DOI: https://doi.org/10.62985/j.huit_ojs.vol26.no2E.413

TRAJECTORY OPTIMIZATION OF ROBOTS VIA MODEL PREDICTIVE CONTROL AND REINFORCEMENT LEARNING

Le Anh Hao, Vi-Do Tran*

Ho Chi Minh City University of Engineering and Technology, Viet Nam

*Email: *dotv@hcmute.edu.vn*

Received: 14 January 2026; Revised: 3 April 2026; Accepted: 22 April 2026

ABSTRACT

Trajectory optimization for industrial robots remains a critical challenge due to complex kinematic constraints, environmental disturbances, and strict real-time performance requirements in modern automation systems. This study proposes a hybrid control framework that combines Model Predictive Control (MPC) with Reinforcement Learning (RL) to generate efficient and robust robot trajectories. MPC is employed to perform short-horizon optimization under explicit system constraints, ensuring precise and feasible motion planning, while a Reinforcement Learning strategy based on Proximal Policy Optimization (PPO) is integrated to learn long-term adaptive policies capable of compensating for uncertainties such as sensor noise and variable payload conditions. The synergy between MPC and RL enables improved motion accuracy, faster response, and reduced energy consumption. Simulation and experimental results validate the effectiveness of the proposed approach, demonstrating notable performance improvements over conventional control strategies. Future work will focus on real-time embedded deployment of the framework for intelligent manufacturing applications.

Keywords: Model Predictive Control, Reinforcement Learning, Proximal Policy Optimization.

1. INTRODUCTION

Trajectory optimization is a core problem in robotic systems, directly influencing motion accuracy, efficiency, and operational safety. Modern robotic applications increasingly require control strategies that can handle nonlinear dynamics, strict constraints[1, 2], and environmental uncertainties while remaining computationally tractable for real-time implementation.

Model Predictive Control (MPC) has been widely adopted for robotic trajectory optimization due to its ability to explicitly incorporate system dynamics and constraints within a receding-horizon optimization framework. By continuously replanning future control actions, MPC provides strong performance guarantees and feasibility. However, its effectiveness depends heavily on model fidelity, and performance may degrade in the presence of modeling errors, disturbances, or time-varying operating conditions.

Reinforcement Learning (RL), on the other hand, enables robots to learn control policies through direct interaction with the environment, without requiring an explicit system model [3, 4]. RL is particularly effective in adapting to uncertainties and optimizing long-term objectives. Nevertheless, purely learning-based controllers often lack interpretability and explicit constraint handling, which limits their applicability in safety-critical robotic systems.

Recent studies have sought to combine MPC and RL to leverage their complementary strengths [5]. Many existing approaches learn control policies directly using unstructured neural networks or approximate MPC solutions, which can compromise the inherent optimization structure and stability properties of MPC.

In this study, we propose a structured hybrid MPC–RL framework for robotic trajectory optimization that preserves the predictive optimization structure of MPC while incorporating learning-based adaptability. A Proximal Policy Optimization (PPO) module is integrated to modulate the MPC cost function rather than replacing the controller [6]. This design enables MPC to handle short-term, constraint-aware control, while RL captures long-term performance objectives and uncertainty compensation. As a result, the proposed approach improves robustness and adaptability without sacrificing real-time feasibility. Simulation and experimental results demonstrate superior performance compared to conventional MPC and standalone RL controllers.

2. METHODOLOGY

2.1. State Space Representation

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} \tag{1}$$

where (p_x, p_y) is the position in the Cartesian plane, θ is the heading angle (orientation).

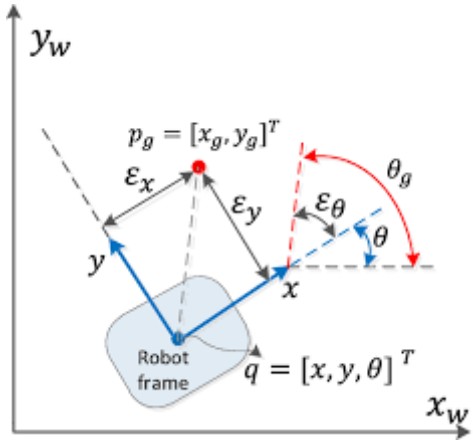

Fig. 1. Diagram illustrating the robot's structure.

Parameter Robot: $v_{\max} = 4.0(m/s)$, $\omega_{\max} = 4.5(m/s)$

The state space is defined as $S \in \mathbb{R}^3$, where $x, y \in [-10, 10]$ meters (the arena boundaries) and $\theta \in [-\pi, \pi]$ radians. The control effort $u = [v, \omega]^T$ is strictly bounded by the maximum values defined in the table above to ensure the physical feasibility of the robot's motors.

2.2. Model Predictive Control

Consider a discrete-time dynamical system with continuous state and input spaces, denoted by $x_k \in \chi$ and $u_k \in \mathrm{U}$ respectively. The time-discretized evolution of the system is described by a function $f : \chi \times U \to \chi$, such that

$$x_{k+1} = f(x_k, u_k), \tag{2}$$

where the index k denotes the system state and control input at discrete time t_k.

The general Optimal Control Problem (OCP) aims to determine an optimal control policy $\pi(x)$, defined as a mapping from the current state to the corresponding control input, $\pi : \chi \to U$, that minimizes a given cost function $J : \chi \to \square^{+}$ Formally, this problem can be written as

$$\pi(x) = \arg\min_{u} J(x) \tag{3}$$

subject to

$$x_0 = x, \quad x_{k+1} = f(x_k, u_k), \quad u_k \in U$$

where the prediction horizon is defined such that $k = 0,...,N$ for the states and $k = 0,...,N-1$ for the control inputs.

Tracking Model Predictive Control: One of the primary applications of Model Predictive Control (MPC) is to regulate the behavior of a dynamical system such that it closely follows a pre-computed reference trajectory. This problem is commonly referred to as trajectory tracking, and the corresponding MPC formulation is known as tracking MPC. the objective J(x) is to minimize a quadratic penalty on the error between the predicted states and inputs, and a given dynamically feasible reference $x_{k,ref}$ and $u_{k,ref}$. Consequently, the cost function J(x) in problem by:

$$J_{tracking} = \sum_{k=0}^{N-1} \left\| x_k - x_{k,ref} \right\|_Q^2 + \left\| u_k - u_{k,ref} \right\|_R^2 + \left\| x_N - x_{N,ref} \right\|_P^2 \tag{4}$$

In this paper, we will search for matrix of Eq. (7) (i.e., Q_k and p_k) using reinforcement learning and differentiable MPC[7].

$$J_{QP}(x) = \sum_{k=0}^{N} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T Q_k \begin{bmatrix} x_k \\ u_k \end{bmatrix} + p_k \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{5}$$

The weight matrix Q is chosen to prioritize state tracking accuracy (time minimization to goal), while R is tuned to penalize large control inputs, thereby ensuring energy efficiency and reducing mechanical wear (path smoothness). In this implementation, R is specifically increased to avoid jerky movements of the differential drive motors.

2.3. Actor-Critic Reinforcement Learning

In this paper, we use a class actor and a differentiable MPC[7] is placed in the final layer[8].

$$\boldsymbol{u_k \sim N\{diffMPC(x_k, Q(s_k), p(s_k)), \Sigma\}}, \tag{6}$$

we construct the matrix $Q(s_k)$ and vector $p(s_k)$ as follows:

$$Q(s_k) = diag(Q(s_k)_{x_1},..., R(s_k)_{u_1},...)$$

$$p(s_k) = \left[p(s_k)_{x_1},..., p(s_k)_{u_1},... \right] \qquad \forall k \in 0,...,T$$

where x_1 ,… and u_1 ,… are states and inputs, $Q(s_k)$ and $p(s_k)$ are the learnable parameters from actor network.

2.4. Simulation setup

A. Observation space:

The observation space (or state space) is a 5-dimensional vector. It provides the Reinforcement Learning agent with a complete representation of the robot's current kinematics and its relative goal.

$$O = \left[x, y, \theta, g_x, g_y \right] \tag{7}$$

where (x, y) represents the robot's Cartesian coordinates in the 2D plane. θ denotes the heading angle (orientation), which is wrapped within the interval $[-\pi, \pi]$ using the $a\tan 2$ logic to maintain continuity. (g_x, g_y) are the coordinates of the target waypoint (goal).

B. Action space:

The action space is defined as a continuous 2-dimensional vector representing the control commands for the differential drive system. To ensure the physical feasibility of the robot's actuators, the agent's output is scaled and clipped by the following maximum velocity constants.

Linear Velocity (v): Scaled within [-4.0, 4.0] m / s^2 .

Angular Velocity (ω): Scaled within [-4.5, 4.5] rad / s

The control input $u = \left[v, \omega \right]^T$ is directly used by the MPC solver as the decision variable to optimize the trajectory over the prediction horizon.

C. Rewards:

The environment employs a multi-objective reward function R to encourage the agent to reach the target state while minimizing energy consumption and maintaining stability. The reward at each timestep t is formulated as:

$$R = \alpha_p.(d_{t-1} - d_t) - \alpha_a.(v^2 + \omega^2) + R_{bonus} \tag{8}$$

Progress Reward $(\alpha_p.\Delta d)$ **:** Encourages the robot to reduce the Euclidean distance to the goal. A high gain of $\alpha_p = 5.0$ is used to ensure strong gradient signals.

Action Penalty $(\alpha_a.\|u\|^2)$:Penalizes excessive control effort with $\alpha_a = 0.01$ to prevent jerky maneuvers and reduce mechanical wear.

Goal Bonus R_{bonus} : A sparse reward of $+10.0$ is granted when the robot reaches the target within a radius of 0.03m, signaling successful task completion.

D. Environment Setup:

The simulation environment is implemented using Visual Studio Code (VS Code) as the primary development platform, instead of relying on an external simulation environment. The environment is designed directly from the underlying dynamic system model, allowing full control over the system dynamics, constraints, and simulation parameters.

Specifically, the environment encapsulates the discrete-time dynamics of the robot system and provides a structured interface for control and learning algorithms. In the environment setup, the following core components are defined:

a) State transition function: Implements the system dynamics based on the discretized model, mapping the current state and control input to the next state.

b) Observation function: Defines how the internal system state is mapped to observable quantities available to the controller or learning agent. This function allows flexibility in selecting full-state or partial-state observations.

c) Action space: Specifies the admissible control inputs and enforces input constraints.

d) Reward (cost) function: Computes the instantaneous reward (or cost) used for reinforcement learning and performance evaluation.

e) Termination conditions: Determines episode termination based on time horizon, constraint violation, or task completion.

3. RESULTS

Before to evaluate the effectiveness of the proposed AC-MPC framework, we conducted a comparative study against a traditional MPC baseline. The target was set at a fixed position $g = [1.0, 1.0]$ with an initial robot state at the origin $[0, 0, 0]$.

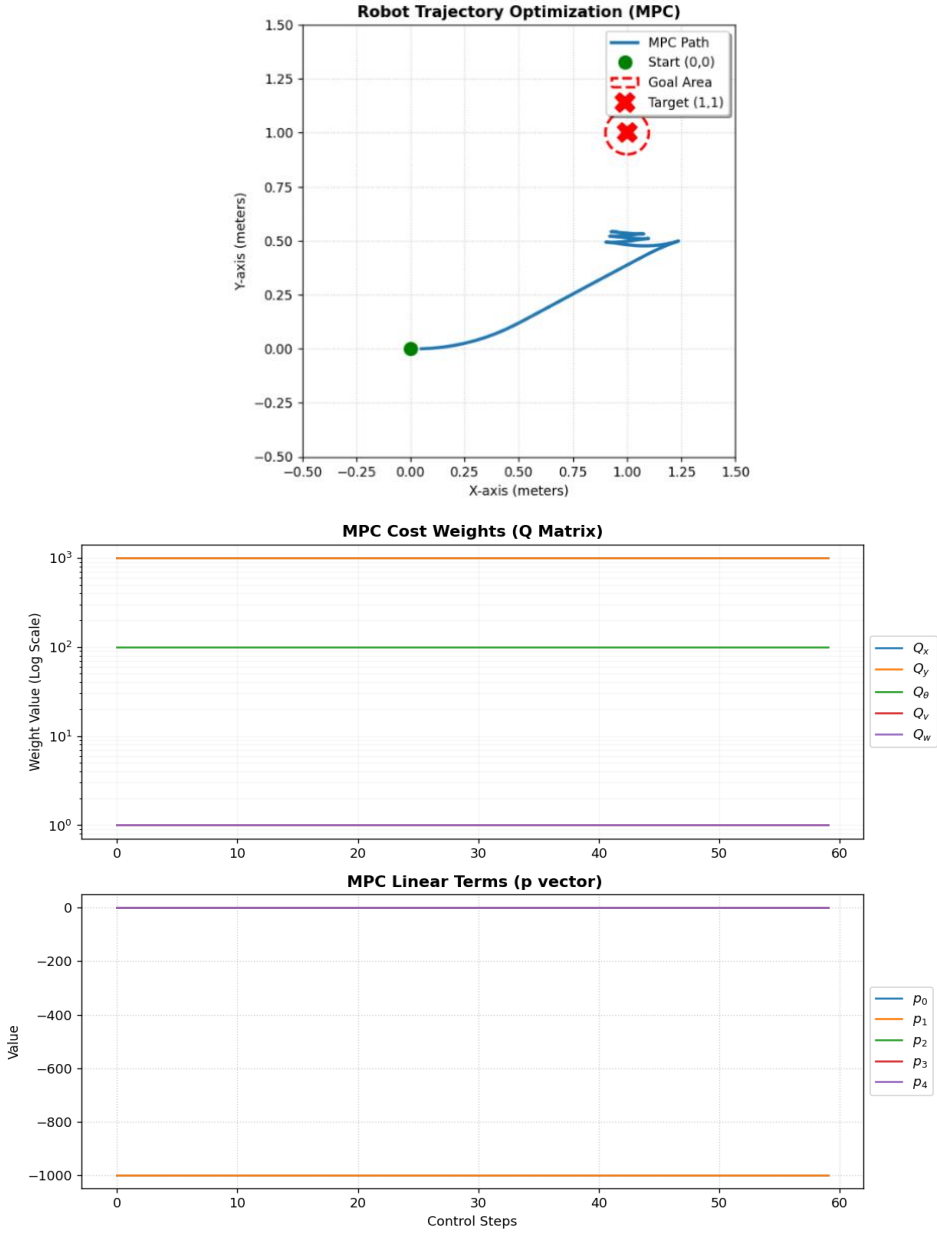

Fig. 2. Traditional MPC with N=5: The robot fails to reach the goal due to static weighting and short look-ahead capability

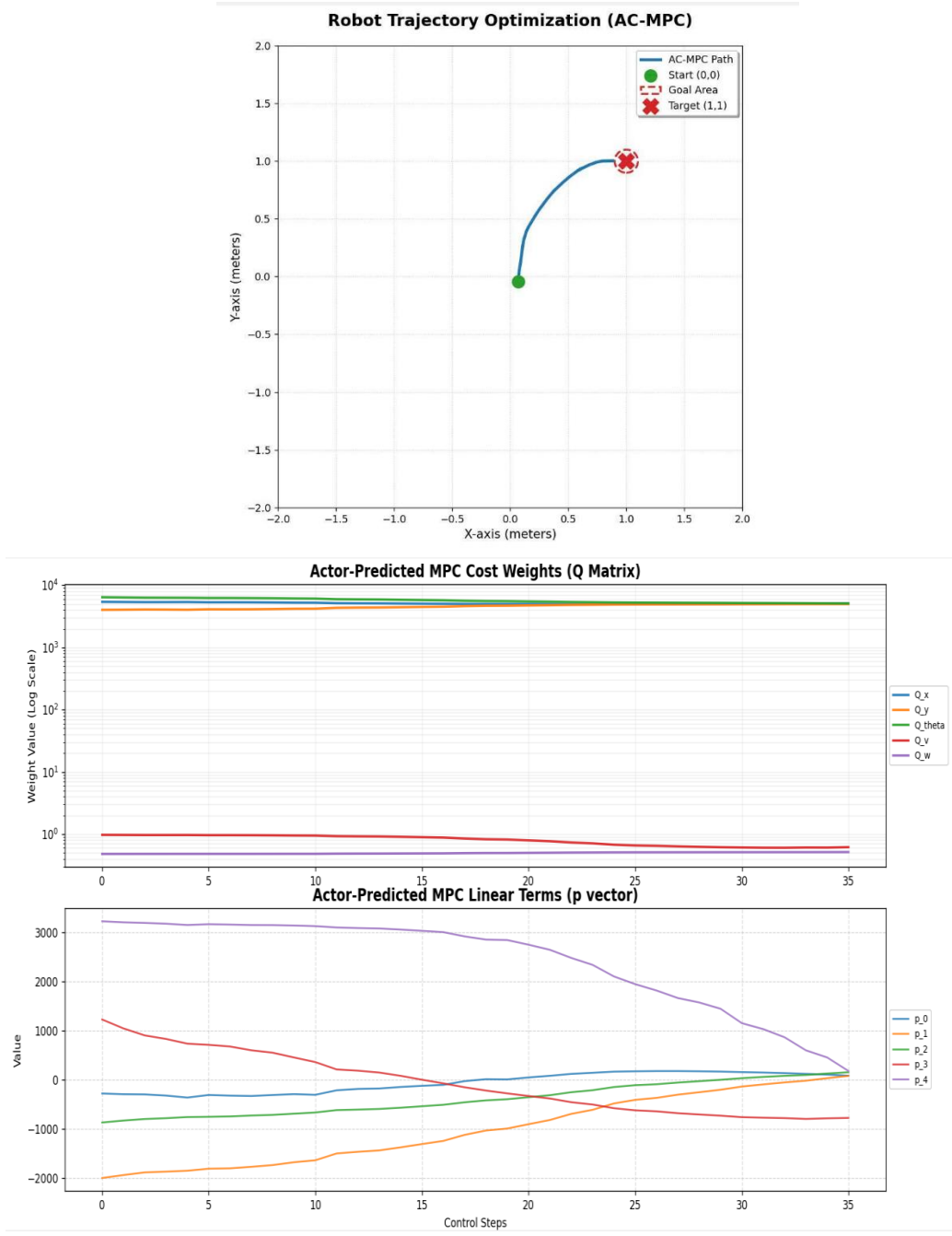

Fig. 3. Proposed AC-MPC with N=5: The Actor network dynamically adjusts MPC parameters, allowing successful goal reaching with the same computational horizon, maintaining real-time performance at 49 FPS.

Table 1. Solve times and inference times for different variations

Horizon	Training Time	Inference time
AC-MPC (N=5)	3h:55m	20 ms/step
AC-MPC (N=15)	7h:10m	33.33 ms/step

A prediction horizon of N=5 was selected based on empirical testing. As shown in Table 1, this value provides a balance between computational efficiency and control performance. Increasing N would lead to higher 'Solve times' per step, potentially compromising real-time feasibility without significant gains in accuracy.

4. CONCLUSION

In this study, an Adaptive Control-based Model Predictive Control (AC-MPC) scheme has been developed for trajectory tracking of robotic manipulators operating under parametric uncertainties and external disturbances. The proposed framework incorporates an adaptive mechanism into the MPC formulation, enabling online compensation of uncertain dynamic parameters without compromising the predictive optimization and constraint-handling capabilities of the system. Simulation results indicate that this approach significantly improves tracking accuracy and robustness compared to conventional MPC and fixed-parameter control strategies; specifically, the controller achieves faster error convergence and lower steady-state tracking errors while ensuring smooth control inputs and strict satisfaction of system constraints. Furthermore, the adaptive component allows for real-time adjustment of model parameters, enhancing the controller's ability to cope with variations in operating conditions and demonstrating that the AC-MPC framework offers an effective, reliable solution for manipulators in dynamically changing environments. Future research will investigate the extension of this method to multi-objective MPC formulations, alongside real-time implementation and experimental validation on multi-degree-of-freedom manipulator platforms.

Acknowledgment: This work was conducted in the Artificial Intelligence for Automation and Control Laboratory (AIAC Lab D203A) with support from Ho Chi Minh City University of Technology and Engineering, Viet Nam.

REFERENCES

[1] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2020. https://cir.nii.ac.jp/crid/1971150415095544201.

[2] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*: Springer, 2007, pp. 207-226. https://link.springer.com/chapter/10.1007/BFb0109870.

[3] R. S. Sutton and A. G. Barto, "*Reinforcement Learning: An Introduction,*" in IEEE Transactions on Neural Networks, vol. 9, no. 5, pp. 1054-1054, Sept. 1998, doi: https://doi.org/10.1109/TNN.1998.712192.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research,* vol. 32, no. 11, pp. 1238-1274, 2013, doi: https://doi.org/10.1177/0278364913495721.

[5] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," *Advances in neural information processing systems,* vol. 30, 2017, doi: https://doi.org/10.48550/arXiv.1705.08551.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347,* 2017, doi: https://doi.org/10.48550/arXiv.1707.06347.

[7] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," *Advances in neural information processing systems,* vol. 31, 2018, doi: https://doi.org/10.48550/arXiv.1810.13400.

[8] A. Romero, Y. Song, and D. Scaramuzza, "Actor-critic model predictive control," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 14777-14784, doi: https://doi.org/10.1109/ICRA57147.2024.10610381.

[9] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems,* vol. 12, 1999. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html.