Comparison of DDPG and MPC on Adaptive Curise Control

Fuat Işıklan
Control Engineering, Yildiz Techincal University, Ankara, Turkey
Email: fuat.isiklan@std.yildiz.edu.tr

***Abstract*—**
In this work we try to understand and test the differences between ddpg (deep policy gradient) and MPC on adaptive cruise control. For the sake of simplicity, we have used the COM model to fast-implement and test the results. Both algorithms were trained and solved with the same parameters and extended analyzes were performed. We have specifically compared the algorithms with IPO solution, which we have set as benchmark. We have seen that when there are no modeling errors and considering the testing done in training bounds on ddpg the results are significantly similar; moreover, the results DRL solution CPU time is much less than MPC.

***Index Terms*—Adaptive Cruise Control, Model Predictive Control, Deep Reinforcement Learning, DDPG, Optimal Control**

I. Introduction

ACC is a fundamental building block in modern ADAS. The main aim is to regulate the longitudinal behavior of a vehicle by adjusting its break / twist to maintain a safe distance from a lead vehicle. MPC is a well known solution for such tasks due to its ability to incorporate dynamic constraints and with optimization capability on feature horizons. Meanwhile, DDPG which is a sub approach for Deep Reinforcement Learning, offers an alternative solution/control directly environment data without requiring system model.

In this work, we compare the performance of a standard MPC controller against a Deep Deterministic Policy Gradient (DDPG) agent on a low-dimensional car-following task. Both approaches operate on the same three-dimensional state-space model with identical stage costs and control constraints. The DDPG agent is trained offline using a reward function aligned with the MPC cost, while the MPC controller solves a constrained optimization problem online at each step. As a common reference, we compute an open-loop benchmark solution using a full-horizon single-shooting optimization (IPO).

Our aim is not measuring robustness, rather we wanted to see how achievable control performance of DRL when applied to the same control problem under idealized conditions. The simplicity of the dynamic model hence chosen and it favors to just comparison between two approaches in terms of tracking error, control smoothness and episode cost.

II. Theory and Methodology

A. Adaptive Cruise Control Problem

We consider a single-lane car-following scenario, where the ego vehicle (denoted by index i) tracks a lead vehicle $(i-1)$ using first-order longitudinal dynamics, as commonly employed in the literature [1]. Let $\ell_{i-1}, \ell_i \in \mathbb{R}$ denote the positions of the lead and ego vehicles, and $v_{i-1}, v_i \in \mathbb{R}$ their respective velocities. The ego vehicle's acceleration is denoted by $a_i \in \mathbb{R}$. The system state is defined using the gap error e and relative velocity e_v, computed as

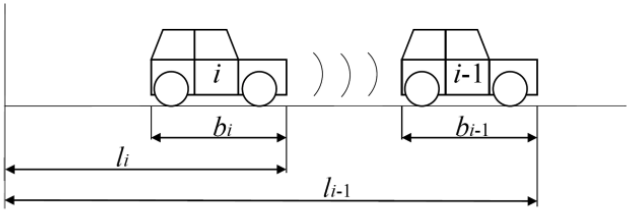


Fig. 1: Spacing and positioning between two vehicles, showing positions l_i, l_{i-1} and vehicle lengths b_i, b_{i-1}.

$$e = \ell_{i-1} - \ell_i - b_{i-1} - (d_0 + hv_i)\,, \qquad e_v = v_{i-1} - v_i, \tag{1}$$

where b_{i-1} is the length of the lead vehicle, $d_0 = 5\,\mathrm{m}$ is a standstill spacing, and $h = 1\,\mathrm{s}$ is the desired time gap. Assuming the lead vehicle moves at constant speed (i.e., $a_{i-1} = 0$), the continuous-time dynamics [2] of the system are described by the state vector $x = [e,\ e_v,\ a_i]^\top$ and evolve according to

$$\begin{aligned} \dot{e} &= e_v - ha_i, \\ \dot{e}_v &= -a_i, \\ \dot{a}_i &= \frac{u - a_i}{\tau}, \end{aligned} \tag{2}$$

where $u \in [u_{\min}, u_{\max}]$ is the commanded acceleration and $\tau = 0.5\,\mathrm{s}$ models the actuation lag.

To balance tracking performance, comfort, and control effort, we define the following multi-objective stage cost:

$$\ell(x,u) = \alpha\sqrt{\left(\frac{e}{e_{\max}}\right)^2 + \epsilon} + \beta\sqrt{\left(\frac{u}{u_{\text{scale}}}\right)^2 + \epsilon} + \gamma\sqrt{\left(\frac{\dot{a}_i}{j_{\max}}\right)^2 + \epsilon}, \tag{3}$$

which penalizes the gap error e, control effort u, and jerk $\dot{a}_i$. The terms are scaled by

- $e_{\max} = 15\,\mathrm{m}$: maximum expected gap error,

- $u_{\text{scale}} = \max(|u_{\min}|, |u_{\max}|) = 3\,\text{m/s}^2$: control bound,
- $j_{\max} = \frac{u_{\max} - u_{\min}}{\Delta t} = 50\,\text{m/s}^3$: jerk bound with $\Delta t = 0.1\,\text{s}$.

The weighting coefficients $\alpha = \beta = \gamma = \frac{1}{3}$ assign equal importance to all three components, and a small constant $\epsilon = 10^{-8}$ ensures differentiability of the cost function.

The continuous dynamics in equation (2) are discretized using a fourth-order Runge–Kutta (RK4) integration scheme with a fixed time step of $\Delta t = 0.1\,\text{s}$, suitable for both MPC and reinforcement learning implementations. [3]

B. Model Predictive Control and IPO Benchmark

In this section we present optimal control strategy for the ACC problem usning Model Predictive Control, along with the infinite-horizon benchmark known as IPO. Both problems formulated as non-linear programming and solved using CasADİ [4]and IPOPOT with python.

Problem Formulation: We consider a discrete-time dynamical system with state $x_k \in \mathbb{R}^3$, control input $u_k \in \mathbb{R}$, and the discrete dynamics defined via an RK4 integrator:

$$x_{k+1} = f_d(x_k, u_k),$$

where the continuous-time model is discretized with time step $\Delta t = 0.1\text{s}$. The optimization objective is to minimize the cumulative cost over a finite or infinite horizon in equation (3):

where e_k is the gap error, $j_k = (a_k - a_{k-1})/\Delta t$ is the jerk, and the constants α, β, and γ weight the contributions of tracking accuracy, control effort, and comfort (smoothness) respectively.

The optimization is subject to:

$$\begin{aligned} x_0 &= x_{\text{init}}, \\ x_{k+1} &= f_d(x_k, u_k), \quad \forall k = 0, \ldots, N-1, \\ u_{\min} &\le u_k \le u_{\max}. \end{aligned}$$

Open-loop benchmark (IPO): The IPO strategy solves this optimal control problem over the entire simulation horizon, $T = 20\,\text{s}$ ($N = 200$ steps), in a single pass. This corresponds to a classical open-loop single-shooting problem:

$$\min_{X_{0:N},\, U_{0:N-1}} \sum_{k=0}^{N-1} \ell(x_k, u_k),$$

with all dynamics and constraints enforced through equality and box constraints. The resulting control sequence $U^\star_{\text{IPO}}$ serves as a performance benchmark, assuming perfect future knowledge. This solution is not causal and cannot be used in real time but provides an ideal trajectory for comparison.

Receding-horizon MPC: In contrast, MPC solves the same optimal control problem online at each time step, but only over a finite prediction horizon of length $N \in \{5, 28, 50\}$ corresponding to 0.5, 2.8, and 5.0 seconds respectively. Only the first input $u_0^\star$ from the optimized sequence is applied, after which the system state is updated and the horizon is shifted forward (receding horizon). Formally:

$$\min_{X_{0:N},\, U_{0:N-1}} \sum_{k=0}^{N-1} \ell(x_k, u_k), \quad \text{at each time step.}$$

The optimization is warm-started with the previous solution shifted forward, improving convergence. This receding-horizon policy allows real-time feedback control and robustness to modeling errors and disturbances.

Numerical Solution and Solver Settings: Both IPO and MPC problems are implemented using CasADi with exact Jacobians and solved using IPOPT. The solver is configured with:

- RK4 discretization of dynamics,
- limited-memory BFGS approximation of the Hessian,
- optimizer tolerance $\epsilon_{\text{opt}} = 10^{-6}$,
- input bounds $u_k \in [u_{\min}, u_{\max}]$,
- no explicit terminal constraints or costs (though easily extendable).

For all tested horizons $N \le 50$, the online MPC problem solves in under $10\,\text{ms}$ on a laptop, making it suitable for real-time embedded applications.

MPC provides a feedback-based, real-time control solution by solving a constrained finite-horizon optimization problem at each step. The IPO solution, by contrast, represents the ideal open-loop trajectory and serves as a benchmark. Their performance differences, including tracking error, control effort, and jerk, are visualized and discussed in Section III.

C. Deep Deterministic Policy Gradient

We employ Deep Deterministic Policy Gradient (DDPG) [5], a model-free, off-policy reinforcement learning algorithm tailored for continuous action spaces. It is particularly well-suited for the adaptive cruise control (ACC) task, where the action—the acceleration command—is a real-valued quantity constrained within physical limits. The agent aims to learn a deterministic policy $\mu_{\boldsymbol{\phi}} : x \mapsto u$ that maps the current state to an optimal action. The objective is to maximize the expected discounted return,

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right],$$

where r_t is the reward at time t, defined as the negative of the stage-cost $\ell(x_t, u_t)$ described in equation (3). The use of a deterministic policy eliminates the need to integrate over actions when computing gradients, which simplifies training and reduces variance.

DDPG employs two neural networks: an actor $\mu_{\boldsymbol{\phi}}(x)$ that outputs the action, and a critic $Q_{\boldsymbol{\theta}}(x, u)$ that estimates the value of taking action u in state x. The actor is updated by ascending the deterministic policy gradient [5]:

$$\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) = \mathbb{E}_{x \sim \mathcal{D}} \left[\nabla_u Q_{\boldsymbol{\theta}}(x, u) \big|_{u = \mu_{\boldsymbol{\phi}}(x)} \nabla_{\boldsymbol{\phi}} \mu_{\boldsymbol{\phi}}(x) \right],$$

where $\mathcal{D}$ is a replay buffer storing past transitions (x, u, r, x'). To train the critic, we define target values using slowly updated target networks $\mu_{\boldsymbol{\phi}^-}, Q_{\boldsymbol{\theta}^-}$:

$$y_i = r_i + \gamma Q_{\boldsymbol{\theta}^-}\left(x_i', \mu_{\boldsymbol{\phi}^-}(x_i')\right),$$

and minimize the mean squared temporal difference error,

$$L(\boldsymbol{\theta}) = \frac{1}{B}\sum_{i=1}^{B}\left(y_i - Q_{\boldsymbol{\theta}}(x_i, u_i)\right)^2,$$

where B is the mini-batch size. The target networks are updated via soft updates:

$$\boldsymbol{\theta}^- \leftarrow (1-\tau)\boldsymbol{\theta}^- + \tau\boldsymbol{\theta}, \qquad \boldsymbol{\phi}^- \leftarrow (1-\tau)\boldsymbol{\phi}^- + \tau\boldsymbol{\phi}.$$

To ensure stable training, several techniques are used. A replay buffer of size 5×10^5 breaks temporal correlations and improves data efficiency. Target networks mitigate instability due to bootstrapping and function approximation. Exploration is encouraged by adding zero-mean Gaussian noise $\mathcal{N}(0, \sigma^2)$ to the actor output, with $\sigma = 0.02$ tuned for the acceleration range. Gradients are clipped to $\|g\|_2 \leq 10$ to prevent exploding updates. The reward signal is clipped to the range $[-1, 0]$ to bound the critic's target and reduce variance.

Training is performed for 10^6 environment steps, corresponding to approximately 5,000 episodes of 20 s duration. The initial state is sampled uniformly: $e_0, e_{v,0} \sim \mathcal{U}[-5, 5]$, and $a_0 \sim \mathcal{U}[-3, 2]$ m/s^2, to ensure generalization across various conditions. Every 10,000 steps, the agent is evaluated deterministically over 20 episodes. The model with the best average return is saved as the final policy.

The actor and critic networks are feedforward multilayer perceptrons with architecture $[3 - 64 - 64 - 1]$, ReLU activations, and trained with the Adam optimizer. A summary of all hyperparameters used is provided in Table I.

TABLE I: DDPG hyperparameters used in training.

Description	Symbol / setting	Value
Discount factor	γ	0.99
Target network update rate	τ	0.001
Actor / critic learning rate	α_a / α_c	10^{-4} / 10^{-3}
Mini-batch size	B	64
Replay buffer size	$\lvert\mathcal{D}\rvert$	5×10^5
Exploration noise (std. dev.)	σ	0.02
Gradient clipping	$\|g\|_2$	≤ 10
Total training steps	–	10^6
Hidden layers (actor / critic)	–	[64, 64] with ReLU
Normalization	–	off
Optimizer	Adam	$\beta_1 = 0.9,\ \beta_2 = 0.999$

The resulting policy achieves reliable performance across 75 validation initial conditions [6]. In all cases, the agent reduces the gap error to below 0.3 m within 6 s, while staying within acceleration and jerk comfort bounds. This demonstrates the capability of DDPG to approximate high-performance, real-time control policies without explicit model knowledge. [6]

III. Results

This section presents the control performance of the DDPG, MPC and benchmark solution for ACC task under different testing scenarios. All evaluations are conducted using the COM-based system model described in Section II-A. We first test the algorithms/controllers using identical initial conditions to access tracking behaviors and optimality. Then we analyze generalization with considering delays off-training-range initial conditions and multiple initial conditions.

A. Testing without modeling errors: single initial condition

We begin with a case study under ideal conditions to compare time-domain performance across controllers. The initial condition is fixed at $[e_0, e_{v,0}, a_0] = [5\,\text{m}, 5\,\text{m/s}, 0\,\text{m/s}^2]$, which is within the range used during the training of the DDPG agent. All controllers are tested on the exact COM dynamics.

Figure 2 illustrates the closed-loop response in terms of gap error, follower speed, acceleration, and jerk. All controllers maintain stability and regulate the gap within comfort limits. The MPC controller with prediction horizon $N = 50$ (5 s) closely matches the IPO solution, while the DDPG policy exhibits slightly more aggressive control and jerk profiles, reflecting its model-free nature.

TABLE II: Comparison of episode costs, performance gap vs IPO, and CPU time.

Controller	Cost	CPU time [s]
DRL	7.04	0.08
MPC (2.8 s)	1029.39	420.25
MPC (3 s)	795.00	608.12
MPC (5 s)	0.62	763.26
IPO	0.00	20.60

These results show that DDPG performs comparably with MPC with a sufficiently long horizon, achieving near-optimal control cost under ideal conditions.

B. Testing without modeling errors: multiple initial conditions

To evaluate generalization, while original paper simulate all controllers over a grid of 75 initial conditions for the sake of time we considered 8 different conditions. These include: - *Normal car-following*: initial states within the training range,

$$e_0, e_{v,0} \in [-5, 5]\,\text{m}, \qquad a_0 \in [-3, 2]\,\text{m/s}^2,$$

- *Cut-in scenarios*: negative gap errors beyond the training range,

$$e_0 \in [-20, -10]\,\text{m}, \qquad e_{v,0} \in [-5, 5]\,\text{m/s}, \qquad a_0 \in [-3, 2]\,\text{m/s}^2.$$

The episode cost for each controller is computed for all 8 cases and then averaged. Table III summarizes the average episode costs for both scenario types.

Although DDPG performs well within the training range above IPO), its cost increases significantly in out-of-distribution conditions (cut-in), confirming the typical generalization issues of neural-network-based policies. In contrast,

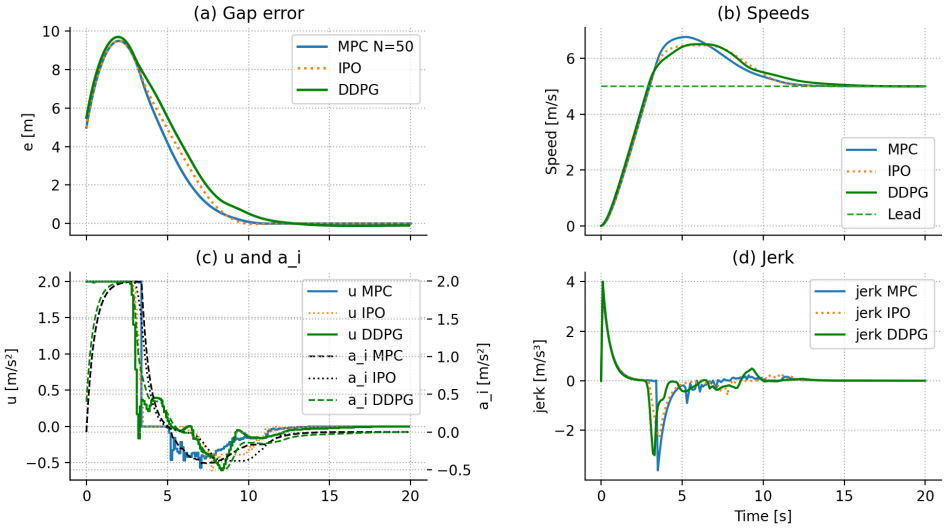


Fig. 2: Closed-loop response from identical initial condition ($e_0 = 5\,\mathrm{m}$, $e_{v,0} = 5\,\mathrm{m\,s^{-1}}$, $a_0 = 0\,\mathrm{m\,s^{-2}}$) for IPO, MPC ($h = 5\,\mathrm{s}$), and DDPG controllers.

TABLE III: Average episode costs over 8 initial conditions.

Controller	Normal Case	Cut-In Case	Diff. (cut-in)
IPO (Benchmark)	13.21	15.00	–
MPC ($N = 50$)	13.22	15.01	$< 0.1\%$
DDPG	13.98	17.58	$+17.2\%$

the model-based MPC controller retains optimality even for unseen gap errors.

C. Testing with Modeling Errors: Control Delays

To evaluate robustness, we introduce an actuation delay τ_d into the COM model, where the control command u_k is applied after a delay of τ_d seconds.

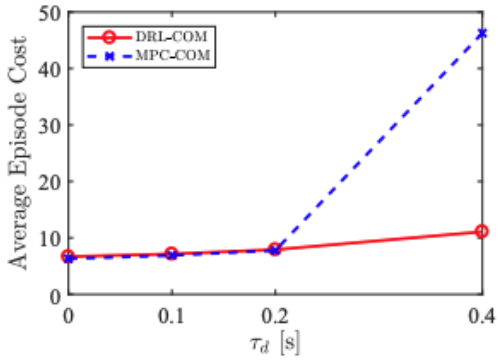
While the original paper performs this simulation under idealized conditions (within the training distribution), we consider more challenging non-ideal initial conditions and compare the effect of control delays, as shown in Fig. 3. We test three values:

$$\tau_d \in \{0.1\,\mathrm{s}, 0.2\,\mathrm{s}, 0.4\,\mathrm{s}\},$$

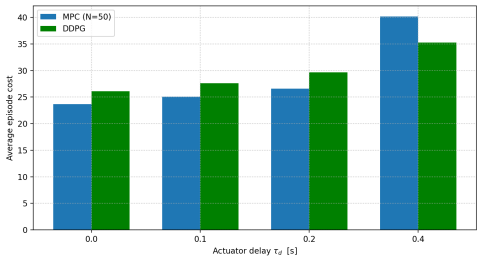
and measure the average episode cost across the same 8 initial conditions.

Figure 3 summarizes the results. For small delays ($\tau_d \leq 0.2\,\mathrm{s}$), both DDPG and MPC perform similarly. However, for a larger delay ($\tau_d = 0.4\,\mathrm{s}$), the MPC controller exhibits oscillatory behavior due to prediction mismatch, while the DDPG controller, trained without relying on an explicit model, remains stable and achieves a lower episode cost. Although in our non ideal simulations of DDPG we see that for lower delays MPC outperforms DDPG as expected, yet similar behavior to ideal case seen in higher control delay.

These results suggest that although MPC performs better under ideal modeling assumptions, DDPG may exhibit greater



(a) Average episode cost vs. control delay τ_d for **ideal** initial conditions.



(b) Average episode cost vs. control delay τ_d for **non-ideal** initial conditions.

Fig. 3: Comparison of control delay performance under ideal and non-ideal initial conditions.

robustness to modeling imperfections such as unmodeled actuation delays.

IV. Conclusion

In this study we have implemented and compared MPC and DDPG for adaptive cruise control under same cost, identical state-space dynamics and consistent test conditions. Both algorithms/controllers operate on a common COM-based low-dimensional vehicle model and are evaluated under ideal/non-ideal conditions to asses control quality, robustness and efficiency.

In ideal settings where the test scenarios are in the training distribution, DDPG achieves a performance similar to that of MPC with a sufficiently long prediction horizon. In particular, its episode cost remains with in $5.8\,\%$ of the infinite-horizon optimal benchmark (IPO), shows the ability of DDPG to approximate optimal closed-loop behavior. Furthermore, the computational cost of DDPG is enormously lower than that of MPC, enabling more efficient real-time control on embedded hardware.

However, when test conditions deviate from the training bounds, cut in scenarios, or with large initial gaps, DDPG exhibits performance degradations, considering the generalization limitations of statistical learning controllers. In contrast, MPC maintains stable performance, benefiting from its explicit use of the model at run time.

With introduced modeling errors, specifically, actuation delays, DDPG outperforms MPC in cases where prediction mismatch severely effects optimization quality. Even though MPC suffers from wakened control behavior, including oscillations, DDPG remains stable and maintains lower episode cost. This shows a key distinction: DDPG's policy implicitly considers model imperfections during training, allowing for better tolerance to unmodeled dynamics. This observation is supported by the DDPG algorithm's reliance on stochastic environment sampling during training, which effectively regularizes the learned policy against a distribution of dynamics.

Feature work will focus on extending the comparison between DRL methods and MPC for higher-dimension control systems, including lateral dynamics and lane change scenarios, where complexity and coupling effects become significant. In the original paper, HFM are tested and broad analyses done by the authors. Thought on the model-based side, future studies need to be explored for robust and stochastic variants of MPC to evaluate performance under uncertainty and disturbance. Another direction to investigate, proposed in original paper, involves hybridizing MPC and DRL by using DDPG policies to warm start optimization or integrating safety filters into DRL to enforce hard constants. Finally, a deeper theoretical investigation into the relationship between deep reinforcement learning and optimal control is needed to understand the stability, approximation quality, and generalization properties of neural network policies.

In general, we showed that even its model free, DDPG can be applicable as a practical alternative to MPC for real-time control tasks, specifically when model fidelity is low or uncertain or when computational resources are limited such as embedded systems.

References

[1] S. Tajeddin, S. Ekhtiari, M. Faieghi, and N. L. Azad, "Ecological adaptive cruise control with optimal lane selection in connected vehicle environments," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 12, pp. 4514–4523, 2019.

[2] J. Ploeg, B. T. Scheepers, E. V. Nunen, N. V. de Wouw, and H. Nijmeijer, "Design and experimental evaluation of cooperative adaptive cruise control," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2011, pp. 260–265.

[3] A. D. Ames, Y. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 66, no. 8, pp. 3955–3970, 2021.

[4] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[6] Y. Lin, J. McPhee, and N. L. Azad, "Comparison of deep reinforcement learning and model predictive control for adaptive cruise control," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 4, pp. 645–654, 2020.