Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 8980664, 14 pages
https://doi.org/10.1155/2022/8980664

Research Article

U-Model-Based Adaptive Sliding Mode Control Using a Deep Deterministic Policy Gradient

Changyi Lei[1] **and Quanmin Zhu**[2]

[1]*King's College London, WC2R, 2LS, London, UK*
[2]*University of the West of England, BS16, 1QY, Bristol, UK*

Correspondence should be addressed to Changyi Lei; changyi.lei@kcl.ac.uk

Received 9 February 2022; Revised 23 August 2022; Accepted 21 September 2022; Published 7 October 2022

Academic Editor: Abdul Qadeer Khan

This paper presents a U-model-based adaptive sliding mode control (SMC) using a deep deterministic policy gradient (DDPG) for uncertain nonlinear systems. The configuration of the proposed methodology consisted of a U-model framework and an SMC with a variable boundary layer. The U-model framework forms the outer feedback loop that adjusts the overall performance of the nonlinear system, while SMC serves as a robust dynamic inverter that cancels the nonlinearity of the original plant. Besides, to alleviate the chattering problem while maintaining the intrinsic advantages of SMC, a DDPG network is designed to adaptively tune the boundary and switching gain. From the control perspective, this controller combines the interpretability of the U-model and the robustness of the SMC. From the deep reinforcement learning (DRL) point of view, the DDPG calculates nearly optimal parameters for SMC based on current states and maximizes its favourable features while minimizing the unfavourable ones. The simulation results of the single-pendulum system are compared with those of a U-model-based SMC optimized by the particle swarm optimization (PSO) algorithm. The comparison, as well as model visualization, demonstrates the superiority of the proposed methodology.

1. Introduction

A U-model is a generic and systematic control method that was proposed by Zhu et al. [1]. Different from other model-based and model-free control methods, it is a model-independent method in that it uses the dynamic model of the plant to design the controller, while the final performance is independent of the target plant. In doing this, the U-model provides a general routine to separate the system design and control design processes [2]. The gist of the U-model lies in designing a robust dynamic inverter that transforms the original plant into an identity matrix [2, 3]. This brings about two advantages. First, by cancelling the dynamics and nonlinearity, the overall system performance can be prescribed by a unit-negative feedback loop. Besides, the phase delay between the control and the output is eliminated, increasing the response speed of the system [4]. Due to its critical importance, the U-model has been combined with other control methods and has yielded a satisfying outcome. For examples, a U-model-based adaptive neural network [5], a U-model-based predictive control [6], a U-model-based fuzzy PID control [7], etc. However, the conventional U-controller has some drawbacks [8]. Firstly, it does not take into account disturbances and uncertainties. Second, difficulties in calculating dynamic inversion in continuous-time make it hard to be applied to continuous-time systems. Finally, the complexity of the U-control inverter depends on the target plant itself. If the plant is itself complex, then the U-model inverter is also hard to calculate. Therefore, finding a robust and simple dynamic inverter that can be concisely applied to continuous-time systems is a critical criterion for the success of the U-model.

A sliding mode control (SMC) is a robust nonlinear controller. Its implementation is usually based on the Lyapunov stability theorem and is distinguished from other controllers by its discontinuity [9]. By constructing a sliding mode variable, it forces the state variables of the system to slide to equilibrium within a given trajectory. The outstanding characteristics of the SMC are robustness, quick response, and easy implementation [10]. Indeed, the SMC makes a good supplement to a U-model control, and a combination of those two methods has attained certain accomplishments [11, 12]. However, its discontinuity in rationale is afflicted by the chattering problem, which has been widely studied [13]. The chattering issue not only impairs the system's performance but also causes damage to physical instruments. Some solutions have been proposed to alleviate the chattering problem of SMC, including the fuzzy system [14], the boundary layer method [15], the PID-based method [16], high-order SMC [17], a neural network and PSO [18], etc. Nonetheless, these approaches either compromise stability and precision or require tedious craftsmanship and expert experience.

Reinforcement learning (RL) is a model-free methodology that optimizes its action on large-scale, complex problems through exploration and exploitation without explicit models [19]. Recently, with the development of deep learning, RL has been combined with deep neural networks to solve many control problems [20–22]. Actor-critic learning is one popular framework of RL. Compared with classical Q-learning [23] and deep Q-learning [24], the actor-critic (AC) framework works with continuous states and action space [25]. This is realized by using an actor network to output continuous action and a critic network to estimate the Q-value. This characteristic guarantees AC's potential in combinatorial optimization problems [26–28]. Deep deterministic policy gradient (DDPG) is based on AC and was proposed in 2016 [29]. The appearance of DDPG enables the direct estimation of continuous action output in the RL realm. As its name suggests, the DDPG outputs a deterministic policy to the agent, with random noise added for exploration. The DDPG has been successfully implemented in many control scenarios [30–32].

Based on the above discussion, the U-model framework has the potential to bridge the gap between linear and nonlinear systems, provided a robust dynamic inverter can be designed. An SMC is a special nonlinear control scheme that is highly regarded for its robustness. Although a combination of the U-model and SMC has been implemented and has yielded certain success, the chattering problem of SMC still an urgent need to be solved. Therefore, figuring out how to ease the chattering of SMC while maintaining its deserved performance is still challenging. In this paper, we propose a U-model-based adaptive SMC tuned by DDPG to tackle this problem. The parameter tuning problem of SMC is modeled as a combinatorial optimization problem, which is to be solved by DDPG. During the training phase, the DDPG undertakes exploration and exploitation to learn optimal action based on the current state automatically. Through penalizing the tracking error and DDPG output, the neuro network tries to minimize the error with minimal cost. Thus, the proposed adaptive SMC can attenuate the chattering issue without loss of stability or precision. Besides, this method

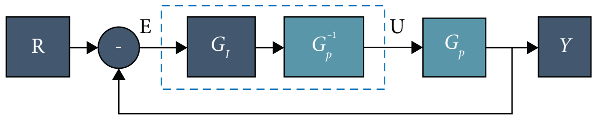

FIGURE 1: A general U-model control framework.

does not require an estimation of the upper bound of the overall disturbance.

The contribution of this paper can be summarized as follows:

(1) An adaptive SMC with variable thickness of the boundary layer, implemented as the dynamic inverter, based on DDPG, is proposed.
(2) An SMC optimized by the PSO algorithm is provided as the baseline for comparison.
(3) A nonlinear single-pendulum environment revised from Gym [33] is provided for simulation.
(4) The simulation tests are conducted to illustrate the advantages and rationality of the proposed method.
(5) Explainable artificial intelligence (XAI) methods are implemented to explain the trained DDPG model.

The rest of this paper is organised as follows. Section 2 gives some preliminaries about the U-model framework and DDPG. Section 3 articulates the details of controller calculation step by step, including a conventional SMC, variable thickness of the boundary layer, a DDPG network, and an invariant controller. Section 4 presents the simulation results and analysis of two different target trajectories. The analysis focuses on settling time, accuracy, and chattering suppression. In addition, output visualization and the SHAP method are implemented to better understand the trained network. Section 5 gives a brief conclusion as well as future work suggestions.

2. Preliminaries

2.1. U-Model

2.1.1. U-Model Control Framework. Considering a general U-model control framework as shown in Figure 1. R is the reference signal; Y is the output vector; $E = R - Y$ is the error vector. The middle part of the pathway is composed of the invariant controller G_I, the dynamic inverter G_p^{-1}, and the target plant G_p. The prioritized task of the U-model control is to design a robust dynamic inverter G_p^{-1} that cancels the dynamics of G_p. In other words, if there exists G_p^{-1} such that $G_p^{-1}G_p = I$, and I is a unit matrix, then the overall system performance is only determined by the invariant controller. Assume the desired transfer function G_{ideal}, then the invariant controller can be derived as $G_I = G_{ideal}/1 - G_{ideal}$. Therefore, the implementation of the U-model-based control framework enables the assignment of system performance using linear system theory regardless of the nonlinearity of the target plant [34].

2.1.2. General U-Model Expression. A SISO CT polynomial dynamic system can be expressed as follows[35]:

$$y^{(M)}(t) = \sum_{j=0}^{J} \lambda_j \left(Y_{M-1}, U_{N-1}, \Theta, t\right)\left(u^{(N)}(t)\right)^j, M > N, \quad (1)$$

where $y(t) \in \mathbb{R}$ is the output, $u(t) \in \mathbb{R}$ is the input, and t is the time. $y^{(M)}(t)$ and $u^{(N)}(t)$ are the M^{th} and N^{th} orders of derivative of y and u, respectively. $\lambda_j(t) \in \mathbb{R}$ is a variable that concludes all $Y_{M-1} = [y^{(M-1)}(t), \ldots, y(t)] \in \mathbb{R}^M$, $U_{N-1} = [u^{(N-1)}(t), \ldots, u(t)] \in \mathbb{R}^N$ and Θ. Θ contains all scalar coefficients. Throughout the study, it is assumed that the polynomial systems are strictly proper $(M > N)$, which guarantees the causality of the systems. Accordingly, for linear polynomial systems, $M > N$ indicates when (1) is converted to its Laplace transform, the denominator Laplace polynomial has higher order than the numerator in the resultant transfer function.

Extend (1) to MIMO expression is given as follows:

$$Y^{(M)}(t) = \sum_{j=0}^{J} \Lambda_j(t)\left(U^{(N)}(t)\right)^j, \quad (2)$$

where $Y^{(M)} = [y_1^{(m_1)}(t), y_2^{(m_2)}(t), \ldots, y_a^{(m_a)}(t)]^T$ is a vector containing all outputs.

$U^{(N)} = [(u_1^{(n_1)}(t))^{j_1}, (u_2^{(n_2)}(t))^{j_2}, \ldots, (u_b^{(n_b)}(t))^{j_b}]^T$ is an input vector with the power j of all inputs. m_* is the order of derivative of y_* that is directly related to u, when u_* has the derivative order n_*. $\Lambda_j(t) \in \mathbb{R}^{a\times b}$ is now a matrix, instead of λ_j being a scalar. For simplicity and without loss of generality, we will omit the dependent variables.

Consider a generalised MIMO continuous-time state-space model expression given as follows:

$$\begin{cases} \dot{X}(t) = F(X(t), U(t)), \\ Y(t) = H(X(t)), \end{cases} \quad (3)$$

where $Y \in \mathbb{R}^a$ is the output, $U \in \mathbb{R}^b$ is the input, and $X \in \mathbb{R}^n$. F is the dynamics of the system that updates the state variables, and H calculates the system output. Extend it to MIMO state-space expression [7].

$$\begin{cases} \dot{x}_1 = \sum_{i=0}^{n} \lambda_{1i} f_{1i}(x_2), \\ \dot{x}_2 = \sum_{i=0}^{n} \lambda_{2i} f_{2i}(x_3), \\ \ldots \\ \dot{x}_n = \sum_{i=0}^{n} \lambda_{ni} f_{ni}(u_1, u_2, \ldots, u_b), \\ y_1 = h_1(x_1, x_2, \ldots, x_n), \\ y_2 = h_2(x_1, x_2, \ldots, x_n), \\ \ldots \\ y_a = h_a(x_1, x_2, \ldots, x_n), \end{cases} \quad (4)$$

where λ_{mi} and $f_{mi}, 0 \le i \le n, 0 \le m \le n$ are time-varying parameters, and $h_l, 1 \le l \le a$ is a smooth mapping from state vector to a specific output. Take the following system as an example:

$$\begin{cases} \dot{x}_1 = x_1 + x_1 x_2, \\ \dot{x}_2 = -x_1 + u_1 + x_1 u_2, \\ y_1 = 2x_1, \\ y_2 = 3x_2. \end{cases} \quad (5)$$

Convert it to a U-model expression based on the following absorbing rule:

$$\begin{cases} \dot{x}_1 = \lambda_{10} + \lambda_{11} f_{11}(x_2), \\ \dot{x}_2 = \lambda_{20} + \lambda_{21} f_{21}(u_1, u_2), \\ y_1 = h_1(x_1, x_2), \\ y_2 = h_2(x_1, x_2), \end{cases} \quad (6)$$

where $\lambda_{10} = x_1, \lambda_{11} = x_1, \lambda_{20} = -x_1, \lambda_{21} = 1, f_{11}(x_2) = x_2$, $f_{21}(u) = u_1 + x_1 u_2$, $h_1(x_1, x_2) = 2x_1, h_2(x_1, x_2) = 3x_2$.

2.1.3. U-Model Dynamic Inversion. Using the following equation, the U-model dynamic inversion is calculated as follows[36]:

$$G_p^{-1} \Leftrightarrow U \in Y_d^{(M)}(t) - \sum_{j=0}^{J} \Lambda_j(t)\left(U^{(N)}(t)\right)^j = \mathbb{O}^a, \quad (7)$$

where $Y_d(t)$ is the desired output vector and $\mathbb{O}^a$ is a $a \times 1$ null vector. The prerequisites for the solution to exist are external stability and the nonminimum phase of the system. Reconsidering (4), apply derivative to the $p^{th} (0 \le p \le a)$ output y_p with respect to x_i:

$$\dot{y}_p = \sum_{i=0}^{n} h_p'(x_i)\dot{x}_i. \quad (8)$$

Replace $\dot{x}_i$ with a polynomial equation and we have

$$\dot{y}_p = \sum_{i=0}^{n}\sum_{j=0}^{n} h_p'(x_i)\lambda_{ij} f_{ij}(x_j). \quad (9)$$

Repeating the above derivative and replacement procedures for m_p times. Rearrange the equation and combine similar terms as follows:

$$y_p^{(m_p)} = \sum_{i=1}^{n-1} P_{ip}(X, t)\dot{x}_i + P_{np}(X, t)\dot{x}_n, \quad (10)$$

where X is the vector of state variables, and P is a function of X. Consequently, we need to solve an equation set of (10) to retrieve the final control.

2.2. DDPG. Deep deterministic policy gradient (DDPG) is an off-policy algorithm proposed in 2014 by Silver et al. [37]. DDPG is based on the actor-critic framework, which learns an action network and a Q network simultaneously. For

```
(1) Initialize policy network $\pi_\phi(s)$, critic network $Q_\theta(s,a)$ and empty replay buffer $\mathcal{D}$
(2) Set target policy network $\pi_{\phi'}'(s)$ and target critic network $Q_{\theta'}'(s,a)$, with $\phi' \leftarrow \phi$, $\theta' \leftarrow \theta$
(3) **repeat**
(4)     Observe state $s$ and execute action $a = \text{clip}\ (\pi_\phi(s) + \varepsilon, a_{low}, a_{high})$, where $\varepsilon \sim \mathcal{N}$
(5)     Observe next state $s$', reward $r$, and done signal $d$ to indicate whether s' is terminal
(6)     Store ($s$, $a$, $r$, $s$', $d$) in the replay buffer $\mathcal{D}$
(7)     If $s$' is terminal, reset environment state
(8)     **if** it is time to update **then**
(9)         **for** the number of updates **do**
(10)            Randomly sample a batch of transitions, $B = (s, a, r, s', d)$ from $\mathcal{D}$
(11)            Compute targets $y(r,s',d) = r + \gamma(1-d)Q_\theta'(s',\pi_\phi(s'))$
(12)            Update Q-function by one step of gradient descent using $\nabla_\phi 1/|B| \sum_{(s,a,r,s',d)\in B} (Q_\theta(s,a) - y(r,s',d))^2$
(13)            Update policy by one step of gradient ascent using $\nabla_\phi 1/|B| \sum_{s\in B} (Q_\theta(s,\pi_\phi(s)))$
(14)            Update target networks with $\theta' \leftarrow \rho\theta' + (1-\rho)\theta \phi' \leftarrow \rho\phi' + (1-\rho)\phi$
(15)        **end for**
(16)    **end if**
(17)    **until** convergence
```

Algorithm 1: Deep deterministic policy gradient.

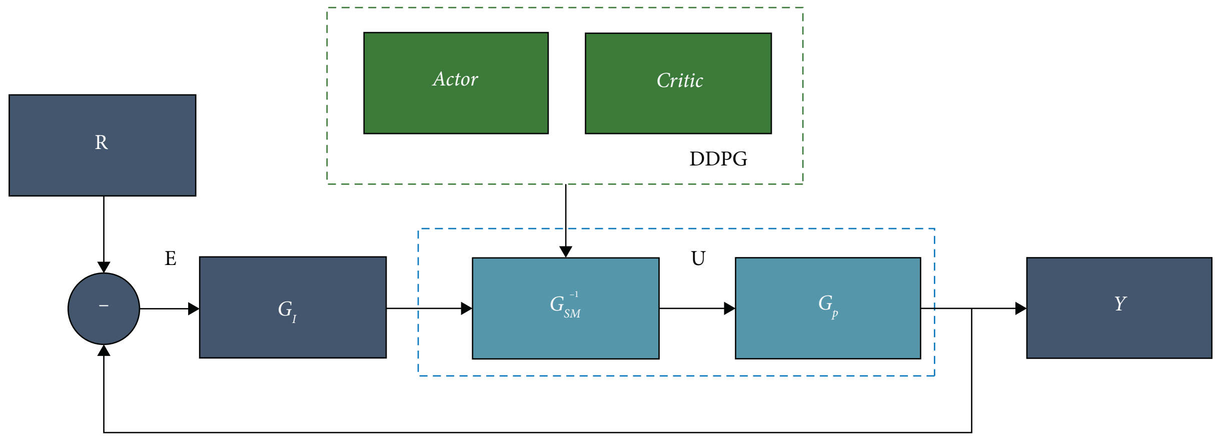

Figure 2: Controller framework overview.

every step, given current states, the actor network outputs the policy added with random noise. By executing the policy, the model receives a reward from the environment, and the policy network is updated using the temporal difference (TD) algorithm accordingly. The Q-value output by the critic network in turn guides the update of the actor network using the policy gradient algorithm [37]. To increase the stability of learning, a target actor network and a target critic network are added. They are updated through a soft update, namely a weighted average of the actor or critic networks and themselves. Besides, a replay buffer storing transition states and actions is implemented to increase the efficiency of sample utilization. Because of this double network configuration, DDPG can deal with situations when the action space and state space are all continuous [29]. The pseudo code of DDPG is shown in Algorithm 1.

3. Controller Design

This section describes the details of the conceptual framework of the controller, which will be implemented in the following simulation. The combined controller is composed of a U-model controller, a sliding mode controller with a variable boundary layer, and a DDPG network.

3.1. Framework Overview. Figure 2 illustrates the workflow of the proposed methodology. $R \in \mathbb{R}^N$ represents the reference signal, where N is the degree of freedom (DoF) of the system. $Y \in \mathbb{R}^N$ is the output vector, and $E = R - Y$ represents the error vector. G_I is the invariant controller, and G_p is the original target dynamic plant. Between them is the dynamic inverter performed by a sliding mode controller, and it outputs the control vector $U \in \mathbb{R}^N$. The parameters of the SMC are calculated by the DDPG module for every time step. Based on the gist of the U-model, if the dynamic inverter G_{SM}^{-1} successfully performs dynamic inversion regardless of uncertainty, then the nonlinearity of the plant is cancelled, which means $G_{SM}^{-1}G_p = I_N$. In this case, the system is equivalent to a unit negative feedback loop, of which the performance is assigned by the transfer function of G_I. In the following part of this section, the details of SMC, DDPG, and G_I will be introduced.

3.2. Sliding Mode Control with Variable Boundary Layer.

In this section, we consider a general second-order dynamic system expression, based on which the SMC with variable boundary layer is designed using backstepping and the Lyapunov stability theorem.

3.2.1. Conventional Sliding Mode Control with Backstepping.

For simplicity and without loss of generality, consider the following single-input-single-output (SISO) dynamic model:

$$\begin{cases} \dot{x}_1(t) = x_2(t), \\ \dot{x}_2(t) = f(x_1, x_2, t) + h(x_1, x_2, t)u(t) + d(t), \end{cases} \tag{11}$$

where x_1, x_2 are state variables, $u(t)$ is the control input vector, and $d(t)$ is the overall disturbance vector. $f(x_1, x_2, t)$ and $h(x_1, x_2, t)$ are time-varying functions dependent on the state variables. For simplicity, the expression afterwards will ignore all dependent variables.

Assumption 1. The overall disturbance $d(t)$ is bounded and satisfies $|d(t)| < D$, in which D is a positively finite scalar.

Remark 1. In practice, the disturbance is often closely related to the state variables of the system. For example, viscous friction is a function of velocity. In practice, the state variables usually have bounds, so the total disturbance is usually bounded. Similar assumptions have been made in many control theory scenarios [38, 39].

The following is the construct virtual control variables:

$$\begin{cases} z_1 = x_1 - z_d, \\ z_2 = x_2 + c_1 z_1 - \dot{z}_d, \end{cases} \tag{12}$$

where $z_d, \dot{z}_d$ forms the reference signal. c_1 is a constant positive value, and z_1, z_2 are virtual control variables. Construct the first partial Lyapunov function is given as follows:

$$V_1 = \frac{1}{2} z_1^2. \tag{13}$$

Take the derivative of (13) and integrate with (12), we have

$$\dot{V}_1 = z_1 \dot{z}_1 = -c_1 z_1^2 + z_1 z_2. \tag{14}$$

According to the Lyapunov stability theorem, since $V_1 \geq 0$, if $z_2 = 0$, then $\dot{V}_1 < = 0$, and that z_1 can converge to the equilibrium asymptotically. Regarding subsystem of (x_1, z_2), design a second partial Lyapunov function as follows:

$$V_2 = V_1 + \frac{1}{2} z_2^2. \tag{15}$$

Take the derivative of (15) and integrate with (11)–(14), we have

$$\dot{V}_2 = -c_1 z_1^2 + z_1 z_2 + z_2 \left(hu + f + c_1 \dot{z}_1 - \ddot{z}_d + d\right). \tag{16}$$

Select z_2 as the sliding mode variable and design control input as

$$u = \frac{1}{h} \left(-\eta \text{sgn}(z_2) - f - c_1 \dot{z}_1 + \ddot{z}_d - c_2 z_2 - z_1\right), \tag{17}$$

where sgn is the sign function. $\eta > D$ is a positive scalar, and is called the switching gain.

3.2.2. Stability Analysis

Theorem 1. *For a general second-order dynamic system as described in (11) and implementing a controller in (17), the system has asymptotic stability in the Lyapunov sense.*

Proof. Integrating (17) into (16), we have

$$\dot{V}_2 = -c_1 z_1^2 - c_2 z_2^2 - \eta |z_2| + z_2 d. \tag{18}$$

According to Assumption 1, $z_2 d < \eta |z_2|$, so $\dot{V}_2 < 0$ and $V_2 \geq 0$. Based on the Lyapunov asymptotic stability theorem [40], z_1 and z_2 will converge to equilibrium asymptotically. Therefore, it means that state variables x_1 and x_2 will follow the desired trajectory. □

3.2.3. Adding Variable Boundary Layer.

The switching gain η plays a critical role in determining the performance of the SMC. If η is large, the system will converge quickly, but the chattering problem is also exacerbated because of the discontinuity of the sign function. To alleviate the chattering issue, the sign function is replaced with a saturation function as follows [10]:

$$sat(s, \delta) = \begin{cases} 1, & s > \delta, \\ \frac{1}{\delta}, & |s| < \delta, \\ -1 & s < -\delta, \end{cases} \tag{19}$$

where s is the sliding mode variable and δ is the thickness of the boundary layer. Thus, (17) becomes

$$u = \frac{1}{h} \left(-\eta sat(z_2, \delta) - f - c_1 \dot{z}_1 + \ddot{z}_d - c_2 z_2 - z_1\right). \tag{20}$$

The introduction of the boundary layer constructs a space where the controller outputs a continuous torque so that the system trajectory can be smoother. However, the alleviation of chattering comes at the cost of lowered control

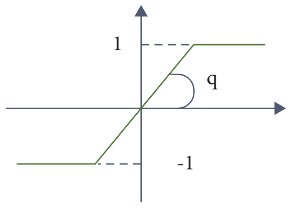

Figure 3: Diagram of saturation function.

accuracy. Figuring out how to select the optimal pair of (η, δ) largely relies on the human experience.

3.3. DDPG Network Module. The purpose of introducing the DDPG module is to adaptively select the optimal pair of (η, δ) for the sliding mode controller so that, when the error is large, the system will converge quickly; whereas when the system is near equilibrium, the chattering can be alleviated while ensuring the same level of accuracy. The DDPG belongs to the actor-critic framework of RL and, therefore, can deal with situations when state and action are all continuous values. In this paper, instead of directly outputting $[\eta, \delta]^T$, it outputs $[1/\eta = \tan(q), \delta]^T$, where $q \in [0, \pi/2]$ is the inclination of the saturation function, as shown in Figure 19.

The actor and the critic network are constructed as fully connected neural networks. The details of the network structure are shown in Figure 4 and 5. The state is a 4×1 vector $[x, \dot{x}, e\_x, e\_\dot{x}]^T$, where $x, \dot{x}$ are state variables and $e\_x, e\_\dot{x}$ are corresponding errors. The output of the actor network is a 2×1 vector $[u_1, u_2]^T = [1/\eta, \delta]^T$. The action ranges of those two outputs are set to $[0, 20]$. The actor network takes the state vector as the input, and then it goes through three fully connected hidden layers with 64 units of nodes. The activation functions of the first two layers are the Rectified Linear Unit (ReLu) function, and the last layer uses the Sigmoid function to map every dimension of the output to $[0, 1]$. The critic network takes the concatenation of state and action as the input. It also uses several fully connected layers to process the information, and then outputs a scalar, which is also called a state-action value $Q(state, action)$. This is an estimate of how well the action is given the current state.

3.4. Invariant Controller. An invariant controller is utilized in the U-model to assign system performance using linear system techniques. It can also be viewd as the implementation of a smooth transition process. Ideally, we hope the system can converge to equilibrium without oscillation or overshoot. Luckily, a critically-damped second-order differential equation meets our requirements and is designed as the invariant controller. The ideal closed-loop system transfer function can be written as follows:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}, \tag{21}$$

where ξ is the damping ratio and ω_n is the natural frequency. They will be designed according to the required stability error and settling time. Also, the invariant controller can be calculated as follow:

$$G_I(s) = \frac{G(s)}{1 - G(s)} = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s}. \tag{22}$$

4. Simulation

4.1. Dynamic Model Establishment. The simulation is implemented on a single pendulum. The structure graph is shown in Figure 6.

θ is the angle of the pendulum, l is the length, m is the weight, and g is the gravity coefficient. For simplicity and without loss of generality, we assume the mass of the pendulum is concentrated at the end of the link, and therefore, the dynamic equation can be derived as follows:

$$\ddot{\theta} = \frac{1}{ml^2}(\tau - b\dot{\theta} - mgl\sin\theta) + d, \tag{23}$$

where $\ddot{\theta}$ is the angular acceleration, $\dot{\theta}$ is the angular velocity, τ is the control torque, d is the disturbance, and b is the damping coefficient. Comparing (23) with (11), we have

$$\begin{cases} f = -\dfrac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta), \\ \\ h = \dfrac{1}{ml^2}. \end{cases} \tag{24}$$

Integrating (24) with (20), we have the sliding mode controller equation for a single-pendulum dynamics as follows:

$$\begin{aligned} \tau = {} & ml^2\left(-\eta sat(z_2, \delta) - c_1\dot{z}_1 + \ddot{z}_d - c_2 z_2 - z_1\right) \\ & - (b\dot{\theta} + mgl\sin\theta). \end{aligned} \tag{25}$$

4.2. DDPG Training Procedure. The training of DDPG follows the procedure of Algorithm 1, and the training-related parameters are shown in Table 1. The network is trained for 5000 episodes, with 200-time steps for each episode. Due to the limited computational resources, the sampling time during training is set to 0.05s. The initial position of the pendulum is randomly set to be between $[-\pi, \pi]^T$ rad, and the initial velocity is between $[-10, 10]^T$ rad/s. The target position, velocity, and acceleration of training are all set to 0. We later verify that, although the target of training is simplistic, the model can learn an effective policy that

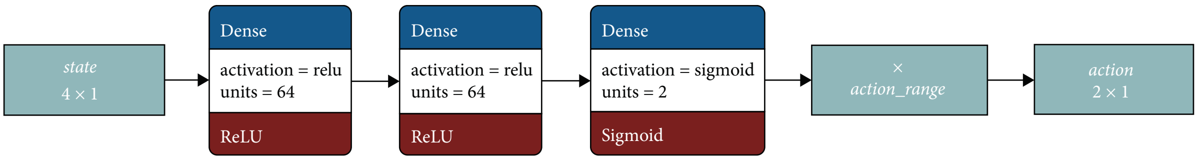

FIGURE 4: Structure of an actor network.

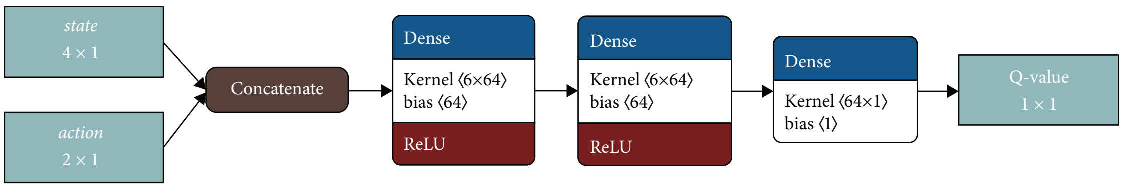

FIGURE 5: Structure of a critic network.

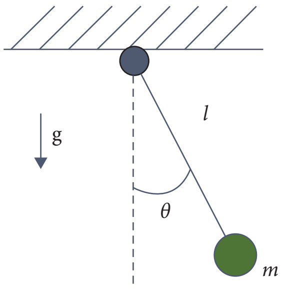

FIGURE 6: Structural graph of a single-pendulum.

TABLE 1: Parameter of controllers.

Controller	Parameters
SMC controller	$c_1 = 2, c_2 = 2$
DDPG	$lr_a = 1e-6, lr_c = 2e-6, \rho = 0.01, \gamma = 0.9$
U-model	$\xi = 1, \omega_n = 5$
PSO	$a_1 = 0.5, a_2 = 0.5, N = 40, \omega = 0.8$

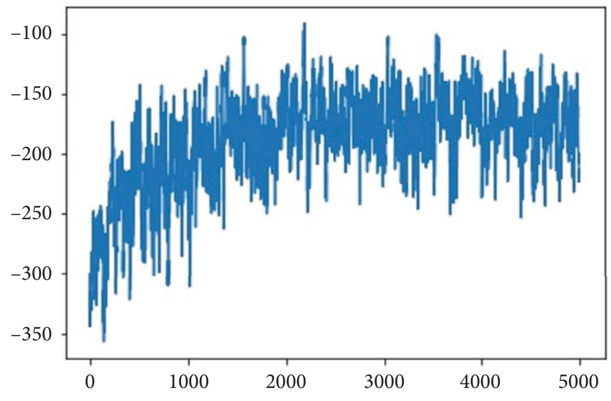

FIGURE 7: Training reward of DDPG.

generalizes to time-varying trajectories (e.g., a sine wave). The state transition reward is designed as follows:

$$\begin{cases} r = -(r_1 + r_2 + r_3), \\ r_1 = (2 \times |e\_\theta| + 0.1 \times |e\_\dot{\theta}|)^2, \\ r_2 = 0.06 \times \arctan(u_1), \\ r_3 = 0.06 \times u_2. \end{cases} \tag{26}$$

The reward is composed of three parts. r_1 takes a similar format as the sliding mode variable. An absolute operation is implemented on the angular error and velocity error, which implies a stricter punishment than the sliding mode variable because it requires both of them to converge to 0 simultaneously. Furthermore, there are slight adjustments to the weighting of the two terms. Since the priority goal is to track the desired position, we hope the reward can guide the model to pay more attention to angular error than velocity error. r_2 calculates the inclination of the saturation function. r_2 and r_3 penalizes the interference of DDPG with the SMC controller. The final constant coefficients are determined through experiments. Intuitively, the model is instructed not to output unnecessarily large parameters to the SMC controller, and so that prevents the chattering issues. Figure 7 illustrates the episode reward during training. We can see that with training, the network gradually finds a nearly optimal policy and optimizes the episode reward.

4.3. PSO Tuning Procedure. PSO is a meta-heuristic global optimization method. It treats the model as a black box and tries to find optimal solutions through inputs and outputs. The update rule of PSO borrows ideas from swarm intelligence. Through interacting with each other, all particles strike a balance between exploration and exploitation. PSO has been proven to be simple in rationale but effective in practice. In this paper, we use PSO to find the optimal parameters for conventional SMC given certain reference signals, which will then be used as the baseline for comparison. The objective of PSO is to find a set of X that produces the minimum value of a so-called cost function $f(X)$. In this paper, the cost function for PSO to optimize is set as follows:

```
(1) Initialization.
(2) for each one of the N particles do
(3)         Initialize the position $x_i$ and velocity $v_i$
(4)         Set particles' best position as current $pbest_i = x_i$
(5)         Calculate the fitness of each particle and set the optimum as the $gbest$
(6) end for
(7) Update.
(8) while stopping Criterion not met do
(9)         Update particles' velocity using $v_i^{k+1} = \omega v_i^k + c_1 R_1 (pbest_i^k - x_i^k) + c_2 R_2 (gbest^k - x_i^k)$
(10)        Update particles' position using $x_i^{k+1} = x_i^k + v_i^{k+1}$
(11)        Evaluate the fitness of all particles $f(x_i^{k+1})$
(12)        If $f(x_i^{k+1}) < pbest$, update individual best $pbest_i^{k+1} = f(x_i^{k+1})$
(13)        If $f(x_i^{k+1}) < gbest$, update global best $gbest^{k+1} = f(x_i^{k+1})$
(14) end while
(15) Output best results
```

ALGORITHM 2: Particle swarm optimization for a minimization problem.

TABLE 2: Parameter of the single-pendulum.

Parameters	Mass (kg)	Length (m)	Damping	Gravity (Nm^2)
Value	$m = 1$	$l = 1$	$b = 0.1$	$g = 9.8$

$$f_{\cos t} = \sum |z_2| \mathrm{d}t = \sum |(x_2 - \dot{z}_d) + c_1 z_1| \mathrm{d}t, \quad (27)$$

which is the sum of the absolute sliding mode variable along one episode. The SMC parameter output by PSO is $0 \le \eta \le 20$ in (17). For a general PSO framework with N particles of the swarm, the position of all particles at k^{th} timestep can be represented as $X = [x_1^k, x_2^k, \ldots, x_n^k]$. The position update function can be written as follows:

$$\begin{aligned} v_i^{k+1} &= \omega \times v_i^k + a_1 \times rand(0,1) \times \left(pbest_i^k - x_i^k\right) \\ &\quad + a_2 \times rand(0,1) \times \left(gbest^k - x_i^k\right), \\ x_i^{k+1} &= x_i^k + v_i^{k+1}. \end{aligned} \quad (28)$$

Among them, v is the velocity vector, which integrates the local best $pbest_i^k$ (the best position that the i^{th} particle ever traversed) and global best $gbest^k$ (the best position that all particles ever traversed). $a_1, a_2 > 0$ are cognitive and social learning coefficients, which are related with local optimum and global optimum, respectively. $\omega \ge 0$ is called inertia weight that adjusts the importance of the previous velocity. The pseudo code of PSO is shown in Algorithm 2.

4.4. Parameters Initialization. The parameters of single-pendulum dynamics are shown in Table 2. The parameters related to DDPG and sliding mode controller are shown in Table 1.

During testing, two target trajectories are assigned. One is a constant value target $[z_d, \dot{z}_d, \ddot{z}_d]^T = [0,0,0]^T$, and the other is a time-varying sine wave target $[z_d, \dot{z}_d, \ddot{z}_d]^T = [2\sin(t), 2cos(t), -2\sin(t)]^T$. The sampling time is $\triangle T = 0.005s$, and one episode takes 4000 steps. The disturbance d is a random value between $[-0.1, 0.1] rad/s^2$. For

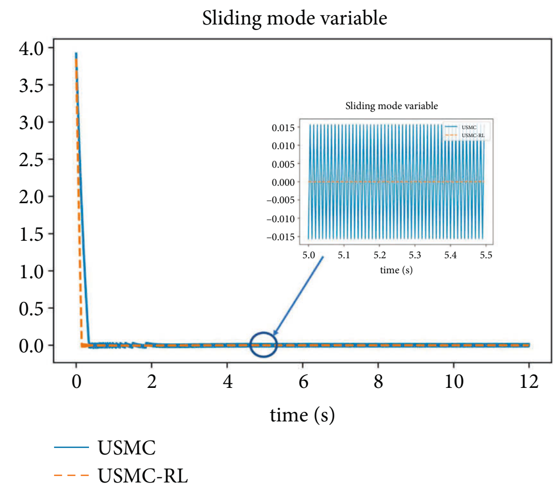

FIGURE 8: Sliding mode variable profile.

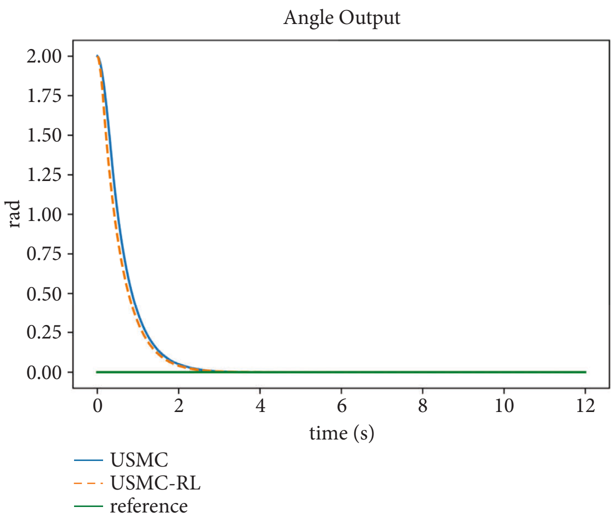

FIGURE 9: Angular profile.

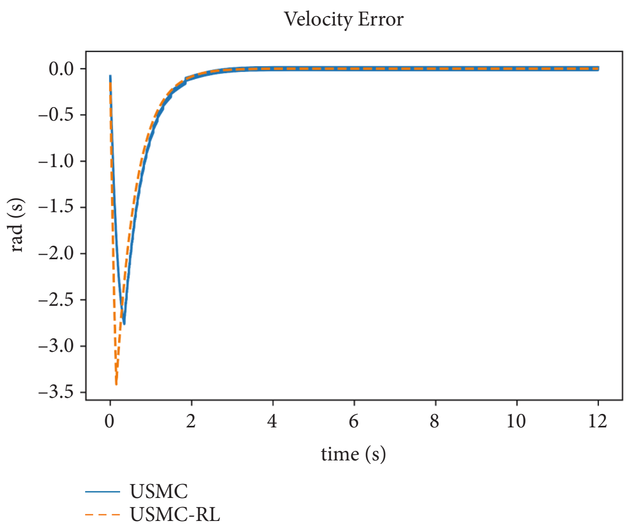

Figure 10: Velocity error profile.

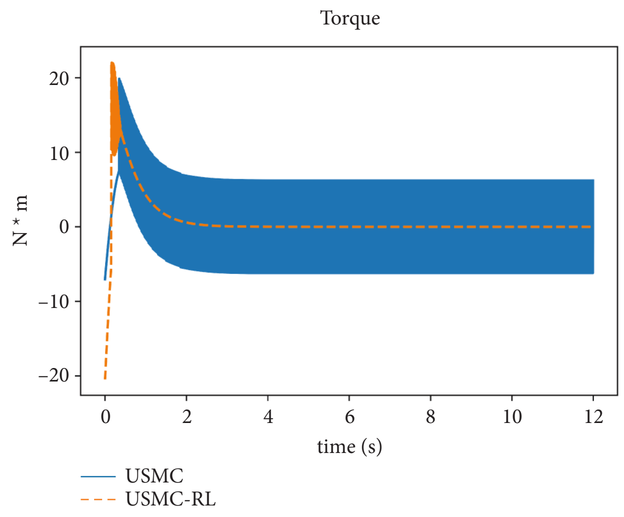

Figure 11: Torque profile.

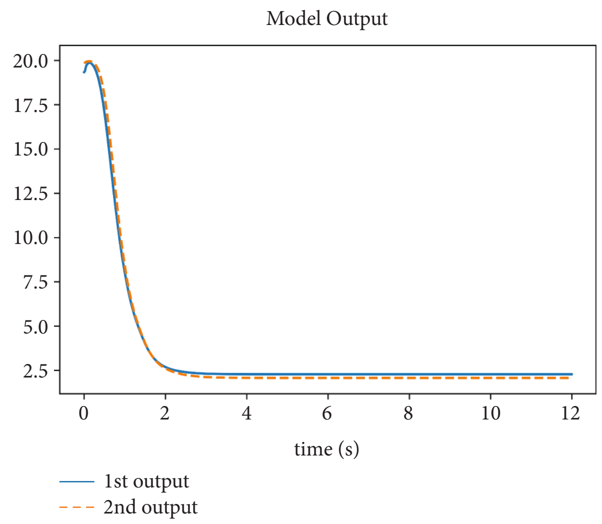

Figure 12: Actor network output.

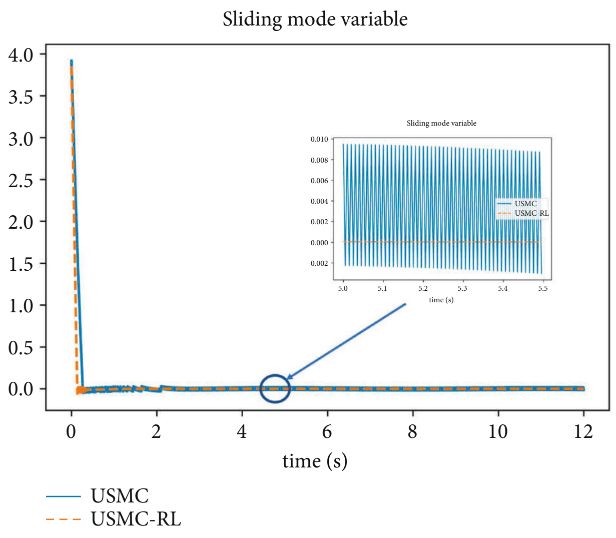

Figure 13: Sliding mode variable profile.

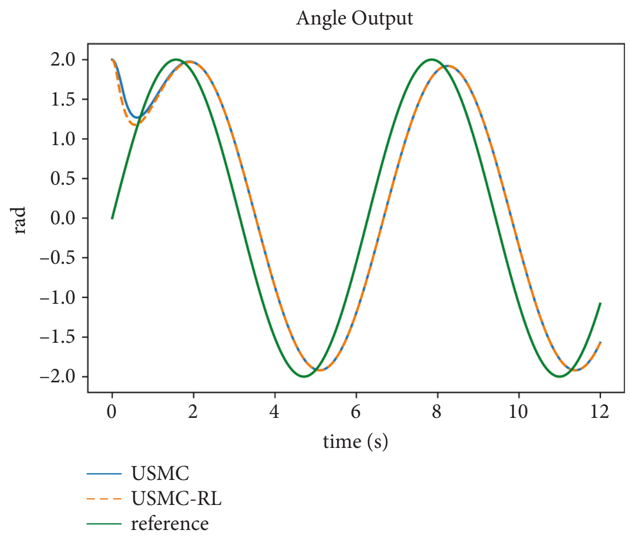

Figure 14: Angular output profile.

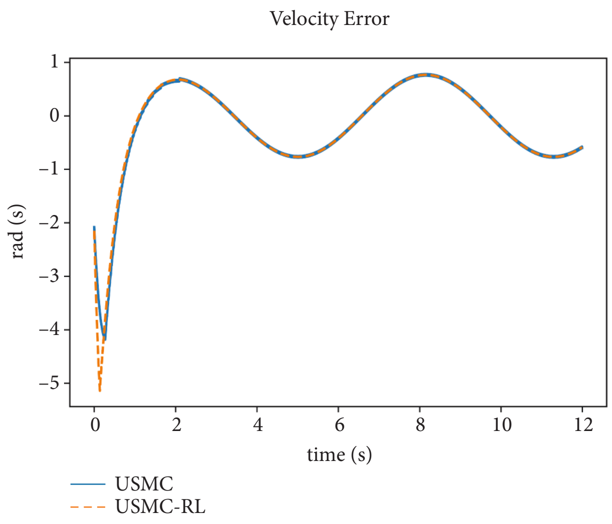

Figure 15: Velocity error profile.

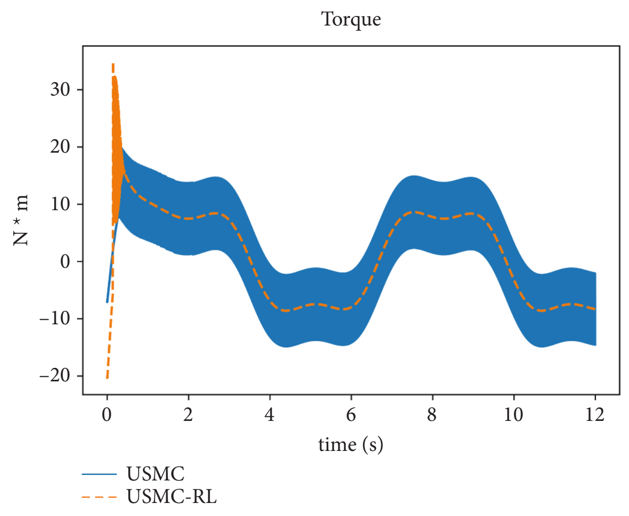

Figure 16: Torque profile.

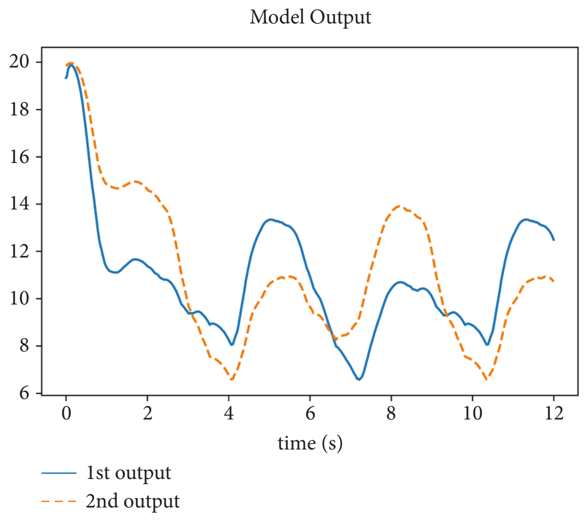

Figure 17: Actor network output.

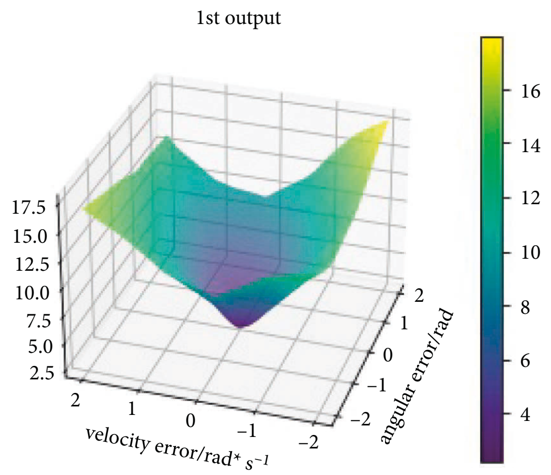

Figure 18: The 1st output of an actor network.

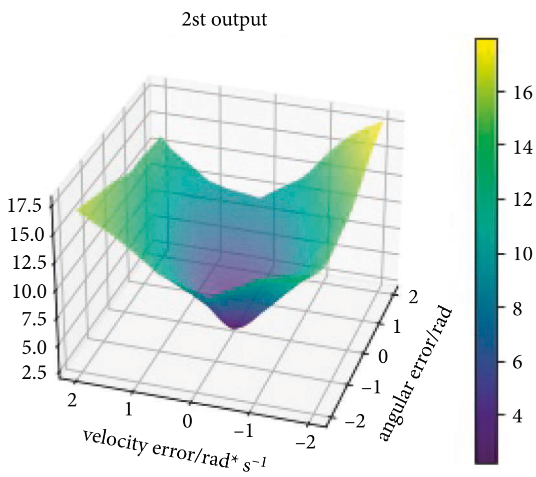

Figure 19: The 2nd output of an actor network.

comparison, the initial state of the system is fixed to $[\theta, \dot{\theta}, \ddot{\theta}]^T = [2, 1, 0]^T$.

4.5. Results and Evaluation. In this section, the testing is implemented using a constant value target and a sine wave target, respectively. The results, including the torque profile, angle profile, model outputs, a sliding mode variable, and velocity error, are presented, with analysis and discussion. The baseline is the conventional sliding mode control optimized by the PSO algorithm, and the optimization function is the sum of absolute sliding mode variables at all time steps, namely $func\_opt = \sum |s|$. For label inside figures, we use "USMC" to represent a U-model-based SMC, and "USMC-RL" to represent a U-model-based adaptive SMC using DDPG. Through testing, the forward calculation time of our deep learning model is only about 600 μs, which satisfies real-time requirements.

4.5.1. Constant Value Target. Figures 8–19 show the simulation result on the tracking control of the constant value target $[z_d, \dot{z}_d, \ddot{z}_d]^T = [0, 0, 0]^T$. The optimal $\eta = 7.60$ for conventional SMC is given by PSO. The sampling time is 0.005 s and the simulation lasts for 2400 steps.

For the sliding mode variable in Figure 8, we can see that USMC-RL renders a shorter settling time than USMC, and that the chattering issue is greatly alleviated in the vicinity of equilibrium. While the chattering amplitude of the USMC is about 0.07, the output of the USMC-RL remains smooth. Figure 9 shows the angle output and shows that USMC-RL converges quicker than USMC. Figure 10 is the profile of velocity error. We can see that, at the beginning, the velocity error of the USMC-RL is greater than that of the USMC. This is because the faster response of adaptive SMC produces a faster speed to converge to equilibrium. This can also be seen in the torque profile in Figure 11 where the USMC-RL performs severe chattering upon arriving at equilibrium. Besides, it is obvious that the USMC renders a much higher chattering amplitude (around 20 Nm) near equilibrium. The output of the model in Figure 19 illustrates that when the error is considerable at the beginning, the outputs of the model are near the upper bound 20; however, when the system is near equilibrium, the outputs are small (about only 2.5). This means that the model successfully learns to speed up the convergence while alleviating chattering according to the current situation. Also, we can explain the chattering upon arriving at the equilibrium of USMC-RL. This is because of the continuity of the model—it cannot jump from a very high output (20) to a very small output (2.5). Therefore, when it should produce a small output, it is still on its way down.

4.5.2. Sine Wave Target. Figures 13–18 show the simulation result of the tracking control of a sine wave target $[z_d, \dot{z}_d, \ddot{z}_d]^T = [2\sin(t), 2cos(t), -2\sin(t)]^T$. The optimal $\eta = 6.05052635$ for conventional SMC is given by PSO. The sampling time is 0.005 s and the simulation lasts for 2400 steps.

The results analysis of the sine wave target is similar to that of the constant value target. For the sliding mode variables in Figure 13, we can see that the USMC-RL renders a shorter settling time than the USMC and that the chattering issue is greatly alleviated in the vicinity of equilibrium. While the chattering amplitude of the USMC is about 0.03, the output of the USMC-RL remains smooth. Figure 14 shows the angle output. We can see that the outputs of both controllers follow the reference signal with some slight lagging. This lagging is caused by the invariant controller of the U-model. Similar lagging is also witnessed in Figure 15. The same overshoot is also shown in Figure 15, which is induced by the chattering of USMC-RL, as shown in Figures 16 and 18, illustrates a periodic pattern of the model outputs. The outputs are elevated when the absolute reference acceleration reaches the peaks.

4.6. DDPG Output Visualization and Interpretation. Visualization is a simple but direct technique to interpret the deep learning model [41]. To justify the effectiveness of the learned policy, part of the output of the actor network is visualized. Figures 18 and 19 illustrate the 1st and 2nd output of DDPG when the state of the pendulum is $[\theta, \dot{\theta}]^T = [0, 0]^T$. The x axis is the angular error, ranging from −2 rad to 2 rad, and the y axis is the velocity error, ranging from −2 rad/s to 2 rad/s. It is clear that both outputs present a similar shape of a valley. When the errors are small, the outputs are small; when the errors grow larger, the outputs are becoming larger. This fits the intuition because larger outputs mean larger control gains, and subsequently, quicker responses and higher accuracy. We think this indicates the correctness of the learned policy. However, we can also see that the outputs following the line $y = x$ are smaller than $y = -x$, which is counterintuitive. If the angular error and velocity error have the same sign, then the error will keep increasing, and it is better to output larger values. We believe this is because, when $y = -x$, the velocity will force the absolute angular error to decrease, which renders a fake stimulation to the model, spurring the model to output larger values when $y = -x$. We believe this is caused by inadequate exploration.

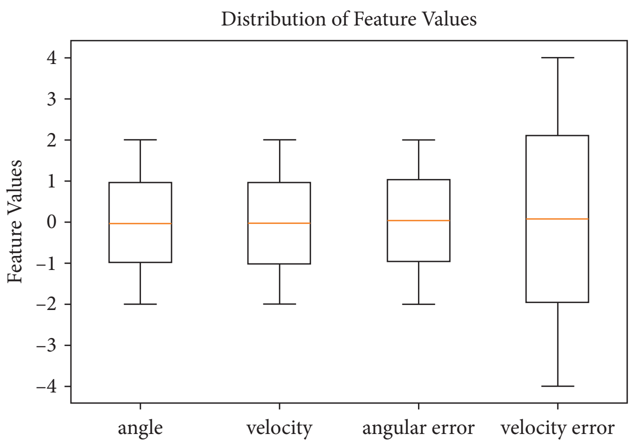

FIGURE 20: Distribution of background data.

4.7. DDPG Interpretation Using SHAP. While [42] visualization renders us a holistic picture of the model output, the information it can provide is only qualitative. To understand the model behaviour in a quantitative manner, we implemented the SHAP method for a model explanation, in terms of both global explanation and local explanation. First, an introduction of SHAP is given, and then, the global and local explanations are presented.

SHAP (SHapley Additive exPlanations) is a model-agnostic, post-hoc explainable artificial intelligence (XAI) method. It explains the final output as the addition of attributions from all inputs. In this way, it can explain the influence of each input on the output, in a positive or negative direction. It borrows ideas from SHapley values [43] to calculate the marginal contribution of each input. To calculate SHapley values, we have to retrieve "background data" through sampling. Similar to the monte-carlo method, the more data points we sample, the more accurate the estimation is. In this paper, the sampling ranges of four input features are set as follows:

$$\begin{aligned} &-2 \le \theta \le 2, \\ &-2 \le \dot{\theta} \le 2, \\ &-2 \le e\_\theta \le 2, \\ &-4 \le e\_\dot{\theta} \le 4. \end{aligned} \tag{29}$$

We took a uniform sample of 5,000 points from this domain, combined them with the respective model outputs, and formed our background data. Figure 20 illustrates the distribution of our background data using a box plot. The mean values of four input features are [−0.01758551, −0.01059006, 0.02874098, 0.05264564], and the corresponding standard deviations are [1.14712378, 1.15875792, 1.15254858, 2.32362302].

(Global explanation). The global explanation focuses on gauging how much influence each input feature has on each

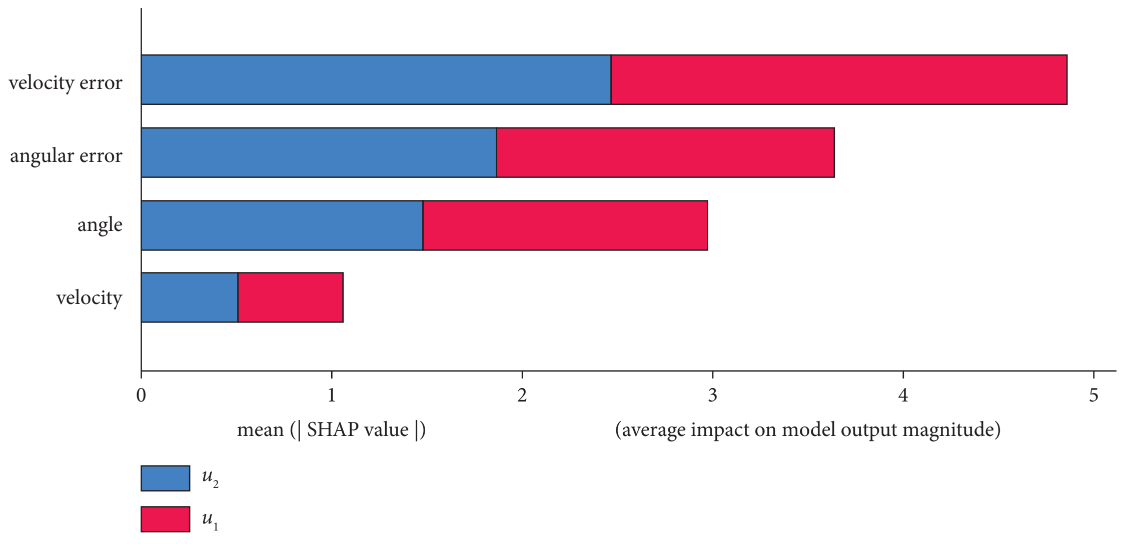

Figure 21: Global explanation of the model.

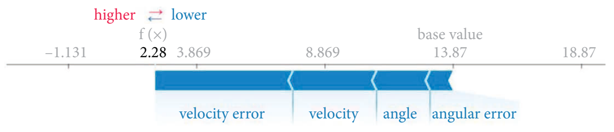

Figure 22: Force plot of u_1 under low error.

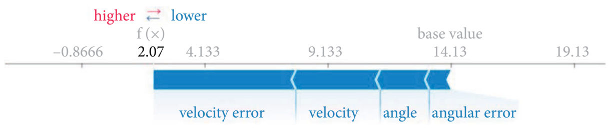

Figure 23: Force plot of u_2 under low error.

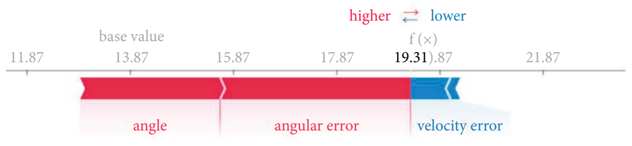

Figure 24: Force plot of u_1 under high error.

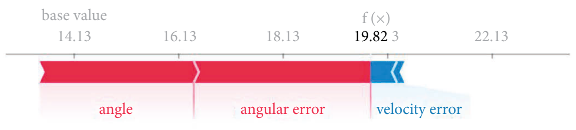

Figure 25: Force plot of u_2 under high error.

of the model outputs. This is carried out by calculating the expectation of the absolute SHapley value of each input feature with respect to the outputs. By using the background data, we have Figure 21, which shows the estimated SHapley value of each input feature with respect to the two model outputs (u_1, u_2). The feature names are listed and depicted in a descending order from top to bottom, according to the SHapley value. In other words, velocity error influences the outputs most, compared with other features. This fits with the intuition because the velocity range is larger than the angle range, and the velocity error accounts for the dynamic process. The second most influential feature is the angular error, and the last two features are angle and velocity. We can see that the first two features are related to the error information, which determines the robustness term of the SMC controller. The last two features are more related to the nominal control term. This result coincides with the rationale of the proposed method because the model outputs account for the robustness term of SMC, and it should take more consideration of the errors.

(Local explanation). The local explanation interprets quantitatively, figuring out why the model outputs certain values compared with the base value (the base value refers to the mean outputs among all the background data). Here, we take two extreme examples to show how our DDPG model can ease the chattering without compromising the settling time and accuracy. Figure 22 and Figure 23 show the force plots of two DDPG outputs with the input features being $[\theta, \dot{\theta}, e\_\theta, e\_\dot{\theta}] = [0, 0, 0, 0]$. We say this input is “under low error” because the errors are 0 and the system is in self-equilibrium. We can see from the figures that all four features are dragging the outputs from base values (13.87 and 14.13) to very low values (2.28 and 2.07). This behaviour fits with common sense since the system is now stable, so DDPG only needs small outputs to reject disturbance. In this way, the chattering issue is eased.

Another extreme is when the errors are large. Consider the initial condition of our simulations, which is $[\theta, \dot{\theta}, e\_\theta, e\_\dot{\theta}] = [2, 0, 2, 0]$. Figures 24 and 25 show the force plots under this situation. We can see that the velocity and velocity error drag the outputs to lower values, while the angle and angular error are dragging it higher. The reasons are twofold. Firstly, the low value of velocity and velocity error suggests that no large values are required. Secondly, both the high value of angle and the high value of angular error require larger outputs to stabilize. In this case, the system can converge quickly.

In summary, the analysis results of the SHAP method show that our DDPG model can make rational decisions; when the errors are large, output large values to converge quickly; when the errors are small, output small values to reduce chattering.

5. Conclusion

An adaptive sliding mode controller based on a deep deterministic policy gradient (DDPG) is proposed and combined with U-model control in this paper, for the tracking control of uncertain nonlinear systems. The proposed methodology successfully integrates the simplicity of the U-model and the robustness of SMC. Besides, the online tuning of DDPG results in lower chattering without compromising settling time or accuracy compared with a conventional sliding mode controller. Simulation on the single-pendulum proves its superiority. We think the combination of RL and SMC complements each other. First, from the controller's point of view, the implementation of RL is an alternative method to transform original controllers into adaptive controllers. Compared with typical adaptive controller design methods, RL can reduce manual efforts and find optimum settings automatically. Second, combining with SMC enables the reservation of some explainability and robustness for RL. On the one hand, while we cannot understand the DDPG output directly, we can have an intuitive understanding by combining it with the formula of SMC. On the other hand, the RL algorithm tends to overfit, and the robustness of SMC can help overcome this drawback. Even when the real environment and dynamics are not completely the same as in training (due to wear, tear, hysteresis, etc.), SMC maintains a certain range of stability, which prevents the RL from failing.

However, compared with the other adaptive methods, there are some challenges existing for the RL-based SMC. First, while decreasing human craftsmanship, the RL algorithm requires a large amount of data for training, which may wear and tear the machines in practice. Many methods can be implemented to increase the efficiency of the data and accelerate convergence, e.g., model-based reinforcement learning and imitation learning to warm up. Second, is the generalization problem. The RL algorithms tend to overfit specific scenarios. How to bridge the gap between simulation and reality, as well as how to transfer the model to another scenario, are open questions.

For future work, researchers may implement maximum entropy reinforcement learning to fully explore the potentially larger action space and avoid converging to a local optimum. Finally, to accommodate real-life problems, this methodology can be extended to multi-input-multi-output (MIMO) systems using multi-agent reinforcement learning (MARL). Last but not least, the implementation of DDPG in this paper is only a simple version. More delicate considerations need to be implemented in order to maximize the potential of reinforcement learning in future research. Also, the DDPG-based method should be compared with other typical adaptive SMC controllers.

Data Availability

The simulation data are generated using Python, specifically sine wave function and random values. The source code can be accessed through https://github.com/AndyRay1998/RL-SMC-U.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

References

[1] Q. M. Zhu and L. Z. Guo, "A pole placement controller for non-linear dynamic plants," *Proceedings of the Institution of Mechanical Engineers - Part I: Journal of Systems & Control Engineering*, vol. 216, no. 6, pp. 467–476, 2002.

[2] Q. Zhu, "Complete model-free sliding mode control (CMFSMC)," *Scientific Reports*, vol. 11, no. 1, p. 22565, 2021.

[3] Q. Zhu, R. Li, and X. Yan, "U-model-based double sliding mode control (UDSM-control) of nonlinear dynamic systems," *International Journal of Systems Science*, vol. 53, no. 6, pp. 1153–1169, 2021.

[4] W. Wei, B. Duan, M. Zuo, and Q. Zhu, "An extended state observer based U-model control of the COVID-19," *ISA Transactions*, vol. 124, pp. 115–123, 2022.

[5] J. Zhang, Y. Li, W. Fei, and X. Wu, "U-model based adaptive neural networks fixed-time backstepping control for uncertain nonlinear system," *Mathematical Problems in Engineering*, vol. 2020, Article ID 8302627, 7 pages, 2020.

[6] X. Geng, Q. Zhu, T. Liu, and J. Na, "U-model based predictive control for nonlinear processes with input delay," *Journal of Process Control*, vol. 75, pp. 156–170, 2019.

[7] F. Xu, Z. Wang, X. Song, and S. Wang, "The Fuzzy PID controller design base on nonlinear U-model," in *Proceedings of the 2018 Chinese Control And Decision Conference (CCDC)*, pp. 5378–5383, Shenyang, China, June 2018.

[8] W. Wei, B. Duan, M. Zuo, and W. Zhang, "Active disturbance rejection control for a piezoelectric nano-positioning system: a U-model approach," *Measurement and Control*, vol. 54, no. 3-4, pp. 506–518, 2021.

[9] J. Slotine and W. Li, *Applied Nonlinear Control*, MIT Press, Cambridge, Massachusetts, United States, 1991.

[10] Y. Shtessel, C. Edwards, L. Fridman, and A. Levant, *Sliding Mode Control and Observation*, Birkhäuser Basel, Basel, Switzerland, 1st edition, 2015.

[11] C. Lei, Q. Zhu, and C. Yang, "U-model-based sliding mode control for manipulator control systems," *International Journal of Cybernetics and Cyber-Physical Systems*, vol. 1, no. 1, Article ID 10048386, 2022.

[12] R. Wang, L. Gao, C. Bai, and H. Sun, "U-Model-Based sliding mode controller design for quadrotor UAV control systems," *Mathematical Problems in Engineering*, vol. 2020, no. 4, 11 pages, Article ID 4343214.

[13] V. Utkin and H. Lee, “Chattering problem in sliding mode control systems,” *IFAC Proceedings Volumes*, vol. 39, p. 5, 2006.

[14] M. R. Soltanpour, S. Zaare, M. Haghgoo, and M. Moattari, “Free-chattering fuzzy sliding mode control of robot manipulators with joints flexibility in presence of matched and mismatched uncertainties in model dynamic and actuators,” *Journal of Intelligent and Robotic Systems*, vol. 100, pp. 47–69, 2020.

[15] K. Hassan, *Nonlinear Systems*, Prentice-Hall, Upper Saddle River, NJ, 2002, https://cds.cern.ch/record/1173048, 3rd ed edition.

[16] K. Lochana and B. K. Royb, “Position control of two-link flexible manipulator using low chattering SMC techniques,” *International Journal of Control Theory and Applications*, vol. 8, no. 3, 2016.

[17] S. Muhammad Amrr and A. S. Alturki, “Robust control design for an active magnetic bearing system using advanced adaptive SMC technique,” *IEEE Access*, vol. 9, no. 2021, pp. 155662–155672, 2021.

[18] Q. Luo, J. Li, and H. Zhang, “Drag coefficient modeling of heterogeneous connected platooning vehicles via BP neural network and PSO algorithm,” *Neurocomputing*, vol. 484, no. 2022, pp. 117–127, 2022.

[19] C. Lei in *Proceedings of the 2021 the 5th International Conference on Information System and Data Mining*Actor-Critic Neural Network Based Finite-time Control for Uncertain Robotic Systems, Silicon Valley USA, May 2021.

[20] M. Gromniak and J. Stenzel, “Deep reinforcement learning for mobile robot navigation,” in *Proceedings of the 2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, pp. 68–73, Nagoya, Japan, July 2019.

[21] K. Ito and F. Matsuno, “A study of reinforcement learning for the robot with many degrees of freedom - acquisition of locomotion patterns for multi-legged robot,” *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 4, pp. 3392–3397, 2002.

[22] D. Liu, X. Yang, D. Wang, and Q. Wei, “Reinforcement-learning-based robust controller design for continuous-time uncertain nonlinear systems subject to input constraints,” *IEEE Transactions on Cybernetics*, vol. 45, no. 7, pp. 1372–1385, 2015.

[23] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3/4, pp. 279–292, 1992.

[24] V. Mnih, K. Kavukcuoglu, D. Silver et al., “Playing atari with deep reinforcement learning,” 2013.

[25] T. Degris, M. White, and S. Richard, “Off-policy actor-critic,” ArXiv abs/1205.4839, 2012.

[26] T. D. Barrett, J. N. Foerster, and A. I. Lvovsky, “Exploratory combinatorial optimization with reinforcement learning,” ArXiv abs/1909.04063, 2020.

[27] Q. Cappart, T. Moisan, L. Rousseau, I. Pr’emont-Schwarz, and A. Augusto Ciré, “Combining reinforcement learning and constraint programming for combinatorial optimization,” ArXiv abs/2006.01610, 2021.

[28] S. Wang, R. Diao, C. Xu, Di Shi, and Z. Wang, “On multi-event Co-calibration of dynamic model parameters using soft actor-critic,” *IEEE Transactions on Power Systems*, vol. 36, no. 1, pp. 521–524, 2021.

[29] T. P. Lillicrap, J. J. Hunt, P. Alexander et al., “Continuous control with deep reinforcement learning,” CoRR abs/1509.02971, 2016.

[30] D. Stephen and W. Zheng, “Twin-delayed DDPG: a deep reinforcement learning technique to model a continuous movement of an intelligent robot agent,” in *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, New York, NY, USA, August 2019.

[31] A. Kumar, N. Paul, and S. N. Omkar, ArXiv abs/1807.05924, Bipedal Walking Robot using Deep Deterministic Policy Gradient, 2018.

[32] Y.-C. Liu and C.-Yu Huang, “DDPG-based adaptive robust tracking control for aerial manipulators with decoupling approach,” *IEEE Transactions on Cybernetics*, vol. 52, no. 8, 2021.

[33] G. Brockman, V. Cheung, L. Pettersson et al., “Openai gym,” arXiv preprint arXiv:1606.01540, 2016.

[34] Q. M. Zhu, D. Y. Zhao, and J. Zhang, “A general U-block model-based design procedure for nonlinear polynomial control systems,” *International Journal of Systems Science*, vol. 47, pp. 3465–3475, 2016.

[35] Q. Zhu, W. Zhang, Na Jing, and B. Sun, “U-model based control design framework for continuous-time systems,” in *Proceedings of the 2019 Chinese Control Conference (CCC)*, pp. 106–111, Guangzhou, China, July 2019.

[36] R. Li, Q. Zhu, J. Kiely, and W. Zhang, “Algorithms for U-Model-Based dynamic inversion (UM-Dynamic inversion) for continuous time control systems,” *Complexity*, vol. 2020, Article ID 3640210, 14 pages, 2020.

[37] D. Silver, L. Guy, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,”vol. 32, pp. I–387–I–395, in *Proceedings of the 31st International Conference on International Conference on Machine Learning*, vol. 32, pp. I–387–I–395, ICML’14, Beijing, China, 2014.

[38] W. Bai, T. Li, and S. Tong, “NN reinforcement learning adaptive control for a class of nonstrict-feedback discrete-time systems,” *IEEE Transactions on Cybernetics*, vol. 50, pp. 4573–4584, 2020.

[39] Z. Qin, X. He, and D. Zhang, “Nonsingular and fast convergent terminal sliding mode control of robotic manipulators,” in *Proceedings of the 30th Chinese Control Conference*, pp. 2606–2611, Yantai, China, July 2011.

[40] 1955 Craig, *Introduction to Robotics: Mechanics & Control*John J. Craig. Addison-Wesley Pub. Co, Menlo Park, Calif, 1986.

[41] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser et al., “Explainable Artificial Intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI,” *Information Fusion*, vol. 58, no. 2020, pp. 82–115, 2020.

[42] S. Lundberg and S Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio et al., Eds., pp. 4765–4774, Curran Associates, Inc, New York, NY, USA, 2017, http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[43] A. E. Roth, *The Shapley Value: Essays in Honor of Lloyd S. Shapley*, Cambridge University Press, Cambridge, United Kingdom, 1988.