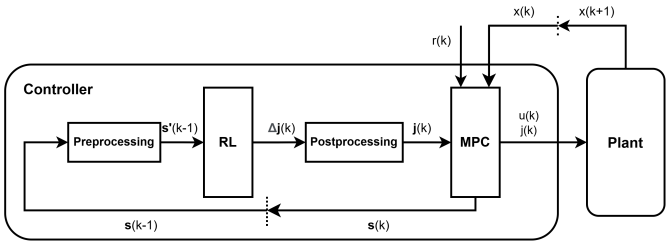


Fig. 5: Control loop at time step k, illustrating the interaction between the RL and MPC controllers for gear shift decision-making and throttle optimization.

IV. RL Training Scheme

The training is performed episode by episode, where each episode contains N_{E} time steps. The detailed training loop within an episode will be introduced in the following.

A. Step 0: Initialization

At the start of each training episode, the reference trajectory and vehicle states are initialized. Since the RL controller relies on predicted states from the MPC, which are unavailable at the initial time step, the initial gear is determined using a PWA approximation based on the initial velocity. This gear, denoted as $j(0)$, is uniformly applied across the prediction horizon:

$$\mathbf{j}(0) = \begin{bmatrix} j(0) & j(0) & \cdots & j(0) \end{bmatrix}_{1\times N}. \tag{14}$$

The gear sequence is used by the MPC to solve the optimization problem $J(x, \mathbf{j})$, producing the predicted state sequence $\hat{\mathbf{x}}(0)$ and control sequence $\hat{\mathbf{u}}(0)$. The environment then steps forward using the first control input, yielding the new observation $x(1)$.

This initialization ensures that the RL controller has the necessary predicted sequences to interact effectively with the MPC from the beginning, facilitating a smooth start to the training process.

B. Step 1: Gear Selection

With the predicted states and control sequence at time step $k-1$, the RL controller determines the gear sequence for time step k. The network outputs Q-values for the three possible gear shifts, and the gear shift is selected based on the index with the highest Q-value. The selected gear shift as Δj is calculated as:

$$\Delta j(k) = \arg \max_{i\in\{0,1,2\}} Q(s^{'}(k-1), i) - 1 \tag{15}$$

where $Q_i(s^{'}(k-1))$ represents the Q-value corresponding to gear shift option i, as predicted by the Q-network. The selected gear shift Δj is determined by finding the index i that yields the maximum Q-value and then subtracting 1 to map it to the appropriate gear shift value. The gear shift is defined as follows:

$$\Delta j = \begin{cases} -1, & \text{downshift} \\ 0, & \text{no shift} \\ 1, & \text{upshift} \end{cases} \tag{16}$$

This formulation ensures that the selected gear shift accurately reflects the intended action: downshift, no shift, or upshift, providing a clear and consistent mapping between the Q-network outputs and the actual gear control decisions. The gear shift sequence $\Delta\hat{\mathbf{j}}$ at time step k is defined as:

$$\Delta\hat{\mathbf{j}}(k) = \begin{bmatrix} \Delta\hat{j}(k) & \Delta\hat{j}(k+1) & \cdots & \Delta\hat{j}(k+N-1) \end{bmatrix} \tag{17}$$

Then, we calculated the explicit gear sequence $\hat{\mathbf{j}}(k)$ using the gear shift sequence $\Delta\hat{\mathbf{j}}(k)$. Within the prediction horizon, each future gear is determined sequentially based on the previous gear value and the corresponding gear shift at that time step. The predicted gear sequence is then expressed as

$$\hat{\mathbf{j}}(k) = \begin{bmatrix} \hat{j}(k) & \hat{j}(k+1) & \cdots & \hat{j}(k+N-1) \end{bmatrix} \tag{18}$$

The individual gears in the sequence are calculated iteratively as follows:

$$\hat{j}(k+i) = \begin{cases} j(k-1) + \Delta\hat{j}(k+i), & i = 0 \\ \hat{j}(k+i-1) + \Delta\hat{j}(k+i), & i = 1, \cdots, N-1 \end{cases} \tag{19}$$

To balance exploration and exploitation during training, an ϵ-greedy exploration mechanism is used. The exploration probability ϵ starts high and gradually decays over time, allowing the RL controller to explore random actions initially and increasingly rely on the learned policy as training progresses. At each time step, the controller explores by selecting a random gear sequence with probability ϵ. Otherwise, it exploits by choosing the gear sequence that maximizes the Q-value with probability $1-\epsilon$.

C. Step 2: Feasibility Check & Solving MPC Problem

After generating the predicted gear sequence, its feasibility is verified to ensure compatibility with vehicle dynamics. As discussed in Section III-B2, infeasibility may occur due to exceeding gear limits or a mismatch between velocity and gear.

The first step is to adjust any gear values that exceed the allowable range (1 to 6) by clipping them to the nearest limit. This adjusted sequence is then provided to the MPC solver. If the solver identifies the sequence as infeasible, **Backup Solution 1** is applied: the gear sequence from the previous time step is shifted forward, with the last gear value repeated to complete the horizon. This provides a simple and efficient fallback mechanism but may not always guarantee feasibility.

If **Backup Solution 1** also fails, **Backup Solution 2** is used. In this approach, the gear corresponding to the current velocity is determined using the PWA gear approximation and applied uniformly across the entire prediction horizon. While this conservative solution sacrifices optimality, it ensures that a valid gear sequence is available for the MPC to proceed.

The adjusted gear sequence is then provided to the MPC controller, which solves the optimization problem to generate the predicted state sequence $\hat{\mathbf{x}}(k)$ and control sequence $\hat{\mathbf{u}}(k)$. These predictions are subsequently used for the next steps in the decision-making process.

D. Step 3: Interaction with Environment

After the MPC controller solves the optimization problem, the first throttle input $u(k)$ of the predicted control sequence $\mathbf{u}(k)$, combined with the first gear choice $j(k)$ of the gear sequence $\mathbf{j}(k)$, is applied to control the vehicle in the environment.

The environment then transitions to a new state $x(k+1)$ based on this control input, following the system dynamics introduced in (2).

E. Step 4: Reward Calculation

As discussed in Section III-B2, the reward function is composed of three components: the tracking cost, the fuel cost, and the penalty cost associated with gear infeasibility. At time step k, these components are calculated as follows:

1) Tracking Cost (J_{tracking}): The tracking cost quantifies the deviation of the current state $x(k)$ from the reference trajectory $r(k)$, which is calculated as (7).
2) Fuel Cost (J_{fuel}): The fuel cost is computed based on the vehicle's velocity $x_2(k)$. It is given by Equation 8.
3) Penalty Cost (J_{penalty}): The penalty cost accounts for violations in the gear sequence and MPC infeasibility. At each time step, the number of gear limit violations is denoted as n_{penalty}, with each violation penalized by ϕ_1. Additionally, a binary variable λ_{penalty} indicates whether the MPC problem is infeasible, incurring a penalty of ϕ_2 if set to 1. The variable λ_{penalty} is defined as:

$$\lambda_{\text{penalty}} = \begin{cases} 1, & \text{if MPC is infeasible} \\ 0, & \text{if MPC is feasible} \end{cases} \tag{20}$$

The total penalty P is then calculated as:

$$J_{\text{penalty}} = \phi_1 \cdot n_{\text{penalty}} + \phi_2 \cdot \lambda_{\text{penalty}} \tag{21}$$

Finally, the reward at time step k is computed as:

$$R(k) = -\big(l_1 \cdot J_{\text{tracking}}(k) + l_2 \cdot J_{\text{fuel}}(k) + l_3 \cdot J_{\text{penalty}}(k)\big), \tag{22}$$

where l_1, l_2, and l_3 are weights used to balance the contributions of tracking cost, fuel cost, and penalty cost in the reward function. Note that the negative sign in the reward function is used because DQN aims to maximize the reward.

F. Step 5: Experience Storage

At the end of each time step, the experience tuple $\Big(\hat{\mathbf{s}}(k-1), \Delta\hat{\mathbf{j}}(k), \hat{\mathbf{s}}(k), R(k)\Big)$ is stored in the replay buffer.

To facilitate training, feasibility checks on actions (introduced in **Step 2**) are performed to ensure the MPC problem can be solved. However, only the original network outputs are stored in the replay buffer, as penalty costs are computed based on these outputs, reflecting the agent's true decision-making process.

G. Step 6: Batch Training and Network Update

During training, a batch of transitions is randomly sampled from the replay buffer to break correlations between consecutive samples, enhancing the RL agent's ability to generalize. Since the states and actions are sequences while the reward is scalar, the reward is broadcast across the prediction horizon to ensure consistency in training and alignment with the sequence data.

The target Q-value is computed using the reward and the discounted maximum Q-value of the next state, while the current Q-value is updated using Huber loss to handle outliers. The Adam optimizer minimizes the loss for stable training. Additionally, the target network is updated through a soft update mechanism, gradually blending the policy network parameters into the target network based on a predefined rate τ, ensuring training stability.

V. Training and Evaluation Results

In this section, we present the results of our research, including the training outcomes and evaluation results for both individual episodes and a large-scale test.

A. Training Results

The training comprises 50,000 episodes, each lasting 60 time steps. A new reference trajectory is generated for every episode, with the vehicle starting from randomized states. The specific training parameters are presented below, followed by the corresponding training results.

The details of the network is outlined in Table X. Additionally, Table XI summarizes the parameters employed by the MPC controller in our approach (9), and Table XII lists the parameters used to train the RL agent.

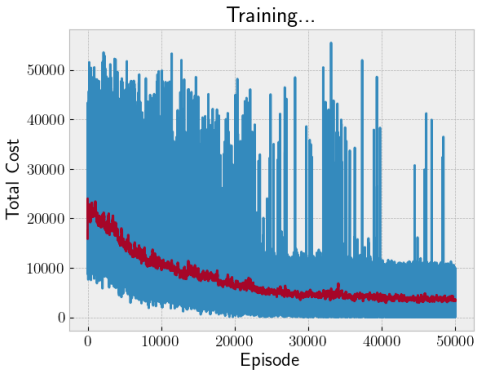
Several key metrics are tracked over 50,000 training episodes, as shown in Figure 6, including total cost, tracking cost, fuel cost, and penalty cost.

The total cost (Figure 6a) starts high due to the agent's exploration phase and gradually stabilizes after 40,000 episodes, with fluctuations diminishing as the agent learns an effective policy. Penalty cost (Figure 6d), a major contributor to the total cost, follows a similar trend, reflecting the agent's improved ability to avoid gear limit violations and MPC infeasibilities.

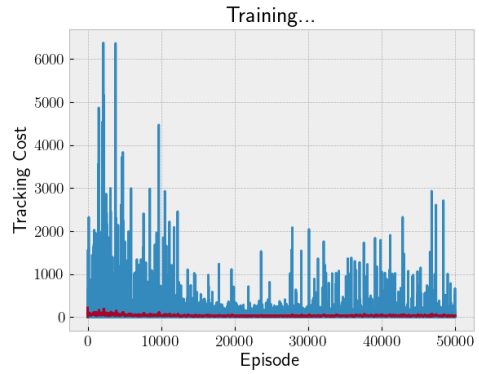
Tracking cost (Figure 6b) remains consistently low throughout training, with minimal variability. This stability highlights the robustness of the MPC controller in maintaining tracking performance, even when gear selections are suboptimal.

Fuel cost (Figure 6c) shows a slight declining trend during training, but it fluctuates due to variations in the velocity profiles across different episodes. Since the chosen fuel model primarily depends on vehicle velocity rather than gear selection, optimizing gear shifts has a minor impact on fuel efficiency. As a result, fuel cost contributes relatively little to the overall cost metric during training.

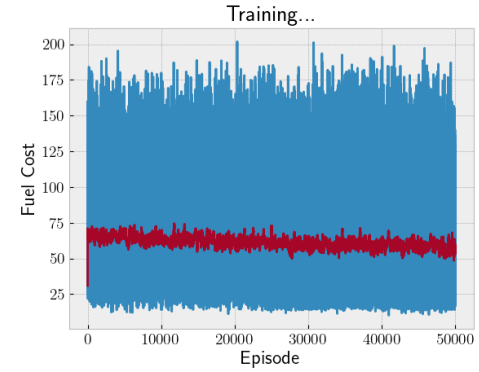
These results demonstrate the RL agent's effectiveness in reducing overall cost metrics, particularly by minimizing penalties and improving gear feasibility.



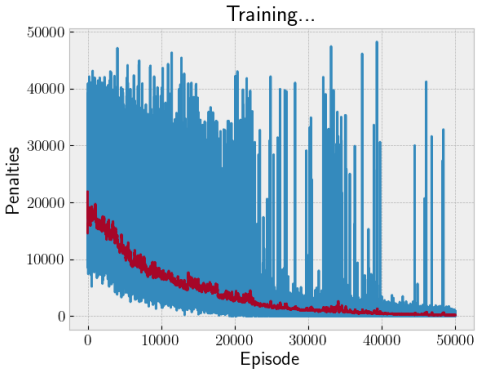
(a) Total Cost



(b) Tracking Cost



(c) Fuel Cost

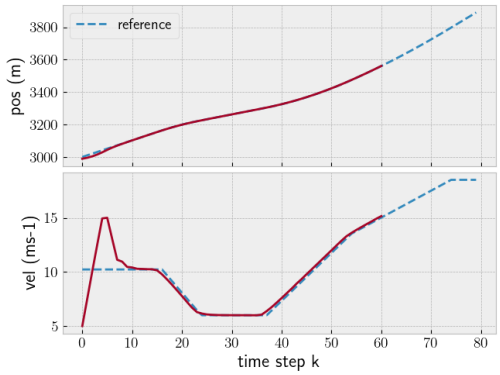


(d) Penalty Cost

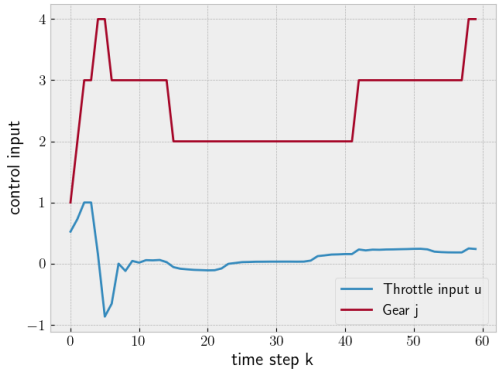
Fig. 6: Training results. In each plot, the blue line represents the metric's value for each individual episode, while the red line shows its 100-episode running average.

B. Evaluation Results

To evaluate our approach, we conducted a simulation with a single reference trajectory to demonstrate the RL's capability in managing gear selections. Following this, we performed a larger test consisting of 100 episodes to comprehensively analyze performance metrics. Each episode in the larger test consists of 60 time steps.



(a) Tracking Results



(b) Control Inputs

Fig. 7: Individual Test Results

1) Individual Test: The simulation results are shown in Figure 7. In Figure 7a, the upper panel depicts the position, and the lower panel shows the velocity. The dotted blue line represents the reference trajectory, while the solid red line indicates the vehicle's actual trajectory. Starting from initial states deviating from the reference, the vehicle quickly converges to the trajectory within a few steps, highlighting the effectiveness of the MPC-RL controller in achieving accurate tracking.

Figure 7b presents the control inputs during the simulation. The red line represents the gear selected by the RL controller, and the blue line shows the throttle input from the MPC controller. The RL agent efficiently selects gears to match the vehicle's velocity, ensuring smooth transitions and accurate tracking. For this trajectory, with reference velocities between 5-15 m/s, the RL controller primarily operates in lower gears (1 to 3), aligning well with the velocity profile and system constraints.

Table I summarizes the performance metrics for this test. Notably, there are no gear limit violations or MPC infeasibilities, indicating that the RL controller adheres to the defined constraints and provides feasible gear selections.

The runtime of the simulation validates the real-time feasibility of the proposed MPC-RL controller. The total runtime is 0.38 seconds, with an average runtime per time step of approximately 0.0063 seconds, corresponding to a control frequency of around 159 Hz. This high control frequency is suitable for high-precision autonomous vehicle control in dynamic environments.

TABLE I: Performance Metrics for Individual Test

Metric	value
Runtime (s)	0.38
Performance Cost	41.57
Tracking Cost	1121.75
Fuel Cost	38.77
Gear Limit Violations	0
MPC Infeasibilities	0

2) Large Tests: We conducted a large-scale test consisting of 100 episodes, and the average performance metrics are summarized in Table II.

TABLE II: Average Performance Metrics Over 100 Test Episodes

Metric	Value
Average Runtime	0.35
Average Performance	99.62
Average Tracking Cost	19764.77
Average Fuel Cost	50.20

We analyzed the occurrence of gear limit violations and MPC infeasibilities across 100 test episodes, recording the number of such events per episode. A single gear limit violation was observed, and only one MPC infeasibility occurred. These isolated events demonstrate that the RL controller effectively learned and applied a policy that generalizes well to diverse scenarios, providing feasible gear selections for the MPC controller in nearly all episodes. Furthermore, in the rare episodes where infeasibilities occurred, the backup solution successfully addressed the issues, ensuring the solvability of the MPC problem and maintaining the continuity of the control process.

VI. Comparative Analysis

In this section, we compare our approach with the standard MPC and PID methods by performing large-scale test.

A. Standard MPC Approach

The standard MPC approach optimizes all objectives simultaneously by solving for all decision variables within the model. For this comparison, we utilized the PWA model for the gear model (II-A1). As demonstrated in [6], this test shows that while the PWA model's performance is only slightly worse compared to the discrete input model, it significantly reduces computation time.

1) Optimization Problem: The standard MPC optimization objective consists of three components: tracking cost, fuel cost, and control cost. The decision variables include both the vehicle's throttle input $\mathbf{u}$ and gear selection $\mathbf{j}$, with the gear selection implicitly incorporated into the optimization problem through the vehicle model, which includes the gear model. The MPC optimization problem is formulated as:

$$
\begin{aligned}
J(x) = \min_{\mathbf{u},\mathbf{j},\mathbf{x}} & \underbrace{\sum_{k=0}^{N+1} \|x(k) - r(k)\|_{Q_x}}_{\text{Tracking Cost}} \\
& \underbrace{+ Q_f \sum_{k=0}^{N} \left(\sum_{m=0}^{3} b_m x_2^m(k) + \frac{x_2(k+1) - x_2(k)}{T} \sum_{l=0}^{2} c_l x_2^l(k) \right)}_{\text{Fuel Cost}} \\
& \underbrace{+ \sum_{k=0}^{N} \|u(k)\|_{Q_u}}_{\text{Control Cost}} \\
& \text{s.t. (4) - (5)} \\
& \text{vehicle model} \\
& x(0) = x
\end{aligned}
\tag{23}
$$

The parameters for standard MPC approach are the same as Table XI, with an additional fuel cost weight Q_f due to the inclusion of fuel consumption cost into the MPC problem. The coefficients of the polynomial in fuel consumption are listed in Table XIII.

2) Large Tests: Large-scale tests were conducted similar to those described in Section V-B2. To achieve the lowest average performance cost, the standard MPC was optimized by exploring various combinations of prediction horizon N and fuel cost weight Q_f, with N ranging from 2 to 5 and Q_f ranging from 1 to 5. Each combination was evaluated over 100 episodes.

TABLE III: Standard MPC Large Tests ($N = 2$)

Q_f	1	2	3
Average Runtime	356.29	373.62	382.03
Average Performance	101.85	101.43	101.31
Average Tracking Cost	18867.31	18945.19	19093.79
Average Fuel Cost	54.68	54.07	53.57

TABLE IV: Standard MPC Large Tests ($N = 3$)

Q_f	1	2	3
Average Runtime	744.03	775.04	783.17
Average Performance	98.53	98.36	98.22
Average Tracking Cost	18353.05	18354.14	18364.11
Average Fuel Cost	52.65	52.48	52.31

Tables III, IV, and V present the results for different N and Q_f combinations. However, due to the increasing complexity of the optimization problem with larger N and Q_f values, the MPC solver failed to converge for certain configurations.

TABLE V: Standard MPC Large Tests ($N = 4$)

Q_f	1
Average Runtime	1926.66
Average Performance	97.28
Average Tracking Cost	18238.11
Average Fuel Cost	51.68

Consequently, results are only available for a subset of the tested combinations.

From the results in Tables III, IV, and V, it is evident that increasing the prediction horizon N results in a reduction in the performance cost. However, this improvement comes at the cost of significantly higher average runtimes per episode. Furthermore, increasing the fuel cost weight Q_f slightly reduces the performance cost across all prediction horizons but also leads to a minor increase in runtime, likely due to the added complexity of the optimization problem.

B. PID Approach

The PID approach uses proportional, integral, and derivative components to compute control outputs based on the current error, the accumulated error over time, and the rate of change of error, respectively. This mechanism highlights simplicity and enables fast control.

1) Controller Design: The control strategy employs two separate PID controllers to regulate the vehicle's acceleration based on position and velocity errors. The desired acceleration is calculated by combining the outputs of these controllers.

The position PID controller computes acceleration based on the position error e_{pos} as:

$$a_{\text{des,pos}} = k_p^{\text{pos}} e_{\text{pos}} + k_i^{\text{pos}} \int e_{\text{pos}} dt + k_d^{\text{pos}} \frac{de_{\text{pos}}}{dt}, \tag{24}$$

where $k_p^{\text{pos}}, k_i^{\text{pos}}, k_d^{\text{pos}}$ are the proportional, integral, and derivative gains.

The velocity PID controller computes acceleration based on the velocity error e_{vel} as:

$$a_{\text{des,vel}} = k_p^{\text{vel}} e_{\text{vel}} + k_i^{\text{vel}} \int e_{\text{vel}} dt + k_d^{\text{vel}} \frac{de_{\text{vel}}}{dt}, \tag{25}$$

where $k_p^{\text{vel}}, k_i^{\text{vel}}, k_d^{\text{vel}}$ are the corresponding gains for velocity control.

The outputs of the two controllers are combined to compute the desired acceleration a_{des} as:

$$a_{\text{des}} = \text{clip}\left(w_{\text{pos}} \cdot a_{\text{des,pos}} + w_{\text{vel}} \cdot a_{\text{des,vel}}, a_{\text{MIN}}, a_{\text{MAX}}\right), \tag{26}$$

where w_{pos} and w_{vel} are the weights for position and velocity controller outputs, respectively. The `clip` function ensures a_{des} remains within the range $[a_{\text{MIN}}, a_{\text{MAX}}]$.

Finally, the desired throttle input is determined using vehicle dynamics:

$$u = \text{clip}\left(\frac{1}{b(j, x_2)}(ma_{\text{des}} + cx_2^2 + \mu mg), u_{\text{MIN}}, u_{\text{MAX}}\right), \tag{27}$$

where u_{MIN} and u_{MAX} are the throttle input bounds.

The parameter values used in the tests are listed in Table XIV.

2) Large Tests: The large-scale tests using the PID controller were conducted, with the results presented in Table VI. It is worth noting that the PID approach demonstrates a favorable trade-off between runtime and performance cost, with an average runtime of only 0.02 seconds per episode and a slightly worse performance cost compared to standard MPC and our MPC-RL approaches. The average fuel cost is notably lower compared to the standard MPC and our MPC-RL approaches. This is likely attributed to the PID controller's lower emphasis on velocity control, resulting in reduced vehicle velocities and consequently lower fuel consumption.

Although the PID approach demonstrates reasonable performance, its limitations in flexibility and handling constraints restrict its applicability. These limitations will be further discussed in the next section.

TABLE VI: PID Controller Large Tests

Metric	**Value**
Average Runtime	0.02
Average Performance	100.46
Average Tracking Cost	21456.30
Average Fuel Cost	46.82

C. Comparison & Discussion

We compared the MPC-RL approach, the standard MPC approach with $N = 4$ and $Q_f = 1$, and the PID approach under large-scale testing. The results are summarized in Table VII.

The standard MPC approach achieves the lowest performance cost, showcasing its strong optimization capability by explicitly incorporating all decision variables within a single optimization problem. However, this comes at the expense of an extremely high runtime, averaging 1926.66 seconds per episode, making it impractical for real-time applications. The excessive computational demand severely limits its feasibility in dynamic scenarios requiring rapid decision making.

In contrast, the PID controller achieves the fastest runtime, averaging just 0.02 seconds per episode. While this makes it computationally efficient, its performance is inferior to both MPC-RL and standard MPC. Additionally, the parameters may need to be retuned to achieve optimal performance when encountering different reference trajectories because the PID controller relies on fixed gain values which are calibrated for specific conditions, highlighting the lack of flexibility in the PID approach compared to our method. Moreover, the PID controller lacks the capability to handle constraints effectively, which can result in unsafe or suboptimal behavior. While input constraints can be managed through clipping, state constraints, such as the vehicle velocity limit, are not inherently guaranteed. In more complex scenarios, such as vehicle platooning, hard constraints like maintaining a safe inter-vehicle distance cannot be reliably enforced using the PID approach. This limitation in flexibility and constraint management restricts its applicability.

The MPC-RL approach effectively overcomes the drawbacks of both PID and standard MPC approaches. It provides a

near-optimal solution while achieving a fast average runtime of 0.35 seconds per episode, making it highly suitable for real-time control applications. Unlike PID, which lacks the ability to handle constraints, MPC-RL incorporates constraint-handling capabilities, ensuring safe and feasible control actions. At the same time, it avoids the excessive computational burden of standard MPC by excluding gear selection and fuel optimization from the optimization problem, offering a practical balance between performance and efficiency for autonomous vehicles in dynamic environments.

TABLE VII: Comparison of MPC-RL, Standard MPC, and PID Approaches

Metric	MPC-RL	Standard MPC	PID
Average Runtime (s)	0.35	1926.66	0.02
Average Performance	99.62	97.28	100.46
Average Tracking Cost	19764.77	18238.11	21456.30
Average Fuel Cost	50.20	51.68	46.82

VII. Conclusion

This research introduces a novel MPC-RL framework that integrates RL with MPC to address computational challenges in autonomous vehicle control. By delegating discrete gear selection to the RL controller and continuous throttle optimization to MPC, the framework significantly reduces computational complexity, enabling real-time implementation with near-optimal performance.

Leveraging a DQN with a bidirectional RNN, the RL controller effectively captures temporal dependencies to generate feasible gear decisions, while MPC ensures trajectory tracking and constraint satisfaction. The framework achieves a low average runtime, balancing computational efficiency and performance. Compared to standard MPC, the MPC-RL framework offers better efficiency with minimal performance trade-offs, and it outperforms PID controllers in flexibility and constraint handling.

Future work can explore advanced RL methods and extend the framework to vehicle platoons using Distributed MPC. This research demonstrates the potential of MPC-RL integration as a promising approach to hybrid control in autonomous driving.

References

[1] E. Shaw and J. K. Hedrick, "String stability analysis for heterogeneous vehicle strings," in 2007 American Control Conference, IEEE, 2007, pp. 3118–3125.

[2] G. J. L. Naus, R. P. A. Vugts, J. Ploeg, M. J. G. van De Molengraft, and M. Steinbuch, "String-stable CACC design and experimental validation: A frequency-domain approach," IEEE Transactions on Vehicular Technology, vol. 59, no. 9, pp. 4268–4279, 2010.

[3] Y. Wu, S. E. Li, Y. Zheng, and J. K. Hedrick, "Distributed sliding mode control for multi-vehicle systems with positive definite topologies," in Proc. 2016 IEEE 55th Conference on Decision and Control (CDC), 2016, pp. 5213–5219.

[4] Y. Zheng, S. E. Li, K. Li, and W. Ren, "Platooning of connected vehicles with undirected topologies: Robustness analysis and distributed H-infinity controller synthesis," IEEE Trans. Intell. Transp. Syst., vol. 19, no. 5, pp. 1353–1364, 2017.

[5] J. Ploeg, D. P. Shukla, N. Van De Wouw, and H. Nijmeijer, "Controller synthesis for string stability of vehicle platoons," IEEE Trans. Intell. Transp. Syst., vol. 15, no. 2, pp. 854–865, 2013.

[6] S. Mallick, A. Dabiri, and B. De Schutter, "A comparison benchmark for distributed hybrid MPC control methods: Distributed vehicle platooning," arXiv preprint arXiv:2401.09878, 2024.

[7] V. Turri, B. Besselink, and K. H. Johansson, "Gear management for fuel-efficient heavy-duty vehicle platooning," in Proc. 55th IEEE Conf. Decision Control (CDC), 2016, pp. 1687–1694.

[8] D. He, T. Qiu, and R. Luo, "Fuel efficiency-oriented platooning control of connected nonlinear vehicles: a distributed economic MPC approach," Asian Journal of Control, vol. 22, no. 4, pp. 1628–1638, 2020.

[9] Y. Zheng, S. E. Li, K. Li, F. Borrelli, and J. K. Hedrick, "Distributed model predictive control for heterogeneous vehicle platoons under unidirectional topologies," IEEE Transactions on Control Systems Technology, vol. 25, no. 3, pp. 899–910, 2016.

[10] M. Hu *et al.*, "Fuel economy-oriented vehicle platoon control using economic model predictive control," IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 11, pp. 20836–20849, 2022.

[11] M. Li, Z. Cao, and Z. Li, "A reinforcement learning-based vehicle platoon control strategy for reducing energy consumption in traffic oscillations," IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 12, pp. 5309–5322, 2021.

[12] G. Li and D. Görges, "Ecological adaptive cruise control for vehicles with step-gear transmission based on reinforcement learning," IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 11, pp. 4895–4905, 2019.

Appendix A
Parameter Tables

In this appendix we list the tables with the parameters used in the paper.

TABLE VIII: Parameter values used in 1

Parameter	Value	Units
m	800	kg
c	0.5	kg/m
μ	0.01	-
g	9.8	m/s^2

TABLE IX: Maximum Traction Force and Velocity Range for Each Gear.

Gear j	Traction force $b(j)$ (N)	Min. vel. (m/s)	Max. vel. (m/s)
I	4057	3.94	9.46
II	2945	5.43	13.04
III	2116	7.56	18.15
IV	1607	9.96	23.90
V	1166	13.70	32.93
VI	838	19.10	45.84

TABLE X: Network Architecture Details

Layer	Number of Nodes
Input Layer	4
Bi-RNN Layer	2×64
Fully Connected Layer	128
Output Layer	3

TABLE XI: MPC Controller Parameters

Parameter	Value
Prediction Horizon (N)	5
Sampling Time (T)	1 s
State Weight Matrix (Q_x)	diag(1, 0.1)
Control Weight Matrix (Q_u)	1
Throttle Bounds ($u_{\text{MIN}}, u_{\text{MAX}}$)	[-1, 1]
Velocity Limits ($v_{\text{MIN}}, v_{\text{MAX}}$)	[3.94, 45.84]
Acceleration Limits ($a_{\text{MIN}}, a_{\text{MAX}}$)	[-2, 2.5]

TABLE XII: RL Training Parameters

Parameter	**Value**
Learning Rate	0.001
Discount Factor (γ)	0.9
Batch Size	128
Number of Episodes	50000
Steps per Episode (N_E)	60
Starting Exploration Rate (ϵ_{start})	0.999
Ending Exploration Rate (ϵ_{end})	0
Exploration Decay Factor (ϵ_{decay})	70000
Replay Buffer Size	100000
Target Network Soft Update Rate (τ)	0.001
Threshold of Huber Loss Function (δ)	1
Optimizer	Adam
Tracking Cost Weight (l_1)	0.001
Fuel Cost Weight (l_2)	1
Penalty Cost Weight (l_3)	1
Weights for Performance Cost (w_1, w_2)	0.0025, 1
Penalties for Gear Infeasibilities (ϕ_1, ϕ_2)	100, 50

TABLE XIII: Coefficients for Fuel Consumption

Parameter	**Value**
b_0, b_1, b_2, b_3	0.1569, $2.450 \cdot 10^{-2}$, $-7.415 \cdot 10^{-4}$, $5.975 \cdot 10^{-5}$
c_0, c_1, c_2	0.0724, $9.681 \cdot 10^{-2}$, $1.075 \cdot 10^{-3}$

TABLE XIV: PID Controller Parameters

Parameter	**Value**
k_p^{pos}	0.05
k_i^{pos}	0.01
k_d^{pos}	0.0
k_p^{vel}	0.7
k_i^{vel}	0.1
k_d^{vel}	0.0
w_{pos}	0.55
w_{vel}	0.45