Deep Reinforcement Learning for Tuning of Adaptive Model Predictive Control for Autonomous Driving*

1st Feras Hamadeh
Department of Electrical and Computer Engineering
RPTU University Kaiserslautern-Landau
Kaiserslautern, Germany
fhamadeh@rhrk.uni-kl.de

2nd Anas Abdelkarim
Interdisciplinary Center for Security, Reliability and Trust (SnT) and
Department of Electrical and Computer Engineering
Luxembourg University and *RPTU University Kaiserslautern-Landau*
Luxembourg, Luxembourg
anas.abdelkarim@uni.lu - abdelkarim@eit.uni-kl.de

3rd Amar Hamadeh
Department of Electrical and Computer Engineering
RPTU University Kaiserslautern-Landau
Kaiserslautern, Germany
ahamadeh@rhrk.uni-kl.de

4th Daniel Görges
Department of Electrical and Computer Engineering
RPTU University Kaiserslautern-Landau
Kaiserslautern, Germany
goerges@eit.uni-kl.de

5th Holger Voos
SNT and the Faculty of Science, Technology, and Medicine (FSTM)
University of Luxembourg
Luxembourg City, Luxembourg
holger.voos@uni.lu

***Abstract*—Model Predictive Control (MPC) has emerged as a pivotal technology for optimizing control tasks in autonomous driving, particularly within Adaptive Cruise Control (ACC) systems. However, the manual tuning of MPC cost function weights and prediction horizons remains a significant challenge. In this paper, we introduce a novel framework that combines Deep Reinforcement Learning (DRL) with MPC to dynamically tune both the weight parameters and prediction horizon in real time. This approach, referred to as the Weights and Prediction Horizon Varying MPC (W-PH-MPC), overcomes traditional MPC limitations by utilizing proximal Policy optimisation and Deep Deterministic Policy Gradient (DDPG) algorithms to adjust control parameters. We evaluate the effectiveness of our approach through simulations in vehicle-tracking scenarios. Simulation results show that the adaptive MPC–RL controller achieves better tracking performance, without compromising power consumption, and lowers longitudinal jerk compared to a fixed-parameter MPC baseline, resulting in smoother and more efficient vehicle behavior.**

I. Introduction

Model Predictive Control (MPC) has been widely used in Adaptive Cruise Control (ACC) systems. For example, [1] introduced an MPC-based Energy Adaptive Cruise Control (EACC) approach to enhance energy efficiency and driving comfort. Inspired by this, our work advances MPC adaptability by integrating reinforcement learning (RL) for automatic weight tuning. MPC computes control variables at each time step by solving an optimization problem, defined by a cost function and a set of constraints. MPC can be efficiently implemented and solved using high-level model languages, equipped with advanced solvers, such as AMPL [2], [3] or using custom solvers such as [4], [5]. However, selecting appropriate weighting matrices for the cost function remains a challenging and time-consuming task that demands expert knowledge. Automating this tuning process is highly desirable as control tasks become increasingly complex in automated driving.

Various methods have been explored for tuning MPC cost functions, including meta-heuristics that can operate without explicit knowledge of the fitness landscape (e.g., [6]), lexicographic optimization [7], optimization via sequential semi-definite programming [8], linear approximation techniques [9], and Bayesian optimization (e.g., [10]). Despite these advancements, none of these approaches fully address the challenge of optimal vehicle guidance. Most either impose high computational costs, making them unsuitable for real-time applications, or require significant simplifications that reduce their effectiveness.

To address these limitations, we propose a dynamic Weights

*This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), MOCCA Project, ref. 17041397.

and Prediction Horizon-varying MPC (W-PH-MPC) approach. The core idea is to adjust both the cost function weights and the prediction horizon online to achieve optimal control performance. This is accomplished by integrating MPC with Deep Reinforcement Learning (DRL), where a deep neural network (DNN) trained using advanced RL algorithms dynamically determines the MPC parameters. In RL, an agent learns by interacting with the environment through actions, which lead to new states (observations) and rewards. The goal is to learn a policy that maximizes cumulative rewards over time. A well-designed reward function is critical to guiding the agent toward the desired behavior, as it influences the learning process and shapes decision-making during training. Since RL can optimize a multi-objective reward structure, it provides a powerful framework for automatic MPC tuning. Furthermore, the tuning process can be repeated whenever system parameters or objectives change, offering a flexible alternative to manual MPC parameter tuning.

Although the idea of learning MPC cost function weights with RL is not entirely new, [11] for instance, applied Q-learning with a Q-table, discrete action space, and a piecewise reward function to a quadcopter application—many studies focus on modifying the control structure online. In [12], for example, a multi-MPC approach switches between different controllers to handle model nonlinearities and adapt to changing operating conditions. Other works, such as [13]–[17] have explored online control structure adaptations but do not explicitly address the simultaneous adjustment of both the cost function weights and prediction horizon. [18] reformulated the prediction horizon as a discrete, positive integer time variable and proposed a nonlinear model predictive control (NMPC) approach for velocity regulation, incorporating a self-correcting mechanism for adjusting the prediction horizons. As demonstrated in [19], a tailored genetic algorithm can be used for real-time optimization of a path-tracking controller based on nonlinear model predictive control (NMPC), specifically designed for low-speed vehicle operation.

Additionally, [20]–[22] demonstrate how Gaussian Process Regression and neural networks can be used as predictive models that self-adjust using collected data, enhancing control accuracy and reducing computational load. Reinforcement learning, in particular, shows promise in MPC cost function tuning. In [23], an RL-based tuning strategy was presented, while [11] proposed a weights-varying MPC that adapts to different driving scenarios using DRL. Since safety is a critical concern in RL-based control, [24] introduced a safe learning framework that constrains DRL actions, enabling the algorithm to identify optimal parameters without compromising safety. Building on this, [25] demonstrated how DRL can adapt MPC weights in real time, ensuring both safety and efficiency throughout the learning process.

Our approach distinguishes itself by employing Policy Gradient RL algorithms, continuous action spaces, discrete action space, and deep neural networks for policy representation, combined with a continuous multi-objective reward function. Unlike prior work that typically focuses on either cost function weights tuning or control structure adaptation, our dynamic W-PH-MPC approach enables real-time adjustment of both the cost function weights and the prediction horizon, providing a more responsive and adaptive control system. To evaluate the effectiveness of our approach, we implement the DRL-driven W-PH-MPC in a vehicle-tracking scenario for autonomous driving. The optimization objectives focus on both tracking accuracy and ride comfort.

A. Comparison with Other Adaptive MPC Tuning Methods

In addition to demonstrating the performance of the proposed W-PH-MPC framework, it is essential to compare it with other adaptive MPC tuning approaches from the literature. Table I summarizes the key differences between DRL-based W-PH-MPC, Bayesian Optimization-based MPC, and Genetic Algorithm-based MPC.

TABLE I: Comparison of Adaptive MPC Tuning Methods

Aspect	**W-PH-MPC**	**Bayesian Opt.**	**Genetic Alg.**
Adaptability	Online	Offline	Offline
Sample Efficiency	Low	High	Low
Computational Demand	High	Moderate	moderate
Convergence Rate	Slow	Fast	moderate
Robustness	High	moderate	Moderate
Real-Time Suitability	Yes	No	No

This comparison highlights that, although Bayesian Optimization and Genetic Algorithm based MPC methods are valuable for offline parameter tuning due to their efficiency and global search capabilities, they generally lack the adaptability required for real time applications. The proposed W-PH-MPC framework, by contrast, provides continuous, online adaptation of the MPC parameters, making it particularly suitable for autonomous driving scenarios where environmental conditions and vehicle dynamics may change unpredictably. This real time adaptability enables the controller to maintain optimal performance without manual retuning, providing a highly responsive and flexible solution for dynamic driving environments. However, W-PH-MPC's online nature means it must solve an augmented quadratic program at each control step, resulting in increased computational cost per step compared to offline methods, in exchange for seamless real time parameter adaptation.

This paper is structured as follows. Section II develops the MPC-based ACC formulation, detailing vehicle dynamics, cost function and constraint handling. Section III defines the RL action spaces for continuous cost-weight tuning and discrete prediction-horizon selection. Section IV introduces signal scaling to normalize MPC cost terms, and Section V designs a multi-objective reward balancing tracking accuracy, ride comfort and energy efficiency. Section VI reviews the PPO and DDPG algorithms used for adaptive control. Section VII presents simulation studies in five parts—training-phase performance, unseen-profile validation, weights and prediction-horizon variation, comfort analysis and

computational metrics. Finally, Section VIII concludes and outlines future research directions.

II. MPC-Based ACC

Our ACC takes into account various factors about the road and traffic ahead, like the road's slope, speed limits, and the predicted speed of the car in front, to decide the best speed for the host car. Model Predictive Control (MPC) is the main method used in EACC because it turns the problem into an online optimization task.

A. MPC Formulation for ACC

The MPC-based ACC optimizes the cost function

$$\min_{F_{t,i},\, F_{b,i},\, \delta_{s,i},\, \delta_{F,i}} \sum_{j=k}^{k+H-1} \Big[f_{\text{app}}(F_{t,i}, v_{h,i})\, T_s + \omega_{\text{track}}\, \delta_{s,i}^2 + \omega_F\, \delta_{F,i}^2 + \omega_b\, F_{b,i}^2 \Big] \tag{1}$$

subject to

$$d_{i+1} = d_i + \frac{v_{p,i} + v_{p,i+1}}{2} T_s - \frac{v_{h,i} + v_{h,i+1}}{2} T_s, \tag{2}$$

$$v_{h,i+1} = v_{h,i} + \frac{T_s}{m_{\text{eq}}} \Big(F_{t,i} - F_{b,i} - \bar{F}_{\text{resist},i} \Big), \tag{3}$$

$$d_{i+1} \geq d_{\min} + h_{\text{safety}}\, v_{h,i+1}, \tag{4}$$

$$d_{i+1} \leq d_{\min} + h_{\text{track}}\, v_{h,i+1} + \delta_{s,i}, \tag{5}$$

$$0 \leq F_{t,i} \leq F_{t,\max}, \tag{6}$$

$$F_{t,i} \leq a_{20} + a_{21}\, v_{h,i}, \tag{7}$$

$$\left| F_{t,i} - F_{t,i-1} \right| \leq \Delta F_{t,\max} + \delta_{F,i}, \tag{8}$$

$$0 \leq F_{b,i} \leq F_{\text{brake,max}}, \tag{9}$$

$$0 \leq v_{h,i} \leq v_{\max,i}. \tag{10}$$

Here, T_s is the sampling time in (1). The energy consumption function $f_{\text{app}}(F_t, v)$ is a polynomial approximation of the vehicle's power consumption, given by

$$f_{\text{app}}(F_t, v) = p_{00} + p_{01} F_t + p_{10} v + p_{11} F_t v + p_{02} F_t^2 + p_{20} v^2, \tag{11}$$

where the coefficients p_{ij} are determined offline through system identification. In this function, F_t represents the traction force applied by the host vehicle and v is its velocity. If the cost function were comprised solely of the energy consumption model, $f_{\text{app}}(F_t, v)$, as defined in (1), the host vehicle would lack any motivation to move since $f_{\text{app}}(F_t, v)$ decreases with a smaller traction force F_t. To counter this, a second term, $\delta_{s,i}^2$, weighted by ω_{track}, is introduced. The slack variable $\delta_{s,i}$ appearing in (5) and illustrated in Fig. 1 increases as a penalty when the inter-vehicle distance d falls below the desired tracking distance d_{tracking}, thereby encouraging the host vehicle to accelerate. In (5), both $d_{\min}$ and h_{track} are constants.

The third component in (8) involves the slack variable $\delta_{F,i}$, which penalizes abrupt changes in the traction force F_t between consecutive control steps, as detailed in (8). Thus, $\delta_{F,i}$ grows larger when the variation in F_t exceeds the maximum allowable tolerance $\Delta F_{t,\max}$, thereby promoting comfort by discouraging strong accelerations.

The final term in (9) is included to limit the use of mechanical braking force F_b as the brakes dissipate kinetic energy. In (3), the resistive force $\bar{F}_{\text{resist}}$ is linearized to handle the nonlinear behavior of air resistance F_{air}. The inequality in (4) ensures a minimum safety inter-vehicle distance, where the constant $h_{\min}$ represents the minimum time headway. Additionally, (10) restricts the maximum vehicle speed according to the road's speed limit $v_{\max}$, and the feasible region for F_t is defined by the piecewise linear constraints in (7).

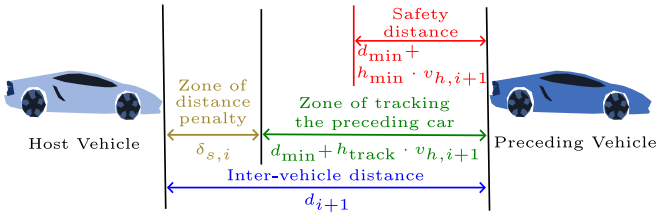

Fig. 1: Design of the inter-car distance control for ACC

III. Action Space Definition for the Two Agents

The action space includes two key elements: the prediction horizon and the MPC weight values, which are dynamically adjusted through reinforcement learning. The MPC employs a penalty-based method with slack variables to guarantee feasibility and ease constraints when it is not possible to meet them fully. This approach enables the RL agents to effectively handle the constraints through the MPC's penalty method, all while optimizing control performance.

The prediction horizon is chosen from a fixed set of values, allowing the agent to change how far ahead the controller plans. This helps the system respond to different situations while keeping calculations efficient.

The MPC weights are continuous values that balance objectives such as energy efficiency, tracking precision, jerk minimization, and constraint enforcement. By constraining the neural network's outputs to lie within predefined bounds, we prevent extreme weight values and ensure consistently high control performance.

A. Continuous Action Space (Weight Optimization by Agent 1)

The first RL agent dynamically adjusts the cost function weights

$$a_{\text{RL1}} = \pi_1(s) = [\omega_{\text{track}}, \omega_b] \tag{12}$$

where s represents the vehicle's state, which includes speed, inter-vehicle distance between the host and the preceding vehicle, acceleration, and road conditions, and $\pi_1(s)$ is the policy that maps the observed state to the optimal weights.

The weights are constrained:

$$\omega_{\min} \leq \omega_{\text{track}}, \omega_b \leq \omega_{\max} \tag{13}$$

By continuously adapting these parameters, the system can balance energy efficiency, tracking accuracy, and passenger comfort.

B. Discrete Action Space (Prediction Horizon Optimization by Agent 2)

The discrete prediction-horizon selection by the second RL agent is defined as follows. First, the number of available horizons is

$$N_H = H_{\max} - H_{\min} + 1, \tag{14}$$

where $H_{\min}$ and $H_{\max}$ are the minimum and maximum allowable horizons. Next, the agent samples an index k from its categorical policy:

$$k \sim \pi_2(k \mid s), \quad k \in \{0, 1, \ldots, N_H - 1\}. \tag{15}$$

Finally, the selected prediction horizon is

$$a_{\mathrm{RL2}} = H_{\min} + k, \tag{16}$$

which maps the sampled index into the actual horizon value. $H_{\min}$ ensures real-time feasibility and $H_{\max}$ allows for long-term predictive planning in complex traffic conditions.

IV. Enhancing RL Learning with Signal Scaling in the MPC Cost Function

To enhance training efficiency, signal scaling was applied to normalize the terms in the MPC cost function, ensuring comparable magnitudes and reducing the learning search space. This approach helps the RL agent converge faster and focus on learning the relative importance of cost terms, rather than being biased by differences in their raw scales. While the MPC controller maintains stability, the learned weights influence performance aspects such as responsiveness, comfort, and tracking quality.

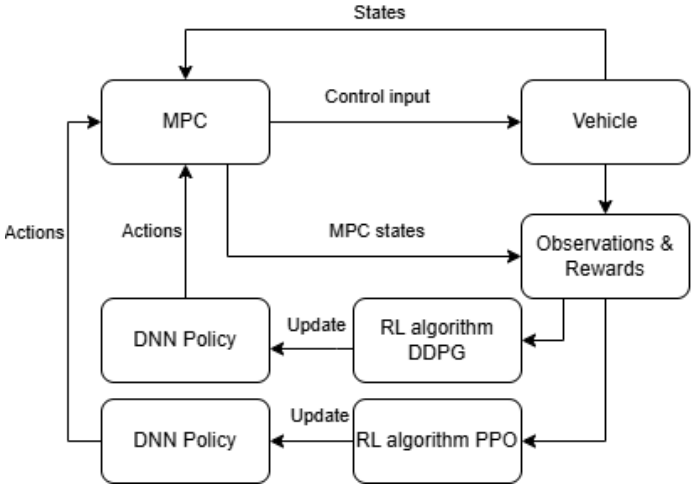

Fig. 2: MPC–RL architecture..

The control loop uses a standard MPC scheme to receive states and generate control actions as seen in Fig. 2. Both agents observe the system states at each step and receive a scalar reward that balances tracking error, ride comfort, and energy efficiency. A PPO agent adapts the prediction horizon, while a DDPG agent tunes the MPC cost-function weights. Each episode lasts T_f seconds ($N_{\mathrm{ep}} = T_f/T_s$ steps, with T_s the sampling interval). During training, the PPO agent updates its policy after collecting N_e steps of experience, while the DDPG agent updates continuously by sampling minibatches of size M from its replay buffer and applying soft target updates. After training, only the learned policies are used to adjust MPC parameters online.

V. Reward Function Design

We use a Gaussian-like distribution function, where rewards are highest when the agent achieves low power consumption, lower jerk, and accurate trajectory tracking. The exponential formulation ensures that as any of these terms increase, the reward will decrease, allowing the agent to understand the action that resulted in a low reward is not desired. We adopt this approach to prevent unintended behavior that exploits the reward function. We design the system so that the agent is naturally drawn toward an optimal operating point where both power consumption and jerk are kept low while still maintaining good traceability. The reward function then ultimately results as

$$R = A \cdot \exp\left(-\left(\frac{c_1^2}{2\sigma_{\mathrm{power}}^2} + \frac{c_2^2}{2\sigma_{\mathrm{jerk}}^2} + \frac{c_3^2}{2\sigma_{\mathrm{distance}}^2}\right)\right) \tag{17}$$

where A scales the overall reward magnitude, and c_1, c_2, and c_3 represent power consumption, longitudinal jerk, and following-distance error, respectively. The parameters σ_{power}, σ_{jerk}, and $\sigma_{\mathrm{distance}}$ control the rate at which the reward decays as each cost term increases.In our implementation, we set $\sigma_{\mathrm{power}} = 2.75$, $\sigma_{\mathrm{jerk}} = 0.73$, and $\sigma_{\mathrm{distance}} = 9.25$ by running a baseline controller on typical driving cycles and computing the sample standard deviations of power, jerk, and distance-error signals. These values reflect real signal variability, helping the reward function provide smooth gradients for learning while penalizing large deviations in a data-driven way.

VI. RL Agents Algorithm

We choose the PPO agent for prediction horizon selection, using on-policy learning from recent interactions to keep changes smooth and predictable. For continuous weight tuning, we use DDPG, which leverages a replay buffer for faster fine-grained adjustments and employs soft-target updates with scheduled exploration noise.

A. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an on-policy reinforcement learning algorithm designed to train stochastic policies in a stable and robust manner. The core principle is to maximize a surrogate objective function while constraining each policy update to prevent overly large, destabilizing changes. In practice, the agent initializes its policy (actor) and value function (critic) networks and interacts with the environment to collect trajectories of states, actions, rewards, and next states. Based on this experience, PPO estimates the cumulative discounted returns and computes advantage values, which quantify how much better a specific action performs compared to the expected value under the current policy. For each sample, the algorithm

then computes the probability ratio between the updated policy and the policy that generated the data. To ensure safe and gradual learning, this ratio is clipped within a predefined range, keeping the new policy close to the old one and preventing sudden shifts in behavior. The actor network is updated by maximizing this clipped objective via gradient ascent, while the critic network is trained to minimize the error between its value predictions and the observed returns. This cycle of data collection, advantage estimation, and clipped policy and value function updates contributes to the learning process and is repeated until the policy achieves satisfactory performance.

Algorithm 1 Proximal Policy Optimization (PPO)

Initialize: Policy parameters θ, value function parameters ϕ
Repeat for each episode:

- Collect trajectories $\{s_t, a_t, r_t, s_{t+1}\}$ by running policy π_θ in the environment
- Compute advantage estimates $\hat{A}_t$ using Generalized Advantage Estimation (GAE)
- Compute rewards-to-go R_t
- **For each policy update step:**
  - Compute policy ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

  - Compute surrogate objective:

$$L^{\text{PPO}}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

  - **Update policy** using gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L^{\text{PPO}}(\theta)$$

- **For each value function update step:**
  - Compute value function loss:

$$L_V(\phi) = \mathbb{E}\left[\left(V_\phi(s_t) - R_t\right)^2\right]$$

  - **Update value function** using gradient descent:

$$\phi \leftarrow \phi - \alpha \nabla_\phi L_V(\phi)$$

The objective function, incorporating this constraint, is given by

$$L_{PPO}(\theta) = \mathbb{E}\left[\min\{r_t(\theta)\cdot\hat{A}_t, \text{clip}(\hat{r}_t(\theta), 1-\epsilon, 1+\epsilon)\cdot\hat{A}_t\right] \tag{18}$$

$r_t(\theta)$ denotes the ratio of the new policy to the old policy, i.e.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{19}$$

This function represents the ratio of the probability of taking action a_t in state s_t under the current policy π_θ to the probability of taking the same action under the old policy $\pi_{\theta_{\text{old}}}$. Specifically, $\pi_\theta(a_t|s_t)$ is the probability of selecting action a_t given state s_t under the new policy parameterized by θ, while $\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability of selecting action a_t given state s_t under the old policy parameterized by θ_{old}. The function $r_t(\theta)$ is essential for ensuring that the updates to the policy are not too large, which helps maintain the stability of training in PPO.

The advantage function, denoted by $\hat{A}_t$, measures how much better taking a particular action at time t is compared to the average value of the state. In the PPO agent, it is computed using Generalized Advantage Estimation (GAE) from

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \tag{20}$$

with the temporal-difference error δ_t defined as

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{21}$$

In these equations, r_t represents the reward received at time t, $V(s_t)$ is the value function for state s_t, γ is the discount factor controlling the importance of future rewards, and λ is a parameter that balances bias and variance.

B. Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an off-policy, model-free reinforcement learning algorithm designed for continuous action spaces. It employs an actor-critic architecture, where the actor network maps states to continuous actions, and the critic network estimates the Q-value of state-action pairs to represent the expected cumulative reward. To ensure sufficient exploration, the agent selects actions by adding noise to the actor's output. After executing an action, the resulting state, action, reward, and next state are stored in a replay buffer. During training, the agent samples mini-batches from this buffer to update its networks, which helps decorrelate experiences and stabilize learning. The critic network is trained to minimize the error between its Q-value predictions and target Q-values computed using the received reward and the target networks. Specifically, the target Q-value is calculated as the immediate reward plus the discounted estimate of the next state's value, given by the target critic and target actor. The actor network is updated through the deterministic policy gradient to maximize the expected Q-value predicted by the critic. To further improve stability, DDPG maintains separate target actor and target critic networks that slowly track the learned networks via soft updates. Combining deterministic policy gradients, target networks, exploration noise, and experience replay enables DDPG to achieve stable and sample-efficient learning for complex continuous control tasks.
The critic's objective is to minimize the error between its predicted Q-values and the target Q-values given by the Bellman equation. The loss function is:

$$L_Q(\phi) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - Q_\phi(s_i, a_i)\right)^2, \tag{22}$$

where the target value y_i is defined as:

$$y_i = r_i + \gamma\, Q_{\phi'}\left(s_{i+1},\ \mu_{\theta'}(s_{i+1})\right). \tag{23}$$

Here, ϕ and θ are the parameters of the critic and actor networks, respectively, while ϕ' and θ' denote the corresponding target network parameters. The target Q-value y_i combines the

Algorithm 2 Deep Deterministic Policy Gradient (DDPG)

Initialize:

- Actor network μ_θ and critic network Q_ϕ with parameters θ, ϕ
- Target networks $\mu_{\theta'}$ and $Q_{\phi'}$ with parameters $\theta' \leftarrow \theta$, $\phi' \leftarrow \phi$
- Replay buffer $\mathcal{D}$
- Exploration noise process $\mathcal{N}$

Repeat for each episode:

- **For each time step:**
  - Select action $a_t = \mu_\theta(s_t) + \mathcal{N}_t$
  - Execute action a_t, observe reward r_t and next state s_{t+1}
  - Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
- **For each gradient update step:**
  - Sample minibatch of transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$
  - Compute target Q-value:

$$y_i = r_i + \gamma Q_{\phi'}(s_{i+1}, \mu_{\theta'}(s_{i+1}))$$

  - Compute critic loss:

$$L_Q(\phi) = \mathbb{E}\left[\left(y_i - Q_\phi(s_i, a_i)\right)^2\right]$$

  - Minimize $L_Q(\phi)$ using gradient descent:

$$\phi \leftarrow \phi - \alpha_Q \nabla_\phi L_Q(\phi)$$

  - Compute actor policy gradient:

$$\nabla_\theta J = \mathbb{E}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q_\phi(s, a)\big|_{a=\mu_\theta(s)}\right]$$

  - Update actor network using gradient ascent:

$$\theta \leftarrow \theta + \alpha_\mu \nabla_\theta J$$

  - Soft update target networks:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

immediate reward r_i and the discounted estimate of the next state's value, using the target critic $Q_{\phi'}$ and target actor $\mu_{\theta'}$.

VII. Results and Discussion

A. Training Settings

In our experiments, the simulation is implemented in Simulink using the `MPC` model, with two agent blocks interacting with the environment. The observation space is split into two parts: a 5-dimensional vector containing d_h, v_h, a_h, the current prediction horizon N, and jerk for the PPO agent, and a 7-dimensional vector with similar features but excluding N and including the MPC weight parameters controlled by the DDPG agent. The action space consists of two components: a discrete set selecting the prediction horizon from $N_{\min}$ to $N_{\max}$, and a continuous space adjusting the MPC weight parameters within specified bounds. With a sample time $T_s = 0.1$ seconds and total simulation time $T_{ep} = 386$ seconds, each episode contains MaxStepsPerEpisode = $\frac{T_{ep}}{T_s} = 3860$ steps. The two agents are trained concurrently under a decentralized learning strategy over 300 episodes.The PPO agent's MLP actor and critic each have two 128-unit ReLU layers; the actor outputs discrete actions over 5 horizons (5–9) without activation, and the critic outputs a scalar. The DDPG actor has two 128-unit ReLU layers followed by a tanh scaled to MPC bounds. Its critic processes state and action separately through 128-unit ReLU layers, merges them, and outputs a scalar Q-value. Learning rates are $3{\times}10^{-4}/3{\times}10^{-3}$ (PPO actor/critic) and $1{\times}10^{-4}/1{\times}10^{-3}$ (DDPG). DDPG uses Gaussian noise ($\sigma = 0.2$) for exploration. A fixed seed ensures reproducibility.

B. Tracking-Optimized Mode

Figure 3 shows the planned velocity profile V_p from a segment of a Real Driving Emissions (RDE) cycle used during training.

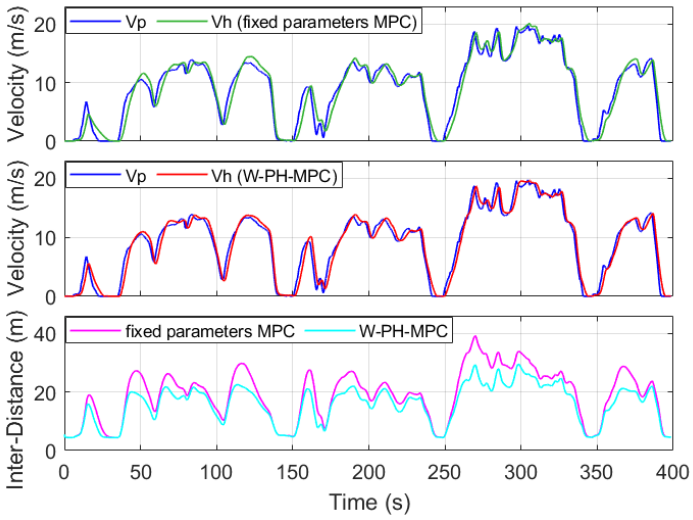

Fig. 3: Performance evaluation of the RL–MPC approach on a velocity profile included in the training phase.

The RDE cycle is a standardized, publicly administered driving test designed to capture a broad spectrum of real-world conditions. In our experiments, we use a 6.5-minute velocity profile recorded from a real vehicle on public roads, which is partitioned into three phases—Phase A (0.0–2.0 min, urban stop-and-go, 0–50 km/h), Phase B (2.0–4.2 min, suburban mixed-flow, up to 50 km/h with occasional stops), and Phase C (4.2–6.5 min, high-speed segments up to 72 km/h with intermittent slowdowns and stop-and-go behavior)—to ensure that the training data span everything from dense congestion to sustained higher-speed driving. During training, the host vehicle's MPC-based ACC system learns to adjust its cost-function weights and prediction horizon online to track this profile smoothly. To evaluate generalization, the trained DRL agents are then applied to a different segment of an RDE drive—one featuring sudden accelerations and decelerations not seen during training and their closed-loop performance (in terms of tracking error, jerk, and energy consumption) is compared against a fixed-parameter MPC baseline.

Figure 3 also compares the tracking performance of the trained RL agents (W-PH-MPC) with that of a baseline using fixed parameters MPC. The first two subplots demonstrate that, under the MPC–RL policy, the host vehicle v_h follows the preceding vehicle v_p more closely than the fixed-parameters MPC, demonstrating the RL-enhanced controller's performance. The

third subplot shows the inter-distance d_h from the host vehicle to the preceding vehicle over time. The cyan curve representing the performance of the RL agents stays below the magenta baseline, indicating that the agents can maintain a shorter following distance, resulting in better tracking performance by **17.07%**. Note that the host vehicle is designed with a hard constraint to maintain a minimum safety inter-distance of 4.5 meters from the preceding vehicle at all times to avoid critical driving situations.

C. Validation on a Different Velocity Profile

To further evaluate the generalization capability of the proposed RL–MPC framework, we assess its performance on a 14.5-minute segment of an RDE cycle that includes both rural and urban driving conditions not present in the training dataset, as shown in Fig. 4. The first two subplots show that under the MPC-RL policy (W-PH-MPC), the host vehicle v_h tracks the preceding vehicle v_p more closely than the fixed-parameters MPC baseline, with a **16.4%** improvement in longitudinal tracking performance. Finally, the bottom subplot of Fig. 4 shows that the RL–MPC policy maintains tighter inter-vehicle distances for tracking purposes, reaffirming its robust performance across diverse driving scenarios.

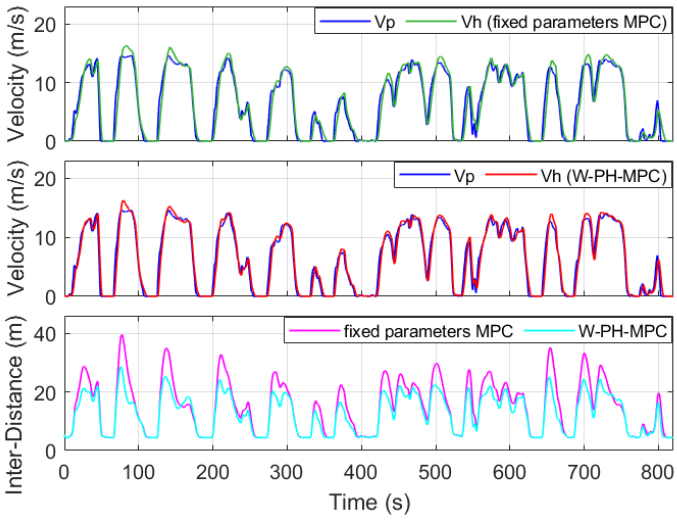

Fig. 4: Performance validation across different velocity profile that was not used during training

D. Weights and Prediction Horizon Variation

In this subsection, we show how reinforcement learning agents dynamically modulate MPC weights and the prediction horizon, directly affecting the slack variables associated with inter-vehicle distance constraints.

The slack variable of the distance, weighted by ω_{track}, changes over time, indicating how the agent relaxes or tightens the distance constraint, as shown in Figure 5. Originally ranging from 500 to 1000, it was scaled to [0.5, 1] to reduce exploration and accelerate the training process. The Brake Change Penalty, weighted by ω_{b}, regulates braking intensity to ensure smooth deceleration. changes in the Brake Change Penalty As seen in

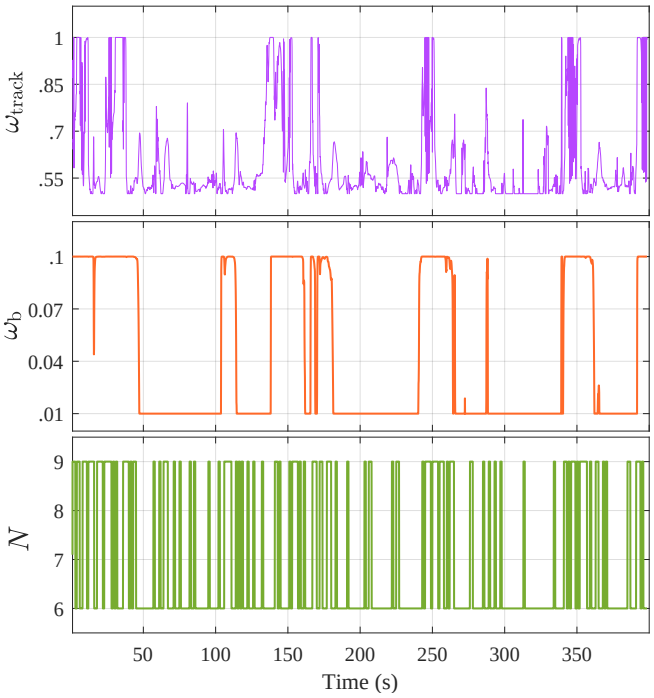

Fig. 5: Weights and prediction horizon variations.

Figure 5, indicate how the system adaptively adjusts its braking behavior over time.

Model Predictive Control (MPC) uses a finite prediction horizon to optimize future control actions. A longer horizon improves foresight but increases computational cost, while a shorter one is faster but potentially suboptimal. The prediction horizon varies between H_{min} and H_{max}, as seen in Figure 5, indicating transitions between short- and long-term planning. These variations suggest an adaptive strategy that adjusts the planning scope based on system requirements or changing conditions.

E. Comfortability and Time Profile Consumption

Table II summarizes the longitudinal jerk statistics across different driving scenarios to assess ride comfort.

TABLE II: Longitudinal Jerk Statistics for Comfort Analysis

Case	Max Abs [m/s^3]	Median [m/s^3]	75$^{\text{th}}$ Perc. [m/s^3]
VP1-Fixed	3.3502	0.0086	0.1731
VP1-Varying	3.2633	0.0004	0.1798
VP2-Fixed	2.9294	0.0060	0.0908
VP2-Varying	2.8644	0.0005	0.0960

A key observation is that using the varying weight–prediction horizon (W–PH) configuration reduces the maximum peak jerk values across both velocity profiles, indicating fewer extreme accelerations and a smoother overall ride. The median jerk, which represents the middle value in the dataset, remains extremely low—almost zero—suggesting that during most of the drive, the vehicle experiences little to no sudden changes in acceleration. The 75$^{\text{th}}$ percentile jerk represents the value below which 75% of the jerk measurements fall, highlighting how the system performs during more active driving segments

without being influenced by rare, extreme events. Although this value shows a slight increase under the varying configuration, the trade-off appears beneficial, as it enables better tracking performance and still results in reduced peak jerk. Furthermore, the maximum peak jerk is reduced by approximately **2.52%** for VP1 and **2.22%** for VP2 under the varying W–PH setup. These results demonstrate that the varying W–PH setup effectively improves both tracking accuracy and ride comfort.

Time (s)	Consumption-Profile 1	Consumption-Profile 2
Fixed parameters MPC	11.0184	10.6575
varying W-PH-MPC	11.0241	10.6592

TABLE III: Time profile consumption for velocity profiles 1 and 2

The tracking performance was considerably improved by a modest rise in time profile consumption, with no adverse impact on energy efficiency as shown in Table III.

For real-time implementation, the control and policy modules designed in MATLAB/Simulink can be converted to ANSI/ISO C code using MATLAB Coder or Simulink Coder. These tools generate efficient, portable C code from supported functions or models, which can be compiled for embedded targets like ARM Cortex-M (e.g., STM32), dSPACE, Speedgoat, or NVIDIA Jetson—ensuring compatibility with real-time automotive and control systems.

VIII. Conclusion

To address the difficulties of manually tuning the MPC cost function and selecting an appropriate prediction horizon, we have introduced a Weights and Prediction Horizon Varying MPC (W-PH-MPC), which autonomously learns a policy to adjust both the cost function weights and the prediction horizon in real time. With the help of advanced Deep Reinforcement Learning (RL) algorithms, our method tunes these parameters to achieve better control performance and adaptability. As the RL-driven W-PH-MPC framework provides a flexible and automated tuning approach, it can be applied to different control systems, eliminating the reliance on manual adjustments.

References

[1] Y. Jia, A. Abdelkarim, X. Klingbeil, R. Savelsberg, and D. Görges, "Performance evaluation of energy-optimal adaptive cruise control in simulation and on a test track," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 4994–5000, 2023.

[2] A. Abdelkarim and P. Zhang, "Optimal scheduling of preventive maintenance for safety instrumented systems based on mixed-integer programming," in *Model-Based Safety and Assessment: 7th International Symposium, IMBSA 2020, Lisbon, Portugal, September 14–16, 2020, Proceedings 7*. Springer, 2020, pp. 83–96.

[3] A. Abdelkarim, Y. Jia, and D. Gorges, "Optimization of vehicle-to-grid profiles for peak shaving in microgrids considering battery health," in *IECON 2023-49th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2023, pp. 1–6.

[4] A. Abdelkarim, Y. Jia, and D. Görges, "An accelerated interior-point method for convex optimization leveraging backtracking mitigation," in *IECON 2023-49th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2023, pp. 1–6.

[5] A. Abdelkarim, "Development of numerical solvers for online optimization with application to mpc-based energy-optimal adaptive cruise control," master thesis, Technische Universität Kaiserslautern, 2020. Available online at http://dx.doi.org/10.13140/RG.2.2.11897.28000, accessed: 2025-02-28.

[6] V. Ramasamy, R. K. Sidharthan, R. Kannan, and G. Muralidharan, "Optimal tuning of model predictive controller weights using genetic algorithm with interactive decision tree for industrial cement kiln process," *Processes*, vol. 7, no. 12, p. 938, 2019.

[7] A. S. Yamashita, A. C. Zanin, and D. Odloak, "Tuning of model predictive control with multi-objective optimization," *Brazilian Journal of Chemical Engineering*, vol. 33, no. 2, pp. 333–346, 2016.

[8] G. Shah and S. Engell, "Tuning mpc for desired closed-loop performance for mimo systems," in *Proceedings of the 2011 American Control Conference*. IEEE, 2011, pp. 4404–4409.

[9] A. Al-Ghazzawi, E. Ali, A. Nouh, and E. Zafiriou, "On-line tuning strategy for model predictive controllers," *Journal of Process Control*, vol. 11, no. 3, pp. 265–284, 2001.

[10] D. Stenger, M. Ay, and D. Abel, "Robust parametrization of a model predictive controller for a cnc machining center using bayesian optimization," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 10 388–10 394, 2020.

[11] M. Mehndiratta, E. Camci, and E. Kayacan, "Automated tuning of nonlinear model predictive controller by reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3016–3021.

[12] M. Soliman, O. Malik, and D. T. Westwick, "Multiple model predictive control for wind turbines with doubly fed induction generators," *IEEE Transactions on Sustainable Energy*, vol. 2, no. 3, pp. 215–225, 2011.

[13] P. Falcone, M. Tufo, F. Borrelli, J. Asgari, and H. E. Tseng, "A linear time varying model predictive control approach to the integrated vehicle dynamics control problem in autonomous systems," in *2007 46th IEEE conference on decision and control*. IEEE, 2007, pp. 2980–2985.

[14] K. Alexis, G. Nikolakopoulos, and A. Tzes, "Switching model predictive attitude control for a quadrotor helicopter subject to atmospheric disturbances," *Control Engineering Practice*, vol. 19, no. 10, pp. 1195–1207, 2011.

[15] I. Masar and E. Stöhr, "Gain-scheduled lqr-control for an autonomous airship," in *18th International Conference on Process Control*, 2011, pp. 14–17.

[16] M. X. Huang and I. Kolmanovsky, "Switch gain scheduled explicit model predictive control of diesel engines," Sep. 19 2017, uS Patent 9,765,621.

[17] R. G. Patel and J. J. Trivedi, "Sagd real-time production optimization using adaptive and gain-scheduled model-predictive-control: A field case study," in *SPE Western Regional Meeting*. SPE, 2017, p. D051S016R006.

[18] Y. Wei, Y. Wei, Y. Gao, H. Qi, X. Guo, M. Li, and D. Zhang, "A variable prediction horizon self-tuning method for nonlinear model predictive speed control on pmsm rotor position system," *IEEE Access*, vol. 9, pp. 78 812–78 822, 2021.

[19] X. Du, K. K. K. Htet, and K. K. Tan, "Development of a genetic-algorithm-based nonlinear model predictive control scheme on velocity and steering of autonomous vehicles," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 6970–6977, 2016.

[20] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 269–296, 2020.

[21] G. Wang, Q.-S. Jia, J. Qiao, J. Bi, and M. Zhou, "Deep learning-based model predictive control for continuous stirred-tank reactor system," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 3643–3652, 2021.

[22] M. Rokonuzzaman, N. Mohajer, S. Nahavandi, and S. Mohamed, "Model predictive control with learned vehicle dynamics for autonomous vehicle path tracking," *IEEE Access*, vol. 9, pp. 128 233–128 249, 2021.

[23] H. Bao, Q. Kang, X. Shi, M. Zhou, H. Li, J. An, and K. Sedraoui, "Moment-based model predictive control of autonomous systems," *IEEE Transactions on Intelligent Vehicles*, vol. 8, pp. 2939–2953, 2023.

[24] B. Zarrouki, V. Klos, N. Heppner, S. Schwan, R. Ritschel, and R. Voswinkel, "Weights-varying mpc for autonomous vehicle guidance: A deep reinforcement learning approach," in *Proceedings of the 2021 European Control Conference (ECC)*. Delft, The Netherlands: IEEE, 2021.

[25] B. Zarrouki, M. Spanakakis, and J. Betz, "A safe reinforcement learning driven weights-varying model predictive control for autonomous vehicle motion control," arXiv preprint arXiv:2402.02624, 2024, available: https://arxiv.org/abs/2402.02624.