biomimetics

Article

A High-Efficient Reinforcement Learning Approach for Dexterous Manipulation

Jianhua Zhang [1], Xuanyi Zhou [2], Jinyu Zhou [2], Shiming Qiu [2], Guoyuan Liang [3], Shibo Cai [2] and Guanjun Bao [2,*]

[1] College of Mechanical Engineering, Beijing University of Science and Technology, Beijing 100083, China; jhzhang@ustb.edu.cn
[2] College of Mechanical Engineering, Zhejiang University of Technology, Hangzhou 310023, China; zhouxuanyi@zjut.edu.cn (X.Z.); 201906040306@zjut.edu.cn (S.Q.)
[3] Guangdong Provincial Key Laboratory of Robotics and Intelligent System, Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China
* Correspondence: gjbao@zjut.edu.cn

Abstract: Robotic hands have the potential to perform complex tasks in unstructured environments owing to their bionic design, inspired by the most agile biological hand. However, the modeling, planning and control of dexterous hands remain unresolved, open challenges, resulting in the simple movements and relatively clumsy motions of current robotic end effectors. This paper proposed a dynamic model based on generative adversarial architecture to learn the state mode of the dexterous hand, reducing the model's prediction error in long spans. An adaptive trajectory planning kernel was also developed to generate High-Value Area Trajectory (HVAT) data according to the control task and dynamic model, with adaptive trajectory adjustment achieved by changing the Levenberg–Marquardt (LM) coefficient and the linear searching coefficient. Furthermore, an improved Soft Actor–Critic (SAC) algorithm is designed by combining maximum entropy value iteration and HVAT value iteration. An experimental platform and simulation program were built to verify the proposed method with two manipulating tasks. The experimental results indicate that the proposed dexterous hand reinforcement learning algorithm has better training efficiency and requires fewer training samples to achieve quite satisfactory learning and control performance.

Keywords: adaptive trajectory planning kernel; dynamic model; reinforcement learning; generative adversarial architecture

Citation: Zhang, J.; Zhou, X.; Zhou, J.; Qiu, S.; Liang, G.; Cai, S.; Bao, G. A High-Efficient Reinforcement Learning Approach for Dexterous Manipulation. *Biomimetics* **2023**, *8*, 264. https://doi.org/10.3390/biomimetics8020264

Academic Editor: Dapeng Yang

Received: 23 April 2023
Revised: 5 June 2023
Accepted: 6 June 2023
Published: 16 June 2023

1. Introduction

Dexterous hands are considered to be the most complex and diverse end effector for robots. Compared to other end effectors, dexterous hands have more flexibility in grasping objects [1], especially irregular objects. However, performing excellent grasping and manipulation with dexterous hands remains a grand challenge.

Excellent grasping performance relies highly on interaction with the environment, where reinforcement learning algorithms have been applied for decades to solved countless sequential challenging problems [2]. In particular, deep reinforcement learning (DRL), a combination of reinforcement learning and deep neural networks, has achieved fruitful results in the field of deep learning [3,4] for robotics, such as robotic dogs, unmanned vehicles and humanoid robots [5]. High-level, domain-specific knowledge for robotic planning could be learned automatically. Moreover, the computational framework for robotic tasks and automation have been developed to facilitate the study and development of manipulation skills for robotic hand.

Transferring difficulties in robotics to reinforcement learning [6] has enabled robots to address previously intractable problems [7], and DRL has provided a powerful means

of representing complex policies in high-dimensional environments [8,9]. A considerable number of scholars have reported their achievements in applications on real-world robots. For example, Tsurumine proposed a deep reinforcement learning network that combines the nature of smooth policy to enhance stability and efficiency for robotic cloth manipulation [10]. A path integration reinforcement learning algorithm was proposed to solve the control problem of tendon-wired robotic actuators without modeling the dexterous hand or environment, inspired by the human brain's muscle synergy to obtain planning and control [11]. However, reinforcement learning in real-world robots is very time-consuming [12]. For instance, a model-free reinforcement learning algorithm applied to a three-finger dexterous hand system took seven hours of training time to complete various control tasks in the real world [13]. This slow training efficiency remains a stumbling obstacle to real-time robotic control. To accelerate the training speed, researchers have proposed various techniques. For example, a hierarchical planning method was proposed to achieve the movement of turning small balls by learning the best strategies for different levels in a decoupled manner [14]. Q-learning was applied to high-level discrete motions, and an improved path integration strategy was used to learn Dynamic Movement Primitives (DMPs) for low-level control. Adding a small amount of artificial demonstration data was found to be effective. In order to improve the efficiency of data use, a DPG-R algorithm was proposed based on the DDPG algorithm and Q-Learning algorithm. The efficiency of learning data samples was improved by separating the update frequency of the network from the environment [15]. Kumar designed a non-grasping five-finger dexterous hand by learning the local linear time-varying model of the dexterous hand and building a local trajectory planning controller for the model [16]. These controllers were first initialized with manual demonstration data. Moreover, the control strategy could be learned only through tactile and dexterous hand proprioceptive feedback without relying on the visual feedback of the object. Nagabandi proposed planning with a deep dynamics model (PDDM), a model-based reinforcement learning algorithm [17]. In contrast to the local linear time-varying model, deep neural networks were used to learn a global model of the system, and the learned model was used to plan motion based on the cross-entropy method (CEM). The algorithm was applied to control the multi-fingered dexterous hand to perform multiple tasks [18,19]. Andrychowicz from the OpenAI company proposed a technique that uses domain randomization to pre-train dexterous hands in the virtual world through reinforcement learning. The pre-trained model was then fine-tuned using transfer learning to operate on a physical robotic hand. To estimate object poses from visual information, multi-layer convolutional neural networks were commonly used [20]. Proximal policy optimization (PPO) was often employed to train dexterous hands in a virtual environment using thousands of different parameters [21]. However, one difficulty with PPO is that it may require a large amount of training data to achieve acceptable results.

To address this issue, Soft Actor–Critic (SAC) was proposed based on the maximum entropy theory. Compared to algorithms such as Deep Deterministic Policy Gradient (DDPG), SAC is more efficient in selecting initial network parameters. Additionally, due to the adoption of maximum entropy, SAC can achieve better control performance for different random seeds. Therefore, applying the SAC algorithm to the dexterous hand system to improve data learning efficiency is a promising approach. Overall, in applying domain randomization, transfer learning, and multi-layer convolutional neural networks, SAC can improve the efficiency of dexterous hand learning in both virtual and real-world environments.

In this paper, a dynamic model of the dexterous hand is proposed based on generative adversarial architecture to learn the dynamic changing principle of the state of the dexterous hand. The contributions of this paper are as follows:

- To address the problem of prediction error explosion over long spans, the state of the dexterous hand is learned through generative adversarial architecture with high prediction accuracy.

- An adaptive trajectory planning method is proposed. Moreover, an adaptive trajectory programming kernel is built to generate High Value Area Trajectory (HVAT) based on the dexterous dynamic model and object. A smooth distance loss function and a U-shaped loss function are designed to calculate the loss value of the dexterous hand system at the reference point.
- A new Actor–Critic-based reinforcement learning algorithm is proposed for the control of the dexterous hand.

This work enables the dexterous hand to explore high-value areas through a certain execution sequence, thereby improving the learning speed and performance of reinforcement learning algorithms.

2. Methodologies

The hybrid reinforcement learning strategy proposed in this paper is shown in Figure 1, which mainly consists of the SAC network, physical model of the dexterous hand and trajectory planning kernel. There are three networks in SAC, including CriticNet, ValueNet and ActorNet. Firstly, the system state obtained by the sensor in the manipulating environment is taken as the input of the SAC algorithm to the $Actor_\phi$ network. Then the output motion of the dexterous hand is generated by trajectory planning. As the dexterous hand executes the motion, the environment state is updated and employed to renew $Actor_\phi$ to generate the subsequent motion.

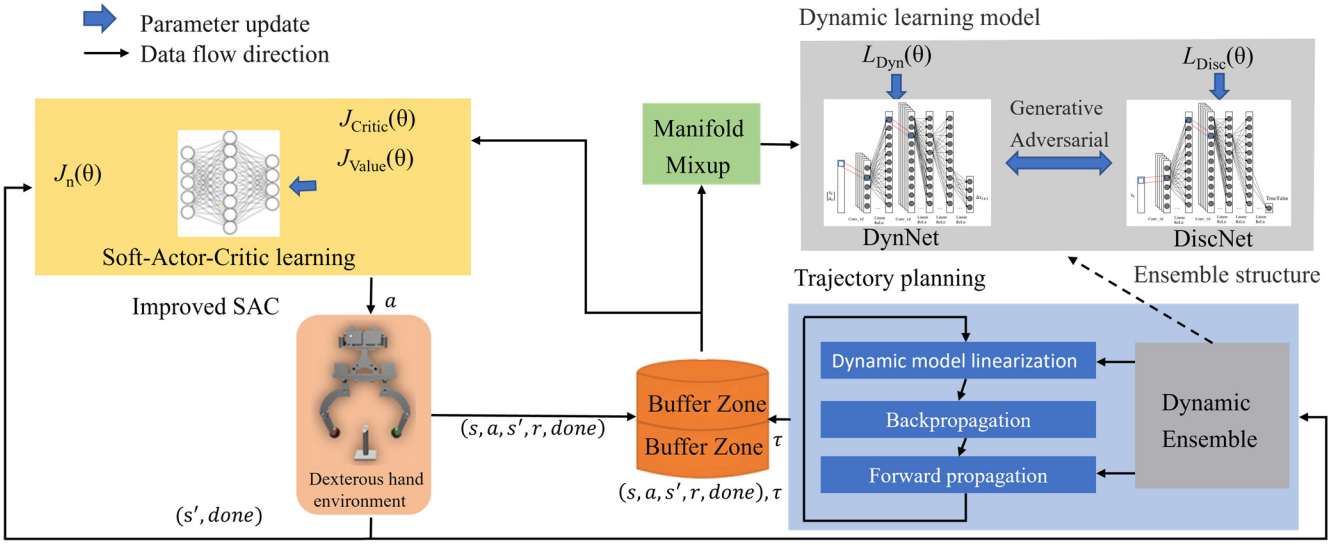


Figure 1. Schematic diagram of the proposed method.

2.1. Generative Adversarial Architecture

In this study, the dexterous manipulation scenario is composed of a robotic hand and its operating target, which is a rotatable object such as a ball or blade. As shown in Figure 2, the state of the system includes variables such as the joint angle, joint velocity, joint torque, fingertip position, fingertip velocity, the position and velocity of the object, etc.

The state vector $s = [s_0, \cdots, s_N]^T$ and output vector o are established for the kinematics and dynamics of the dexterous hand:

$$s' = f(s, a) \tag{1}$$

where a is the input of the system.

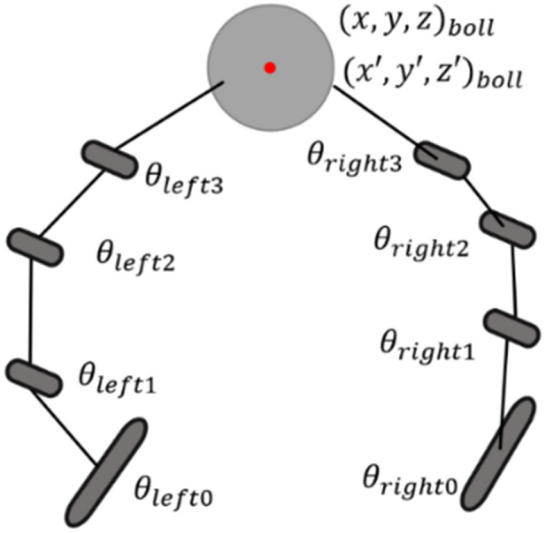


Figure 2. Dynamic model of the dexterous hand.

Traditionally, the dynamic model of the system is established through analyzing the dynamic model. In some cases of collision and slide, the dynamic model is hard to establish accurately. In this paper, a deep neural network is used to learn the dynamic model of the dexterous hand. The state equation of the dynamic model is able to automatically be designed. Another advantage is that the Dyn_{NN} is applied to obtain the changing principle of the system state. Therefore, when selecting the system state s, it only needs to satisfy that the state s satisfies the model of the Markov decision, instead of mapping the system state s into the observed value in traditional control methods.

The dynamic model of the system is obtained through the data-fitting method. Moreover, it can be used for model predictive control. Finally, a dynamic model of the system can be obtained:

$$s_{t+1} = f(s_t, a_t) \approx Dyn_{NN}(s_t, a_t) \tag{2}$$

where Dyn_{NN} represents a neural network model.

An improved generative adversarial nets (GAN) architecture is proposed for building dexterous hand dynamic models. The improved generative adversarial architecture consists of two modules, a generative network module and adversarial network module $Disc_\epsilon$, which is represented by the network parameters ϑ and ϵ, respectively. The structure of Dyn_ϑ is shown in Figure 3.

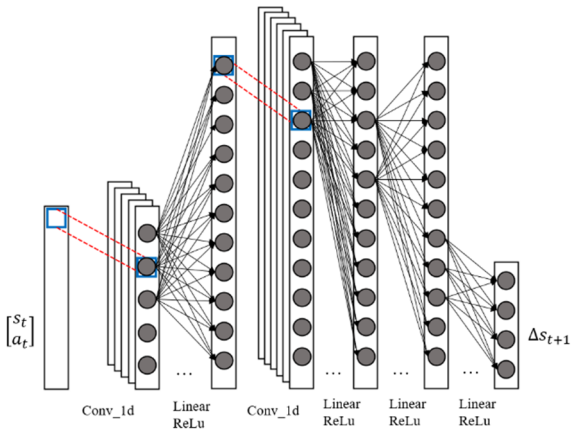


Figure 3. Generative network diagram for the dexterous hand.

The input of Dyn_ϑ is a two-dimensional vector $\begin{bmatrix} s_t \\ a_t \end{bmatrix}$, composed of the state of the dexterous hand at t-moment and the input of t-moment system. The output Δs_{t+1} of Dyn_ϑ is the difference between the predicted state s_{t+1} of the dexterous hand at t-moment and the system state s_t at t-moment.

When predicting the state of the dexterous hand system at the next moment, the current tensor $[s_t, a_t]^T$ first passes through a one-dimensional convolution layer with a kernel size of 1, stride of 1 and out channels of 16. This convolutional layer is mainly used to extract the polarity characteristics of each element in $[s_t, a_t]^T$, and the parameters of this convolutional layer are initialized to a normal distribution with a mean of 0 and variance of $\sqrt[2]{1/8}$. The tensor then passes through a linear and ReLu layer of 256 neurons for nonlinear mapping. Then the tensor continues to pass through a one-dimensional convolutional layer with a kernel size of 1, stride of 1 and out channels of 16. The parameters of this convolutional layer are initialized in the same way as those of the above convolutional layer, and are also used to extract the polar features of each element in the current layer. A basic microarchitecture of Dyn_ϑ is constructed by adjusting the polarity features of the extracted tensor and the number of nonlinear mapping levels. Theoretically, the more times the microarchitecture is reused, the more complex the dynamic model that can be learned by the Dyn_ϑ model. With the above structure, the final tensor passes through three mapping layers composed of linear and ReLu layers, and outputs the predicted value Δs_{t+1}.

The adversarial network module Dyn_ϑ structure is shown in Figure 4. The $Disc_\epsilon$ donates the dexterous hand system state and the predicted dexterous hand system state by True or False, respectively. If the input $Disc_\epsilon$ of the tensor is a true dexterous hand state, then the $Disc_\epsilon$ should be 1, or True. On contrary, if the tensor entered by the $Disc_\epsilon$ is predicted by Dyn_ϑ, the output of the $Disc_\epsilon$ should be 0, or False. In practice, the $Disc_\epsilon$ outputs a value of $[0, 1]$, and the larger the value, the more likely it is that the current input state is a real dexterous hand system state. $Disc_\epsilon$ also has the same structure of one-dimensional convolutional layers and nonlinear mapping as Dyn_ϑ for extracting polarity features.

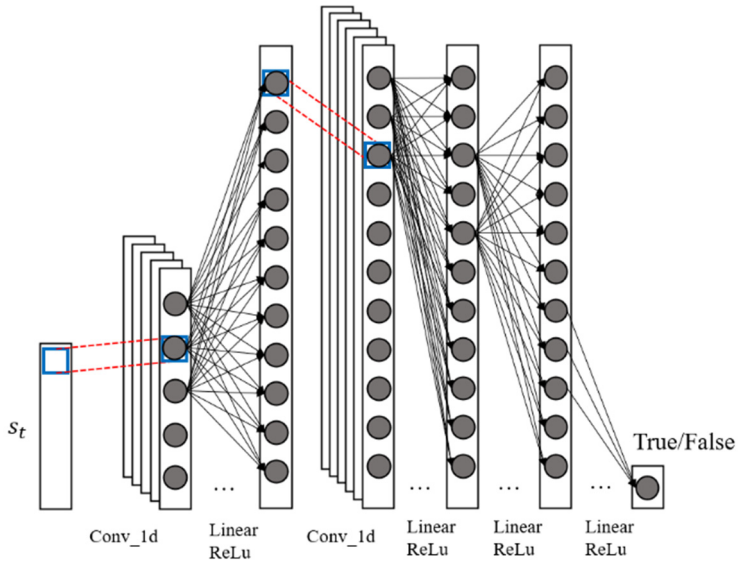


Figure 4. Adversarial network diagram of the dexterous hand.

In the dynamic model of a dexterous hand with a training model, the loss function of Dyn_ϑ is:

$$L_{Dyn}(\vartheta) = E_{s_t, a_t, s_{t+1} \sim Buffer}[\chi \log(1 - Disc_\epsilon(Dyn_\vartheta(s_t, a_t) + s_t)) + (1 - \chi)\|s_{t+1} - s_t - Dyn_\vartheta(s_t, a_t)\|] \quad (3)$$

The loss function of $Disc_\epsilon$ is:

$$L_{Disc}(\epsilon) = -E_{s_t,a_t\sim Buffer}[\log(1 - Disc_\epsilon(s_t)) + \log(Disc_\epsilon(Dyn_\vartheta(s_t, a_t) + s_t))] \quad (4)$$

It can be concluded from the Dyn_ϑ network that the loss function consists of two parts, $\|s_{t+1} - s_t - Dyn_\vartheta(s_t, a_t)\|$ and $log(1 - Disc_\epsilon(Dyn_\vartheta(s_t, a_t) + s_t))$. The former is the prediction performance of the Dyn_ϑ, which reflects the difference of the real states before and after the system under motion control input a_t. The latter can reflect the probability of the true state in the state of next moment $\hat{s}_{t+1} = Dyn_\vartheta(s_t, a_t) + s_t$ predicted by $Disc_\epsilon$ and Dyn_ϑ. For the $Disc_\epsilon$ network, the loss function also consists of two parts, respectively, the $\log(1 - Disc_\epsilon(s_t))$ and $\log(Disc_\epsilon(Dyn_\vartheta(s_t, a_t) + s_t))$. The former reflects the real state of the dexterous hand system for the possibility of True; and the latter reflects by the Dyn_ϑ prediction of the dexterous hand system status of the next moment $\hat{s}_{t+1}$ for the possibility of False. It is used for training $Disc_\epsilon$ by maximizing the loss function.

2.2. Theoretical Analysis of Trajectory Planning Algorithm

For a limited interval $t = 0 : N$, $\mu = \{a_0, a_1, \ldots, a_{N-1}\}$ represents the control input sequence. A loss function $l(s_t, a_t)$ satisfying concave function is used to calculate the system loss value at t-moment. The loss function belongs to a part of the control system. The loss value is calculated according to the target trajectory and the corresponding system trajectory. Then the cumulative loss value of the system in the interval $k = i : N - 1$ can be expressed as:

$$J_i(s, \mu) = \sum_{i}^{N-1} l(s_k, a_k) + l_f(s_N) \quad (5)$$

Afterwards, the optimal control problem can be expressed as finding an optimal trajectory sequence so that the loss accumulation value is a minimum within $t = 0 : N$:

$$\mu^*(s) = arg\min_{\mu} J_0(s, \mu) \quad (6)$$

where μ^* represents the optimal motion sequence.

The loss value function $V(s, i) = \min_{\mu_i} J_i(s, \mu_i)$, represents the cumulative value of system losses in the optimal motion subsequence $\mu_i^* = \left\{a_i^*, a_{i+1}^*, \ldots, a_{N-1}^*\right\}$, and it is worth noting that the size of $V(s, i)$ only depends on the system state s. According to the memorylessness property, $V(s, i)$ can be obtained:

$$V(s, i) = \min_{a_i}[l(s_i, a_i) + V(s_{i+1}, i + 1)] \quad (7)$$

It can also be obtained from $s_{i+1} = f(s_i, a_i)$:

$$V(s, i) = \min_{a_i}[l(s_i, a_i) + V(f(s_i, a_i), i + 1)] \quad (8)$$

given an initial trajectory $\tau_{old} : \{s_0, a_0, \ldots, s_T\}$ at $t = 0 : N$. In order to obtain the optimal sequence of motion, the control contains two parts, dynamic programming and optimal control. The method of dynamic programming is backpropagation and forward propagation. The optimal control under the linear system is discussed below, namely $f(s_t, a_t) = F\begin{bmatrix} s_t \\ a_t \end{bmatrix}$. In nonlinear systems, optimal control is obtained by iteration and local linearization.

Applying backpropagation, the Jacobian matrix and Hessian matrix of value function can be obtained at $t = i$:

$$V_s(i) = Q_s - Q_a Q_{aa}^{-1} Q_{as} \quad (9)$$

$$V_{ss}(i) = Q_{ss} - Q_{sa} Q_{aa}^{-1} Q_{as} \quad (10)$$

Q_t, K_t, k_t, $V_s(t)$ and $V_{ss}(t)$ in $t = N : 0$ can be calculated according to the above equations, and the expression of the motion increment δa_t with respect to the state change δs_t at each time interval of $t = 0 : N$ is obtained. The motion sequence is applied to the forward propagation and the motion τ_{new} is updated.

Applying forward propagation, the expressions of K_i and k_i at $t = i$ are:

$$K_i = -Q_{aa}^{-1}Q_{as} = -\left(l_{aa} + f_a^T(V'_{ss} + \mu I)f_a + V'_s f_{aa}\right)^{-1}\left(l_{as} + f_a^T(V'_{ss} + \mu I)f_s + V'_s f_{as}\right) \tag{11}$$

$$k_i = -Q_{aa}^{-1}Q_a = -\left(l_{aa} + f_a^T(V'_{ss} + \mu I)f_a + V'_s f_{aa}\right)^{-1}\left(l_a + f_a^T V'_s\right) \tag{12}$$

The detail of the backpropagation and forward propagation are in the Supplementary Materials. Finally, in the nonlinear system, the optimal control is calculated by successive iterations, and each iteration includes the above backward propagation and forward propagation.

2.3. Design of Loss Function

The dexterous hand system is a nonlinear system. In order to enhance the stability of the controller by reducing the change gradient of the loss function, a smooth distance function is designed to evaluate the state loss of the dexterous hand system:

$$l_s(s) = \sqrt{e^2 + {\alpha_s}^2} - \alpha_s \tag{13}$$

where α_s represents the smoothness parameter, as shown in Figure 5, and $e = s - s_{goal}$ representants the system error. Choosing different loss function α_s, the linearity will be regulated.

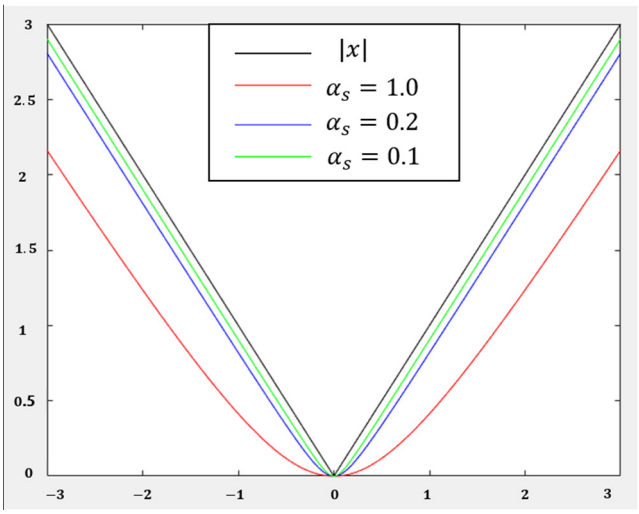


Figure 5. Loss function for different α_s.

In this paper, $\alpha_s = 0.2$ is adopted, and l_s and l_{ss} are defined as follows:

$$l_s = \frac{e}{\sqrt{e^2 + {\alpha_s}^2}} \tag{14}$$

$$l_{ss} = \left(e^2 + {\alpha_s}^2\right)^{-\frac{1}{2}} - e\left(e^2 + {\alpha_s}^2\right)^{-\frac{3}{2}} \tag{15}$$

For the dexterous hand control input, it is necessary to ensure that the system control input has a small loss value within the allowable range. Moreover, after exceeding the

system limit, the system loss value corresponding to the control input increases rapidly. In this paper, a U-shaped loss function is designed as follows:

$$l_a(a) = {\alpha_a}^2 \left(\cosh \frac{a}{\alpha_a} - 1 \right) \tag{16}$$

where α_a is a U-shaped coefficient.

When the control input is greater than 1 or less than -1, the loss value of the system feedback is too large. Figure 6 shows the image of the loss function for different α_a. In this paper, $\alpha_a = 0.3$ is adopted. The loss function, l_a and l_{aa} are expressed as:

$$l_a = \alpha_a \sinh \frac{a}{\alpha_a} \tag{17}$$

$$l_{aa} = \cosh \frac{a}{\alpha_a} \tag{18}$$

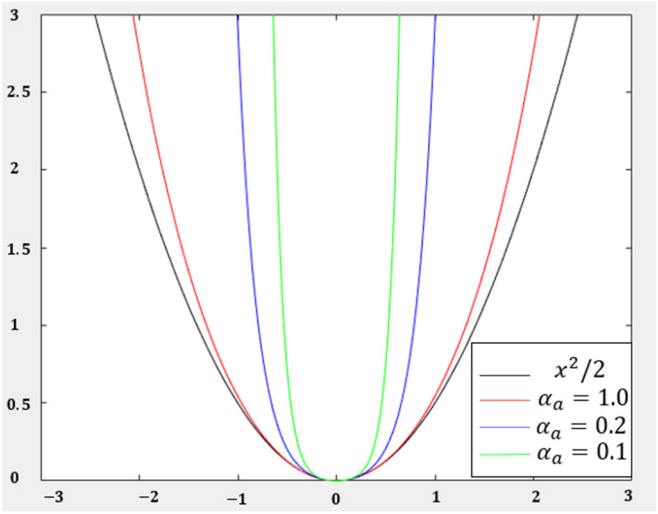


Figure 6. U-shaped loss function for different α_a.

2.4. Kernel Design of Adaptive Trajectory Planning

For nonlinear control systems, in order to obtain the optimal trajectory, there are three steps, as shown in Figure 7. Firstly, the reference point system model should be linearized, then the control feedback matrix K and k are obtained by backpropagation. As a result, the reference trajectory is obtained by forward propagation. An adaptive trajectory planning algorithm is built. By adjusting the linear search coefficient α and LM coefficient μ, the deviation can be controlled between the newly generated trajectory and the original reference trajectory. In the trajectory planning procedure, the existing control strategy is applied to generate an initial reference trajectory according to the initial system state. Then the program linearizes the reference points of the system at each moment of the reference trajectory with respect to Dyn_ϑ to obtain the local linear model, and obtains the Jacobian matrix and Hessian matrix of the loss function according to Equations (14), (15), (17) and (18).

Backpropagation is applied to calculate $V_s(t)$, $V_{ss}(t)$, Q_t, K_t, k_t and $V_s(N) = l_s$, $V_{ss}(N) = l_{ss}$ from time t = N: 0. A linear search is carried out. The linear search coefficient is initialized as $\alpha = 1$, and the new motion sequence $A\{a_0, a_1, \ldots, a_{N-1}\}$ [22] is calculated by K_t, k_t to obtain the new trajectory. If the loss value obtained along the new system trajectory is less than that of the original trajectory, it means that the program accepts this iteration. However, if the loss value obtained along the trajectory is greater than that corresponding to the original system trajectory under the current coefficient, the change

value is reduced by dividing the linear search coefficient α by 1.1. Moreover, the motion sequence is recalculated and the loss value is compared.

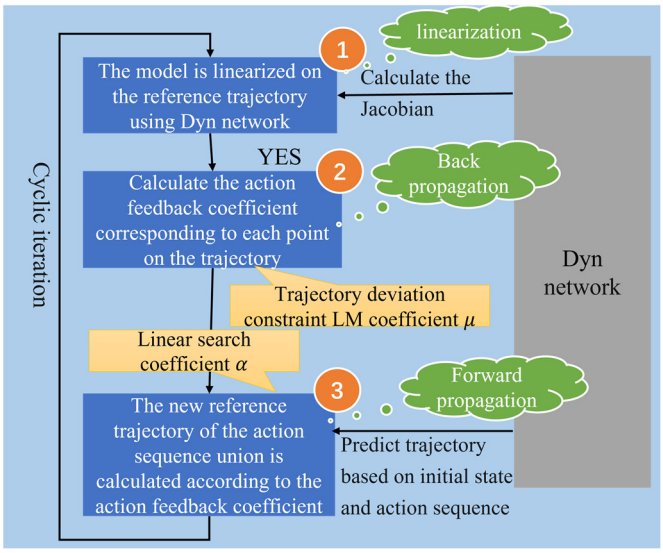


Figure 7. Steps for adaptive trajectory planning.

Then, if the linear search coefficient α has reached the minimum value, the LM coefficient μ is increased and K_t, k_t are recalculated to impose greater constraints on the deviation from the original trajectory. Afterwards, the linear search link is re-activated. The process is repeated until a new trajectory is adopted. By adjusting the linear search coefficient α and LM coefficient μ, the adaptive planning trajectory is generated.

2.5. Improved SAC Algorithm

In this paper, an improved SAC algorithm is proposed to generate the motion of the dexterous hand, including $Value_\psi$ and $TargetValue_{\overline{\psi}}(s_t)$ networks to evaluate the value of motions in the current state and $Actor_\phi$ network. The improved Bellman equation is obtained by incorporating the entropy of the control policy action distribution under the current system state s_t.

$$Q(s_t, a_t) := r(s_t, a_t) + \gamma E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}[V(s_{t+1})] \tag{19}$$

The data cache DataBuffer has an area for HVAT to store HVAT data generated through trajectory planning. In addition, this part of the data is generated by trajectory planning in the fitted dynamic model of the dexterous hand rather than obtained in the actual dexterous hand environment by the control strategy π. The loss functions of $Critic_\theta$ and $Value_\psi$ under HVAT are obtained as follows:

$$J_{Critic}^{HVAT}(\theta) = E_{(s_t, a_t, EUCTN_t) \sim \tau\_Buffer}\left[\frac{conf_t}{2}\left(Critic_\theta(s_t, a_t) - \hat{Q}(s_t, a_t)\right)^2\right] \tag{20}$$

where $conf_t$ is the confidence value predicted by Dyn_ϑ under s_t, a_t, $conf = \frac{1}{EUCTN_t^{Dyn_\vartheta}}$. $EUCTN$ represents the cognitive uncertainty and can be written as:

$$EUCTN_t^{Dyn_\vartheta} \approx \frac{\sum_i^{ES}\left(\Delta s_{t+1_i} - \overline{\Delta s_{t+1}}\right)^2}{ES} \tag{21}$$

where $ES = 3$ is the model number of the ensemble, Δs_{t+1} is the output of the Dyn_ϑ and $\overline{\Delta s_{t+1}}$ is the average predicted value.

The loss function of $Value_\psi$ can be written as:

$$J_{Value}^{HVAT}(\psi) = E_{s_t,a_t\sim HVAT\_Buffer}\Big[\frac{1}{2}\big(Value_\psi(s_t) - \Big(\frac{1}{2}\big(E_{a'_t\sim\pi}[Critic_\theta(s_t,a'_t)] + Critic_\theta(s_t,a_t)\big) - \log\pi(a'_t|s_t)\big)\big)^2\Big] \tag{22}$$

where a_t is sampled in the HVAT area. So the evaluation scope of $Value_\psi$ includes the motion distribution space generated by trajectory planning.

At this point, the loss function gradient for $Critic_\theta$ and $Value_\psi$ is defined as:

$$\hat{\nabla}_\theta J_{Critic}^{HVAT}(\theta) = conf_t\nabla_\theta Critic_\theta(s_t,a_t)\big(Critic_\theta(s_t,a_t) - r(s_t,a_t) - \gamma TargetValue_{\bar{\psi}}(s_{t+1})\Big) \tag{23}$$

$$\hat{\nabla}_\psi J_{Value}^{HVAT}(\psi) = \nabla_\psi Value_\psi(s_t)\big(Value_\psi(s_t) - \Big(\frac{1}{2}\big(E_{a'_t\sim\pi}[Critic_\theta(s_t,a'_t)] + Critic_\theta(s_t,a_t)\big) - \log\pi(a'_t|s_t)\big)\big) \tag{24}$$

Different from the direct generation of the trajectory by the control strategy, this method makes the high-value state region updated preferentially in the value iteration; thus, it indirectly accelerates the learning of the control strategy and avoids the direct influence of the low-value trajectory on the updating strategy of control.

Iteration of Control Policies

To maximize the entropy of actor–critic, the updating rule for the control policy π is defined as:

$$\pi_{new} = arg\min_\phi D_{KL}\Big(\pi(\cdot|s_t)\|\frac{\exp(Q^{\pi_{old}}(s_t,\cdot))}{Z^{\pi_{old}}(s_t)}\Big) \tag{25}$$

where $Z^{\pi_{old}}(s_t)$ is the sum of all $\exp(Q^{\pi_{old}}(s_t,\cdot))$.

Since Critic is used to denote Q, the loss function for π_ϕ can be expressed as follows:

$$J_\pi(\phi) = E_{s_t\sim Buffer}\Big[D_{KL}\Big(\pi_\phi(s_t)\|\frac{\exp(Critic_\theta(s_t,\cdot))}{Z_\theta(s_t)}\Big)\Big] \tag{26}$$

Meanwhile, $Actor_\phi(\epsilon_t;s_t)$ neural network is applied for reparameterization:

$$a_t = Actor_\phi(\epsilon_t;s_t) \tag{27}$$

where ϵ_t is the noise vector sampled from the Gaussian distribution.

Therefore, Equation (24) can be rewritten as:

$$J_\pi(\phi) = E_{s_t\sim Buffer,\epsilon_t\sim N}\big[\log\pi_\phi\big(Actor_\phi(\epsilon_t;s_t)\big|s_t\big) - Critic_\theta\big(s_t,Actor_\phi(\epsilon_t;s_t)\big)\big] \tag{28}$$

It is worth noting that $Actor_\phi$ identifies a unique control strategy π_ϕ. Its gradient can be approximated as:

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi\log\pi_\phi(a_t|s_t) + \big(\nabla_{a_t}\log\pi_\phi(a_t|s_t) - \nabla_{a_t}Critic_\theta(s_t,a_t)\big)\nabla_\phi Actor_\phi(\epsilon_t;s_t) \tag{29}$$

Finally, the parameters of the $Actor_\phi$ network are updated using gradient descent:

$$\phi \leftarrow \phi - \lambda_\pi\hat{\nabla}_\phi J_\pi(\phi) \tag{30}$$

where λ_π is the learning rate of the control strategy π.

3. Experiments

3.1. Model-Fitting Experiments for Dynamic Processes

The dynamic model-fitting experiment of the dexterous hand is performed in a customized two-fingered hand simulation environment, as shown in Figure 8.

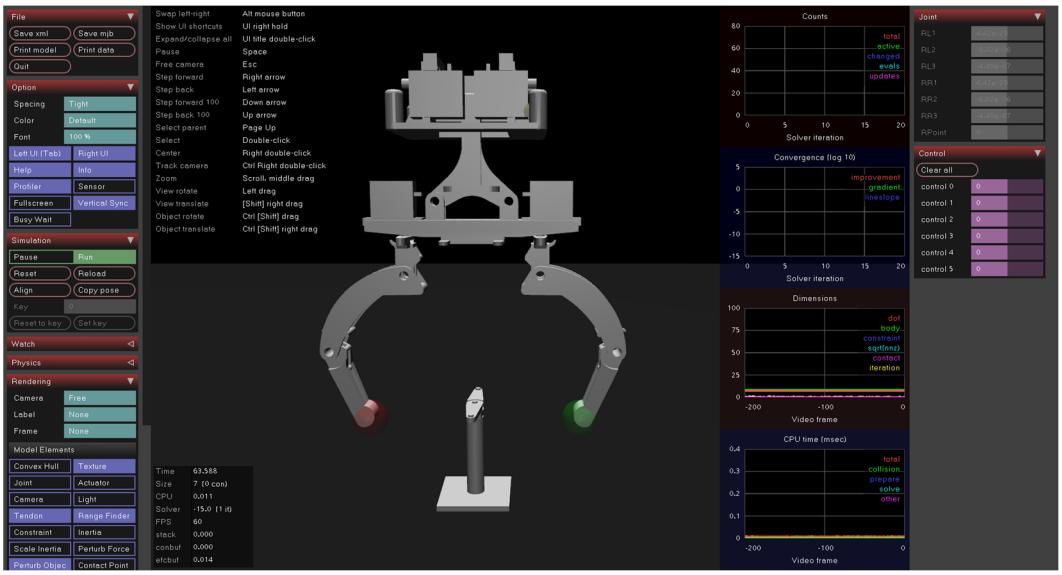


Figure 8. Simulation environment of a two-fingered hand.

The hyperparameters used in this experiment are shown in Tables 1 and 2. The cycle in the experimental simulation environment is 0.02 s, and the time step used in each simulation is 500.

Table 1. Parameters of the improved SAC algorithm.

Parameter	Symbol	Value
Learning rate	lr	0.0003
Hidden layer	$deep$	2
Number of monolayer neurons		256
The entropy coefficient	tmp	10
Discount factor	γ	0.99
Batch size		256

Table 2. Parameters of the dynamic model of the dexterous hand.

Parameter	Symbol	Value
Learning rate	lr	0.0001
Number of monolayer neurons		256
Number of polar layers		2
Adversarial coefficient	χ	0.4
Weight decay		0.0001

We designed two types of prediction errors as performance indexes for the dynamic model of the dexterous hand based on the number of predicted steps; that is, short-span and long-span prediction errors. These indexes are used to analyze the variation principle of prediction errors and the phenomenon of error explosion with an increase in learning times. To do so, we initialized the dexterous hand system status and used the same control

strategy for both types of prediction errors. The short-span prediction error is calculated based on the dynamic model and is obtained by predicting the state change of the dexterous hand system according to the reference trajectory in the next moment. The calculation for the short-span prediction error is:

$$\delta = \text{mean}\left(E_{s_t, a_t \sim \tau} \left[\sum_{i=0}^{T-1} |s_{i+1} - \hat{s}_{i+1}| \right] \right) \tag{31}$$

where $\hat{s}_{i+1} = s_i - Dyn_\vartheta(s_i, a_i)$.

The long-span prediction error is used to predict the complete state trajectory of the dexterous hand system based on its initial state and motion sequence, and the prediction error is calculated by comparing it with the reference state trajectory. Unlike the short-span prediction error, the long-span prediction error can better reflect the model's ability to learn the overall trend of the system state. The calculation for the long-span prediction error is:

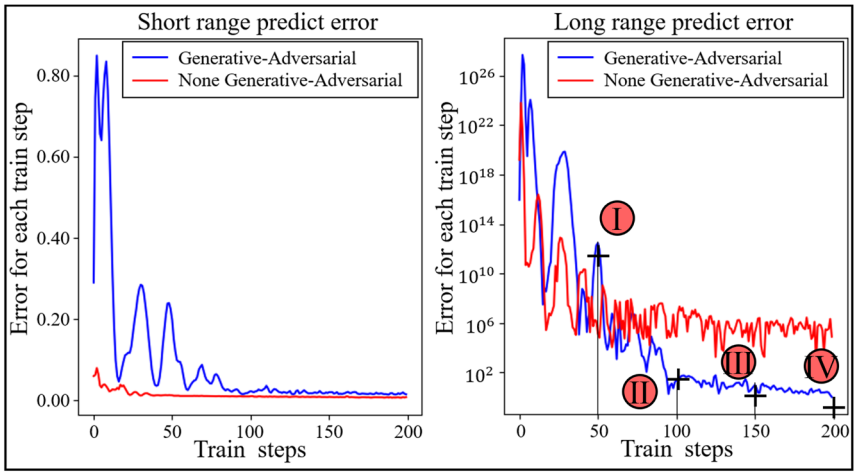
$$\delta = \text{mean}\left(E_{s_0, (s_t, a_t) \sim \tau} \left[\sum_{i=0}^{T-1} |s_{i+1} - \hat{s}_{i+1}| \right] \right) \tag{32}$$

$$\begin{aligned} \hat{s}_{i+1} &= s_i + Dyn_\vartheta(\hat{s}_i, a_i) \\ \hat{s}_i &= \hat{s}_{i-1} + Dyn_\vartheta(\hat{s}_{i-1}, a_{i-1}) \\ \hat{s}_0 &= s_0 \end{aligned} \tag{33}$$

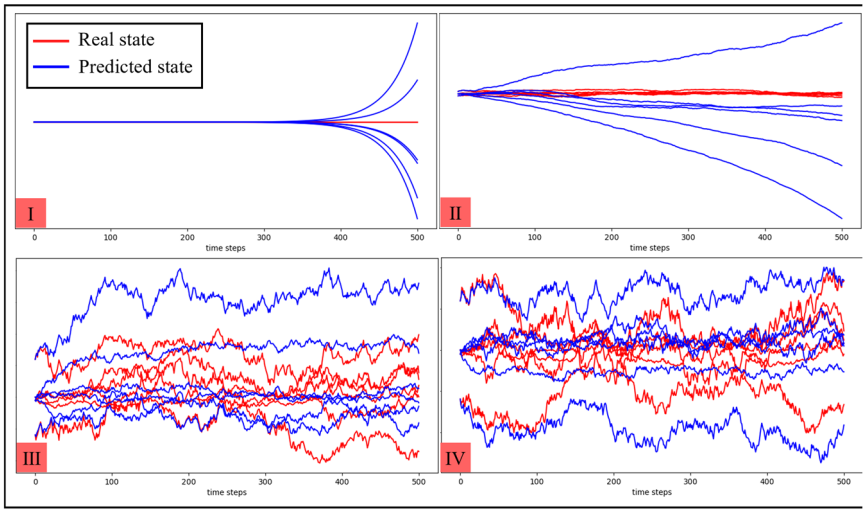
Then, the fixed state of the transfer dataset was utilized to train the dynamic model of the dexterous hand. Initially, the SAC algorithm was employed to generate motion sequence, and the control strategy was updated according to the control task to obtain the system state transfer dataset, which includes 10,000 system state transfer tuples. The dataset was then used to train the two models the same number of times. After each training, the prediction performance was tested five times, and the average value was computed to obtain the model's prediction error. Moreover, after each training, the dynamic model's two different span prediction errors, with and without the generative adversarial architecture, were recorded, and the error trend chart was obtained, as shown in Figure 9. In the figure, the horizontal direction represents an increase in the predicted time, the vertical direction represents the state size of the dexterous hand system and the red curve represents the real system state, while the blue curve represents the system state predicted by the dynamic model.

Figure 9a illustrates the trend of short- and long-span prediction errors in the dynamic model with increasing training times. The red curve indicates that the generative adversarial architecture was adopted, while the blue curve indicates that the architecture was not adopted. The smaller the value in the figure, the higher the prediction accuracy. The lower part of the figure is divided into four subgraphs, respectively, based on the dynamic model of generative adversarial architecture in training, to reach 50(I), 100(II), 150(III), 200 times(IV) of full long-span prediction, forecasting the state change curve and the true state graph.

As can be seen in Figure 9, with the increase in the number of dynamic model trainings, the overall prediction accuracy of the model for the state change of the environment was gradually improved. The model without the generative adversarial architecture was gradually stabilized after 120 trainings on the short-span prediction error, which is slightly better than that using the generative adversarial architecture mode. The short-span prediction error using the generative adversarial architecture model shows an upward trend at the initial stage of training, and eventually falls back to a smaller value, which is due to the fact that the generative adversarial architecture was used for training. Because the loss function of Dyn_ϑ in Equation (3) contains adversarial loss items, the error value in the graph experienced an upward and then a downward trend during the training progress.



(**a**). Predict error of each train step of short and long range



(**b**). Dynamic error of real and predicted state

Figure 9. Evolution of the dynamic model's prediction error.

Regarding long-span prediction performance, the use of the generative adversarial architecture significantly reduced the prediction error compared to the model without the architecture. The prediction error of the former eventually stabilized at the same order of magnitude as the short-span prediction error, while the latter exhibits the phenomenon of error explosion. These results demonstrate that the dynamic model using the generative adversarial architecture can capture the overall trend of the system state change without significant loss in short-span prediction performance. Thus, when the control strategy remains unchanged, the prediction performance of the model for long spans can be significantly improved, thereby enhancing the model's overall prediction accuracy.

3.2. Adaptive Trajectory Planning Experiments of the Dexterous Hand

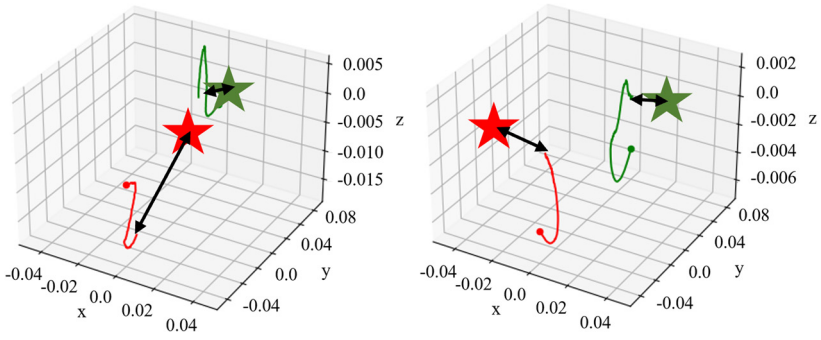
In order to verify the performance of the trajectory planning kernel, experiments with the dexterous hand were conducted. The experimental hyperparameters are listed in Table 3:

Table 3. Kernel hyperparameters of adaptive trajectory planning.

Parameter	Symbol	Value
Initial LM coefficient	μ_0	0.1
Maximum LM coefficient	$MAX\mu$	10
Gain coefficient	Δ^*	1.2
Initial gain coefficient	Δ_0	1.5
Maximum number of iterations		20
Predicted steps to control		15

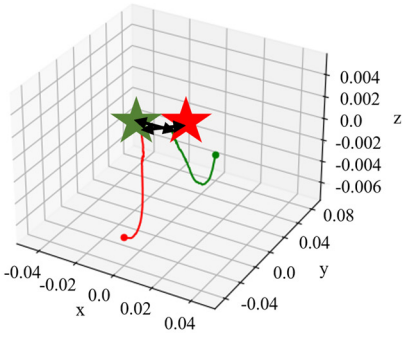
In order to reduce the computational complexity, the nimble fingers space was selected as an important condition. A total of 100 rounds of dexterous hands' state transfer data were collected firstly, with a total number of 100 steps for each round. Then, the dynamic model was fitted. Moreover, the dynamic model and the trajectory planning kernel were used for model predictive control. Then, 10 steps were predicted forward at each time, and the spatial coordinates of the fingertips of dexterous hands were randomly generated as target positions for trajectory planning experiments.

Figure 10 shows the fingertip trajectory of the dexterous hand under a total of 400 times of dynamic model training and under different training times with the dynamic model, where green and red represent finger 1 and finger 2, respectively, the dot represents the initial coordinate of the fingertip, the asterisk is the target coordinate of the fingertip and the curve is the actual trajectory of the corresponding fingertip. Figure 9 selected the dynamic model under different training times using adaptive kernel trajectory planning; in the fingertip trajectory graph, we can see that with the increase in dynamic model training, at the same time and steps, the tip of the finger position with respect to the target movement shows a trend of gradually ascending, and the trajectory planning effect also was improved.

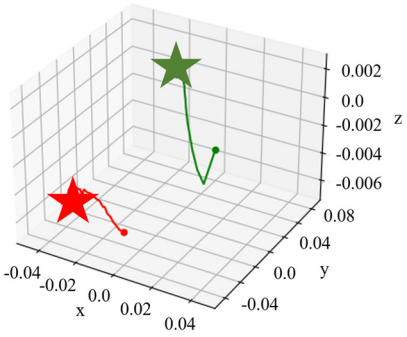


(**a**). 100 training times

(**b**). 100 training times



(**c**). 300 training times



(**d**). 400 training times

Figure 10. The trajectory of the dexterous hand.

According to the experiment, in the early stage of the dynamic model training, due to the large cognitive deviation of the model in the state change of the environment, the dynamic model error occurred in the fingertip trajectory planning near the start point, which led the trajectory planning into local minimum, wrong direction and other problems, resulting in the fingertip's mismatch with the desired position. Although the method of model predictive control can reduce the trajectory error caused by the dynamic model to a certain extent, the performance of the dynamic model had a major impact on the trajectory planning within a certain number of control steps. In the later stages of dynamic model training, a high-value fingertip trajectory, namely HVAT data, could be obtained through adaptive trajectory planning due to the improved accuracy of the model.

3.3. Controlling Experiments of the Dexterous Hand

The HVAT data are obtained via the above method and the dexterous hand control experiment was carried out by combining the SAC algorithm. The pseudo-code of the specific algorithm in the experiment is shown in Table 4.

Table 4. Reinforcement learning algorithm for dexterous hand manipulation.

01: Initialize these network parameters and DataBuffer, such as Dyn_{ϑ}, $Disc_{\epsilon}$, $Actor_{\phi}$, $Critic_{\theta}$, $Value_{\psi}$
02: Collect transfer data for the initial system state $\longrightarrow$ DataBuffer
03: Each episode executes in the iteration loop
04: Sample control targets randomly
05: The number of steps at each time t is executed in the iteration loop
06: Obtain $(s_t, a_t, s_{t+1}, r_t) \longrightarrow$ DataBuffer, with the $Actor_{\phi}$ sampling motion
07: Policy updates in the iteration loop
08: Sample batch data from DataBuffer
09: Update the network parameters of $Critic_{\theta}$, $Value_{\psi}$, $Actor_{\phi}$, $TargetValue_{\bar{\psi}}(s_t)$
10: Dynamic model updates in the iteration loop:
11: Sample batch data from DataBuffer
12: Update Dyn_{ϑ} and $Disc_{\epsilon}$
13: The HVAT sample is taken from the iteration loop
14: Sample the initial data from DataBuffer
15: The optimal trajectory planning is carried out to generate HVAT data $\longrightarrow$ DataBuffer

The experimental platform is shown in Figure 11, which includes a two-fingered hand and its sensing and controlling units, a camera and the ROS program.

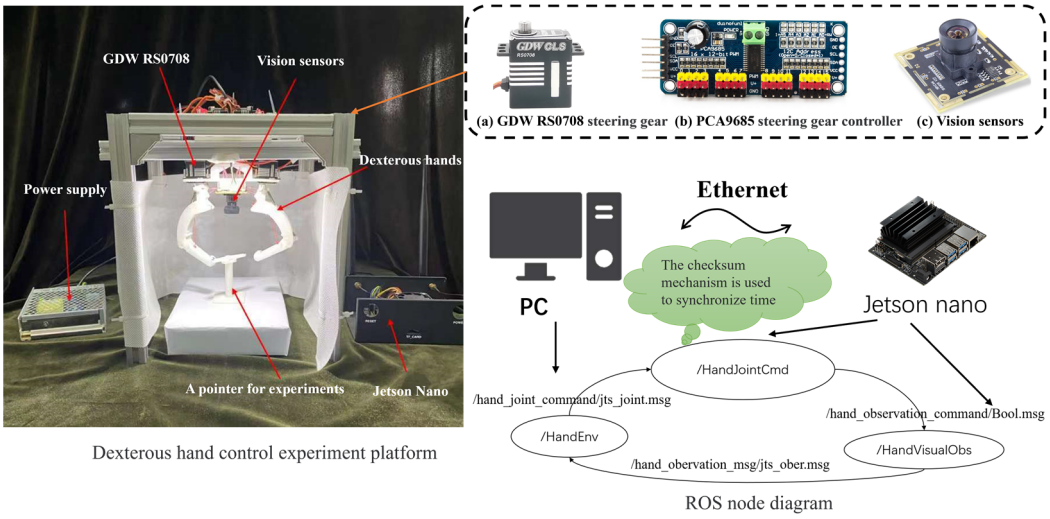


Figure 11. Scheme of the experimental platform.

The experimental results of fingertip controlling are shown in Figure 12, in which the algorithm proposed in this paper (the green one) is compared with the model-free reinforcement learning algorithms SAC [22], PPO [23], DDPG [24], Policy Gradient [24–26] and Actor–Critic [25]. A total of 250 sets of training, which contain 5000 simulation time units, are adopted. From Figure 12, it can be concluded that the exploration ability of the model-free reinforcement learning algorithm is improved. By applying the maximum entropy control strategy, the experimental curve can achieve the maximum reward in the shortest time. Therefore, the overall learning speed of the algorithm is faster than that of the others.

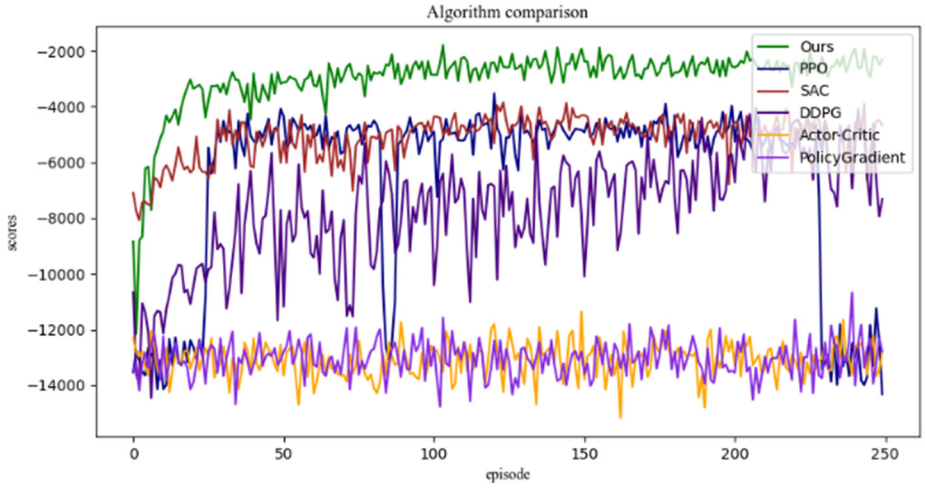


Figure 12. Fingertip controlling experimental results.

For the task of rotating the pointer, the proposed control algorithm achieved the expected motion to a specified angle locally on the experimental platform by learning the relationship between the dexterous hand movements and the state changes of the pointer. Figure 13 shows the captured image by the camera of the experimental platform during the experiment, in which the blue numbers at the top of the image represent the randomly generated desired angle and the white numbers represent the current angle of the pointer. It can be seen from the photos that the dexterous hand can change the angle of the pointer by touching it. Figure 14 shows the curve of the angle changes of the pointer in this experiment.

Meanwhile, comparative experiments were conducted on different reinforcement learning algorithms for the same experimental environment, as shown in Table 5, which indicates the number of sample collection times required by different algorithms to achieve the same reward value. Compared with the others, it can be found that the control algorithm proposed in this paper requires fewer data samples to achieve the same control effect; i.e., the average reward value is in the same range in each round of the control experiment. It should be noted that the reward size of each round is related to the duration of the control steps. The control time steps of each round are set to be 5000.

The experimental results once again proved that the model-based reinforcement learning algorithm proposed in this paper for dexterous hand control can significantly improve the efficiency of training and enhance the learning speed compared to other model-free reinforcement learning algorithms.

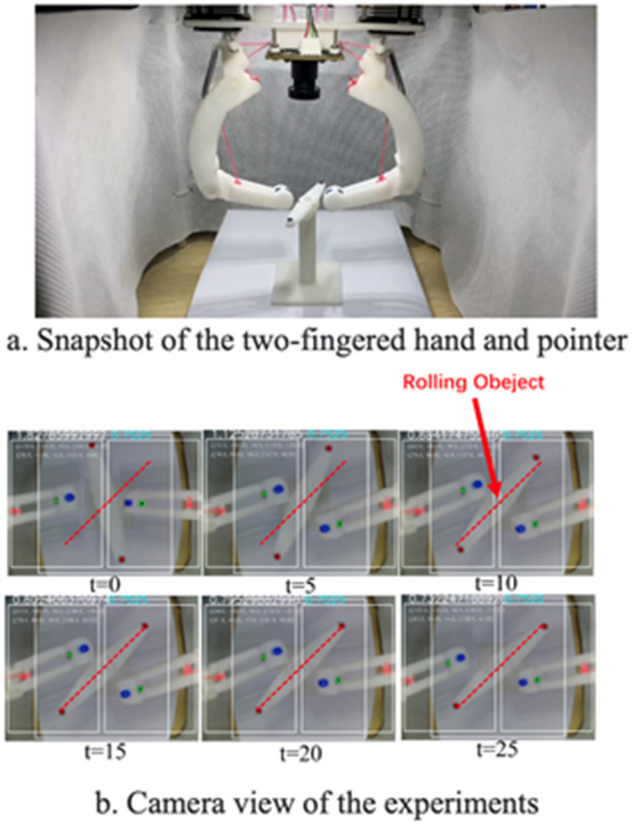


Figure 13. Rotation pointer experiments.

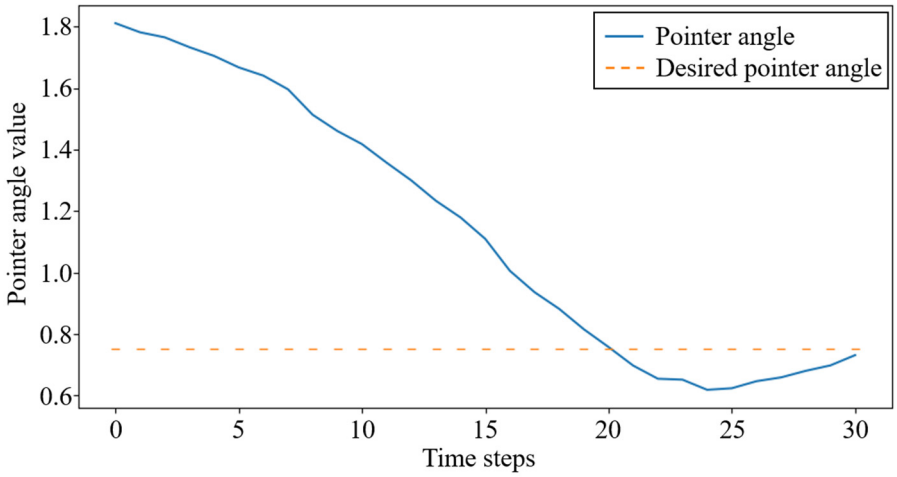


Figure 14. Convergence process of the pointer orientation.

Table 5. Comparison of training sample efficiency of different algorithms.

Methodology	Reward-10,000	Reward-8000	Reward-6000	Reward-4000
Ours	6	10	20	25
SAC	7	12	21	30
PPO	135	151	162	N.A.
DDPG	7	18	63	N.A.

4. Conclusions

Controlling both dexterous hand motion and in-hand manipulation in an unstructured environment with multiple tasks remains a great challenge. To address this issue, one

feasible solution is to allow dexterous hands to interact with the environment and learn to achieve control goals through interaction, similar to that of humans.

In this paper, we proposed a dynamic model of dexterous hands based on generative adversarial architecture, which is capable of learning the dynamic change principle of the dexterous hand system state. To solve the problem of prediction error explosion over long spans, the generative adversarial architecture was employed to learn the entire change rule of the dexterous hand system state while maintaining local prediction accuracy. Then, an iterative optimal control theory of trajectory planning was proposed. An adaptive trajectory planning kernel was built based on this theory to generate the HVAT trajectory according to the dynamic model of the dexterous hand. The kernel can adjust the linear search coefficient α and LM coefficient μ to constrain the change of trajectory, so as to achieve the purpose of self-adaptation. Subsequently, a U-shaped loss function was designed to calculate the loss value of the dexterous hand system at the reference point. Finally, the SAC algorithm was improved and a new Actor–Critic-based reinforcement learning algorithm was proposed for the end-to-end control of dexterous hands, including two critic networks for evaluating the value of motions in the current state. A $Value_{\psi}$ and $TargetValue_{\_\psi}(s_t)$ network was used to assess the value of the current state, and an $Actor_{\phi}$ network was used to generate motions. For random data in the cache, the algorithm adopts the update strategy of the maximum entropy framework, while for HVAT data in the cache, the algorithm adopted the hybrid update strategy.

The proposed algorithm can be used to enable dexterous hands to explore in high-value areas through a certain execution sequence, thereby improving the learning speed and performance of reinforcement learning algorithms. The dexterous hand was enabled to explore high-value areas through a specific execution sequence, thereby improving learning speed and performance. The above contributions provide a robust framework for end-to-end control of dexterous hands in unstructured environments with multitasking capabilities.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/biomimetics8020264/s1. S1: Backpropagation; S2: Forward propagation.

Author Contributions: Conceptualization, J.Z. (Jianhua Zhang) and G.B.; methodology, J.Z. (Jinyu Zhou); software, X.Z.; validation, J.Z. (Jinyu Zhou) and X.Z.; formal analysis, G.L.; data curation, S.C.; writing—original draft preparation, X.Z. and S.Q.; writing—review and editing, G.L. All authors have read and agreed to the published version of the manuscript.

Funding: The research was financially supported by the Joint Fund of the National Natural Science Foundation of China with Shenzhen City (Grant No. U2013212), the Key Research and Development Program of Zhejiang (Grant No. 2021C04015) and the Natural Science Foundation of Zhejiang (Grant No. LZ23E050005).

Data Availability Statement: There are no data to be shared.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Gao, G.; Gorjup, G.; Yu, R.B.; Jarvis, P.; Liarokapis, M. Modular, Accessible, Sensorized Objects for Evaluating the Grasping and Manipulation Capabilities of Grippers and Hands. *IEEE Robot. Autom. Lett.* **2020**, *5*, 6105–6112. [CrossRef]
2. Nguyen, T.T.; Nguyen, N.D.; Nahavandi, S. Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications. *IEEE Trans. Cybern.* **2020**, *50*, 3826–3839. [CrossRef] [PubMed]
3. Arulkumaran, K.; Deisenroth, M.P.; Brundage, M.; Bharath, A.A. Deep reinforcement learning: A brief survey. *IEEE Signal Process. Mag.* **2017**, *34*, 26–38. [CrossRef]
4. Li, X.; Serlin, Z.; Yang, G.; Belta, C. A formal methods approach to interpretable reinforcement learning for robotic planning. *Sci. Robot.* **2019**, *4*, eaay6276. [CrossRef] [PubMed]
5. Kormushev, P.; Calinon, S.; Caldwell, D.G. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics* **2013**, *2*, 122–148. [CrossRef]

6. Iwata, T.; Shibuya, T. Adaptive modular reinforcement learning for robot controlled in multiple environments. *IEEE Access* **2021**, *9*, 103032–103043. [CrossRef]
7. Wiedemann, T.; Vlaicu, C.; Josifovski, J.; Viseras, A. Robotic Information Gathering With Reinforcement Learning Assisted by Domain Knowledge: An Application to Gas Source Localization. *IEEE Access* **2021**, *9*, 13159–13172. [CrossRef]
8. Kober, J.; Bagnell, J.A.; Peters, J. Reinforcement learning in robotics: A survey. *Int. J. Robot. Res.* **2013**, *32*, 1238–1274. [CrossRef]
9. Kiran, B.R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A.A.; Yogamani, S.; Perez, P. Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 4909–4926. [CrossRef]
10. Tsurumine, Y.; Cui, Y.; Uchibe, E.; Matsubara, T. Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation. *Robot. Auton. Syst.* **2019**, *112*, 72–83. [CrossRef]
11. Rombokas, E.; Malhotra, M.; Theodorou, E.A.; Todorov, E.; Matsuoka, Y. Reinforcement Learning and Synergistic Control of the ACT Hand. *IEEE-Asme Trans. Mechatron.* **2013**, *18*, 569–577. [CrossRef]
12. Rothmann, M.; Porrmann, M. A Survey of Domain-Specific Architectures for Reinforcement Learning. *IEEE Access* **2022**, *10*, 13753–13767. [CrossRef]
13. Zhu, H.; Gupta, A.; Rajeswaran, A.; Levine, S.; Kumar, V. Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019.
14. Andrychowicz, M.; Baker, B.; Chociej, M.; Jozefowicz, R.; McGrew, B.; Pachocki, J.; Petron, A.; Plappert, M.; Powell, G.; Ray, A.; et al. Learning dexterous in-hand manipulation. *Int. J. Robot. Res.* **2020**, *39*, 3–20. [CrossRef]
15. Gao, Y.; Ma, S.W.; Liu, J.J.; Xiu, X.C. Fusion-UDCGAN: Multifocus Image Fusion via a U-Type Densely Connected Generation Adversarial Network. *IEEE Trans. Instrum. Meas.* **2022**, *71*, 1–13. [CrossRef]
16. Jain, D.; Li, A.; Singhal, S.; Rajeswaran, A.; Kumar, V.; Todorov, E. Learning Deep Visuomotor Policies for Dexterous Hand Manipulation. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Montreal, CA, Canada, 20–24 May 2019; pp. 3636–3643.
17. Fu, Q.; Santello, M. Chapter 3—Sensorimotor Learning of Dexterous Manipulation. In *Human Inspired Dexterity in Robotic Manipulation*; Watanabe, T., Harada, K., Tada, M., Eds.; Academic Press: Cambridge, MA, USA, 2018; pp. 27–52.
18. Kim, C.I.; Kim, M.; Jung, S.; Hwang, E. Simplified Frechet Distance for Generative Adversarial Nets. *Sensors* **2020**, *20*, 1548. [CrossRef] [PubMed]
19. Luo, W.J.; Wang, P.; Wang, J.H.; An, W. The research process of generative adversarial networks. In Proceedings of the International Seminar on Computer Science and Engineering Technology (SCSET), Shanghai, China, 17–18 December 2018.
20. Abdelgader, A.A.; Viriri, S. Deep Residual Learning for Human Identification Based on Facial Landmarks. In Proceedings of the 15th International Work-Conference on Artificial Neural Networks (IWANN), Gran Canaria, Spain, 12–14 June 2019; pp. 61–72.
21. Xu, Z.; Zhou, X.; Bai, X.; Li, C.; Chen, J.; Ni, Y. Attacking asymmetric cryptosystem based on phase truncated Fourier fransform by deep learning. *Acta Phys. Sin.* **2021**, *70*, 144202. [CrossRef]
22. Dolanc, G.; Strmčnik, S. Design of a nonlinear controller based on a piecewise-linear Hammerstein model. *Syst. Control Lett.* **2008**, *57*, 332–339. [CrossRef]
23. Hamalainen, P.; Babadi, A.; Xiaoxiao, M.; Lehtinen, J. PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation. In Proceedings of the 2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP), Espoo, Finland, 21–24 September 2020; p. 6. [CrossRef]
24. Bernat, J.; Apanasiewicz, D. Model Free DEAP Controller Learned by Reinforcement Learning DDPG Algorithm. In Proceedings of the 2020 IEEE Congreso Bienal de Argentina (ARGENCON), Resistencia, Argentina, 1–4 December 2020; p. 6. [CrossRef]
25. de Jesus, J.C.; Kich, V.A.; Kolling, A.H.; Grando, R.B.; Cuadros, M.A.d.S.L.; Gamarra, D.F.T. Soft Actor-Critic for Navigation of Mobile Robots. *J. Intell. Robot. Syst.* **2021**, *102*, 31. [CrossRef]
26. Ju-Seung, B.; Byungmoon, K.; Huamin, W. Proximal Policy Gradient: PPO with Policy Gradient. *arXiv* **2020**, arXiv:2010.09933.