



Article

A Proposed Priority Pushing and Grasping Strategy Based on an Improved Actor-Critic Algorithm

Tianya You ¹, Hao Wu ^{1,*}, Xiangrong Xu ^{1,*}, Petar B. Petrovic ² and Aleksandar Rodić ³

¹ Faculty of Mechanical Engineering, Anhui University of Technology, Ma'anshan 243000, China; ytyjla@163.com

² Faculty of Mechanical Engineering, University of Belgrade, Kraljice Marije 16, 11120 Belgrade, Serbia; pbpetrovic@mas.bg.ac.rs

³ Institute Mihajlo Pupin, University of Belgrade, Volgina 15, 11060 Belgrade, Serbia; aleksandar.rodic@pupin.rs

* Correspondence: hao.wu@ahut.edu.cn (H.W.); xuxr@ahut.edu.cn (X.X.)

Abstract: The most basic and primary skills of a robot are pushing and grasping. In cluttered scenes, push to make room for arms and fingers to grasp objects. We propose a modified Actor-Critic (A-C) framework for deep reinforcement learning, Cross-entropy Softmax A-C (CSAC), and use the Prioritized Experience Replay (PER) based on the theoretical foundation and main methods of deep reinforcement learning, combining the advantages of algorithms based on value functions and policy gradients. The grasping model is trained using self-supervised learning to achieve end-to-end mapping from image to propulsion and grasping action. A vision module and an action module have been created out of the entire algorithm framework. The prioritized experience replay is improved to further improve the CSAC-PER algorithm for model sample diversity and robot exploration performance during robot grasping training. The experience replay buffer is dynamically sampled using the prior beta distribution and the dynamic sampling algorithm based on the beta distribution (CSAC- β) is proposed based on the CSAC algorithm. Despite its low initial efficiency, the experimental simulation results show that the CSAC- β algorithm eventually achieves good results and has a higher grasping success rate (90%).

Keywords: deep reinforcement learning; FCN; beta distribution; robotic manipulation



Citation: You, T.; Wu, H.; Xu, X.; Petrovic, P.B.; Rodić, A. A Proposed Priority Pushing and Grasping Strategy Based on an Improved Actor-Critic Algorithm. *Electronics* **2022**, *11*, 2065. <https://doi.org/10.3390/electronics11132065>

Academic Editor: Cecilio Angulo

Received: 29 May 2022

Accepted: 26 June 2022

Published: 30 June 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Pushing and grasping are basic operations of a robot and are the foundation and key to the robot's ability to perform various tasks, according to the study of robot manipulation skills. The robot's decision-making and adaptive capabilities have been put to the test as it has to deal with unstructured grasping environments and unknown objects. Deep learning is not only a process of learning relationships between multiple variables, but it is also a process of learning the knowledge that controls the relationships and the knowledge that understands them [1]. The grasping problem is transformed into a target detection problem thanks to deep learning, allowing the mapping from image to grasping action. However, there are still issues such as poor recognition and large localization errors when dealing with complex environments with a wide variety of objects. Furthermore, deep learning-based robotic grasping methods rely on large-scale annotated data, which can be labor-intensive and time-consuming to collect, tag, and build. Through continuous interaction with the environment, reinforcement learning methods learn optimal strategies. The method uses a predetermined reward as a feedback signal instead of manual annotation, combining a deep reinforcement learning algorithm with deep learning. Deep reinforcement learning uses deep neural networks' powerful feature extraction capabilities to make reinforcement learning methods more perceptive and expressive, as well as to handle problems in high-dimensional state and action spaces.

Rather than using previously defined heuristics or hard-coded targets to push actions, Zeng et al. propose using model-free deep reinforcement learning to discover and learn from experience the synergy between pushing and grasping strategies for sequential operations [2]. They employ the traditional DQN (Q-learning) algorithm, which is based on value functions [3], but the value function-based approach struggles to deal with the large action space, particularly in the case of continuous actions. Other issues include low sample utilization, the poor synergy between pushing and grasping actions, and the instability of the value obtained from training. Experience replay and target network freezing were used to solve the problem in the original Actor-Critic framework of deep reinforcement learning algorithms, which learns both Q-functions and policies [4,5]. On the other hand, this approach is unstable and has issues running large-scale reinforcement learning tasks with hyperparameters. Even though improvements to this network could constrain the Q function to be a convex function of the action, making it easier to back-propagate losses, the action convex-valued function is a poor fit, and the Q function is not convex in the input.

Calculating the Temporal Difference Target (TD-Target) is a complex problem that involves optimizing the continuous action space to select the optimal action rather than just the most worthwhile action. The cross-entropy method (CEM) algorithm is a parameter perturbation-based search algorithm [6]. Give the parameter space v some reasonable perturbations, then use cross-entropy to guide the updating so that the direction of the perturbations converges to the direction of the target optimization and is moderately robust to local optimality for low-dimensional problems. The cross-entropy algorithm from the evolutionary strategy optimization algorithm is integrated into our algorithmic framework to maintain a distribution of possible optimal solutions and further select the optimal action in the continuous space.

The introduction of softmax in the discrete action space has been shown to control the gap between the value function and the optimal value [7]. This is crucial when using softmax in continuous action space because errors can be kept within reasonable intervals. The softmax operation is used to smooth the deep reinforcement learning algorithm, reducing the iterative error accumulation problem caused by error transmission along with the policy network and ensuring the algorithm's performance in terms of convergence and stability. The estimation errors caused by their respective estimates are partially balanced, the overestimation bias is reduced, and the evaluation network and objective function calculation strategy gradient algorithm is optimized.

At this stage, experience replay is an indispensable technique for improving sampling efficiency and breaking the correlation of data when training while exploring. The number of replays of historical data is increased by the experience replay mechanism, reducing resource waste and allowing reinforcement learning algorithms to converge faster and better [8–10]. However, existing algorithms in the experience replay lack data filtering, making data training inefficient and algorithm convergence slow. Adding a new experience replay to an existing experience replay, where the data is sorted according to the level of Temporal Difference error, and uniform sampling is replaced with priority sampling, improves the experience replay. Dynamic experience sampling based on beta distribution is proposed based on preferential experience sampling as the number of training steps increases to increase the diversity of the samples sampled and to allow the robot to further explore the environment; though training efficiency is sacrificed to some extent, and satisfactory results are eventually achieved.

To improve the learning efficiency and success rate of robots when operating on target objects in complex environments, this paper was written. Our algorithm is based on the A-C algorithm and adds the following features:

1. To calculate the TD-Target, use the CME optimization algorithm to select the best action in continuous space.
2. Softmax operations are used to improve the temporal differencing method in the policy gradient algorithm.

3. Improving the experience replay structure by sampling with a priority sampling strategy, which is then used to propose a dynamic sampling approach.

2. Related Work

2.1. Deep Learning for Robotic Grasping

Grasping objects is a basic yet challenging task in robotics. Thanks to the development of deep learning, methods represented by convolutional neural networks can learn and mine feature representations from large amounts of annotated data that are superior to those designed manually; the robot is trained on the grasping dataset, performs pose estimation and grasp estimation on the grasping object, and learns the object grasping frame position to realize the grasping of the object. Lenz et al. proposed a search-window approach to grasp detection, innovatively applying deep learning algorithms to robotic grasping [11]. They created a two-stage cascaded neural network that received RGB-D images as input. Multiple candidate grasping frames were generated on the object's surface in the first stage, and those with a higher probability of successful grasping were initially evaluated and screened. Redmon et al. transformed the grasping problem into a regression problem and proposed a real-time robot grasping detection method based on convolutional neural networks [12]. Without the use of a sliding window or candidate network, single-stage regression is performed directly on the graspable bounding box. The sliding window method of identifying candidate rectangle locations necessitates a time-consuming traversal search to obtain the optimal solution. This method, on the other hand, does not necessitate traversal of all possible grasping locations and has a high accuracy rate while greatly increasing detection speed. Jeffrey Mahler et al. at the University of California, Berkeley, built a grasping dataset called Dex-Net2.0 and trained a grasping quality assessment convolutional neural network model (GQ-CNN), which uses a point cloud with edge detection as input to evaluate the effectiveness of grasping at each location [13]. It should be noted that the method primarily evaluates grasping without learning how the controller grasps the object, which is more difficult to train using large-scale datasets and does not account for camera noise, mechanical deformation, and other real-world problems. Chu et al. proposed a multi-object, multi-grasp grasp detection framework that can process RGB-D images of one as well as multiple unknown objects [14]. The entire framework is improved based on Faster RCNN [15], which changes the category of objects to grasp the rotation angle of rectangles and fails to recognize the object category, making accurate grasping of the desired objects difficult. Mingshuo Han et al. addressed the problem of grasping and locating densely stacked objects in a warehouse logistics scenario by using suction cups for the grasping task and proposed a two-stage suction point detection algorithm [16]. However, the accuracy of grasping point detection is low for objects with large differences in geometric features and that are prone to deformation. Fei-Fei Li's team at Stanford University collaborated to propose a novel network called DenseFusion, which uses a color map and a depth map together as input to directly estimate the 6-degree-of-freedom pose of an object for grasping pose determination [17]. This method alters the traditional method of fusing RGB-D features, and this pixel-level fusion method is capable of doing so, which is critical for dealing with heavy occlusion cases.

2.2. Reinforcement Learning for Robotic Grasping

Deep reinforcement learning combines deep learning's feature extraction capability with reinforcement learning's decision-making capability, allowing traditional reinforcement learning methods to solve problems in high-dimensional state and action spaces. Reinforcement learning methods use predetermined rewards as feedback signals, without needing constant interaction with the environment by human annotation, to learn the optimal strategy. Lerrel Pinto and Abhinav Gupta of Cameron University proposed an adversarial grasp learning method that trains a convolutional neural network to directly predict the optimal grasp action of a robot for a given image region by controlling two

robots to learn from each other [18]. The network showed good generalization for unseen objects, achieving a success rate of 66% when tested. Sergey Levine et al. built a learning-based robot hand-eye coordination system with continuous-servo control of the robot to continuously predict the optimal path and update the robot's motion commands for successful grasping, eventually achieving a grasping accuracy of about 80% [19]. Dmitry Kalashnikov et al. combined a massively distributed optimization method with deep reinforcement learning to propose a scalable self-supervised grasping learning method QT-Opt, which achieved a 96% success rate for grasping unknown objects after 580,000 steps of robot training [20]. To reduce training time for deep reinforcement learning algorithms that require a large number of training steps, multiple robots are used to train on the grasping task simultaneously. Quillen investigated the effect of offline strategies on the learning efficiency of deep reinforcement learning algorithms in learning robot grasping strategies and found that a combined approach based on Monte Carlo return estimation and offline correction had higher learning efficiency through simulation experiments [21]. However, the network is unstable and highly sensitive to hyperparameters. Breyel et al. investigated the effects of reward functions and pre-training strategies, among others, on deep reinforcement learning in terms of learning speed and grasping success rates for grasping tasks [22]. However, the algorithm's migration and generalization performance from the simulation environment to the real environment are poor. Clavera et al. addressed the problem of successive reward functions trapping deep reinforcement learning to strategies in local optima by incorporating task prior knowledge into the state space and the reward function to speed up training convergence, and they also proposed a reward-guided algorithm to reduce training time [23].

Haarnoja et al. proposed a maximum entropy strategy for soft Q-learning training applied to realistic robot manipulation to address the problem that realistic physical robots are limited by the time to interact with their environment during training [24]. Soft Q learning can greatly improve the efficiency of training from scratch by combining existing skills to build new strategies, and it can limit the optimality of the resulting strategies based on the differences between the composed strategies, resulting in improved stability and convergence. Singh et al. enable robots to learn from a small number of examples of successful outcomes, eliminating the need for manually developed reward specifications and eliminating the need to manually design rewards as an efficient and practical way to learn skills [25]. However, the requirement to obtain labels from users imposes additional assumptions, increasing the number of queries required per training. Liang et al. proposed a knowledge-induced deep Q-learning model that actively uses the environment to drive objects and thus achieve grasping [26]. There are limitations in the robotic operation of this work for complex tasks that require the collaboration of different kinematic primitives. Shirin et al. built a grasping Q-network using a dual deep Q-learning framework to train the robot to master grasping techniques and a multi-view camera set up to observe the object, which greatly improved the grasping success rate [27]. More closely related to our work is the work of Zeng et al. [2], who introduced a Q-learning framework to simultaneously learn complementary push and grasp strategies. A Q-function reinforcement learning framework was estimated using a fully convolutional network (FCN) as a function approximator. Similar to Zeng et al.'s work on grasping point detection in images, Kechun Xu proposed a target conditional hierarchical reinforcement learning formulation with high sample efficiency to learn a push grasping strategy for grasping specific objects in clutter, achieving good results in terms of task completion rate and target grasping success rate [28]. And Yang et al. developed a deep learning approach driven by a critical strategy format to search for targets and rearrange the clutter around them for effective grasping [29]. However, it still has issues such as inaccurate grasping point positioning and low efficiency of pushing and grasping collaboration.

3. Methods

3.1. Reinforcement Learning

Reinforcement learning is an important branch of machine learning that differs from supervised and unsupervised learning in that it can be used to solve sequential decision-making problems. The concept is to emulate the human learning process through trial and error. Similarly, in reinforcement learning, an agent interacts with its environment in a trial-and-error process, using feedback signals from each action performed to assess how good or bad the action was and to update its decision-making strategy to make better decisions.

The problem study of reinforcement learning needs to be based on the Markov Decision Process (MDP) [30]. It first needs to satisfy the Markov Property, the future state s_{t+1} obeys a probability distribution that depends only on the current state s_t and the current action a_t , independent of the sequence of state actions at past moments:

$$P[s_{t+1} | s_t, a_t] = P[s_{t+1} | s_0, \dots, s_t; a_0, \dots, a_t] \tag{1}$$

For the MDP dynamic process, the initial state of the agent $S_0 \in \mathbb{S}$, and selects action $a_0 \in \mathbb{A}$ to execute. The environment enters the next state s_1 according to the state transfer function P , while the return r_0 is fed back to the agent, which then enters the process of the next decision. We can describe a state action interaction trajectory in terms of τ , $\tau = (s_0, a_0, s_1, a_1, \dots)$. The specific process is shown in Figure 1. In addition, a discount factor γ needs to be introduced. It is used to assess the cumulative future reward expectation, while a value of zero means that only immediate rewards should be considered. The undiscounted decaying returns and discounted decaying returns for the whole trajectory are:

$$R(\tau) = \sum_{t=0}^T \gamma^t r_t, \gamma \in (0, 1) \tag{2}$$

The performance of a policy can be assessed in terms of the expected return on the trajectory after the adoption of the policy. Assuming that the trajectory length is T and the environmental state transfer and strategy are random, then the probability of this trajectory of length T is:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1} | s_t, a_t) \pi(a_t | s_t) \tag{3}$$

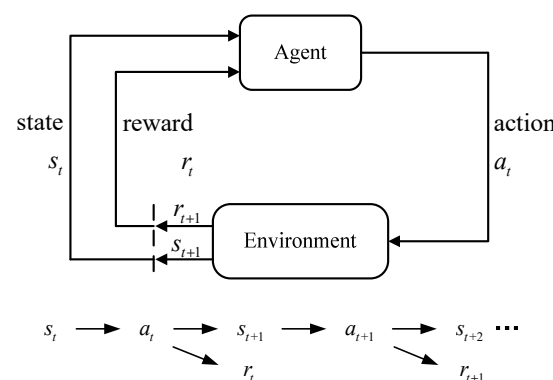


Figure 1. Reinforcement Learning Model.

In the process of optimizing a strategy, it is necessary to calculate the value of each state or state-action, expressed as a value function and a Q-value function, respectively. As both satisfy the Bellman equation, the optimal state value function and action-value function are introduced:

$$V^*(s) = \max_a \sum_{s' \sim P} [r(s, a) + \gamma V^*(s')] \tag{4}$$

$$Q^*(s, a) = \mathbb{E}_{s' \sim P} \left[r(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \quad (5)$$

The core of the optimal value function is the update of the value function, where the algorithm learns the value model through information from the interaction sequence and updates the strategy through the value model. Its optimal strategy is to choose the action with the highest Q value [31]:

$$a^*(s) = \operatorname{argmax}_a Q^*(s, a) \quad (6)$$

Dynamic planning can be used to update the action-value function in an environment where the state transfer model is known:

$$V^\pi(s) \leftarrow r(s, \pi(s)) + \mathbb{E}_{s' \sim P} [V^\pi(s')] \quad (7)$$

Whereas in the case of unknown models, the two main types of methods are Monte Carlo (MC) and Time Difference (TD). Monte Carlo is an unbiased estimation method with a strategy update that requires a complete trajectory, large variance, and low efficiency. Another class of TD algorithms, on the other hand, combines the advantages of Monte Carlo and dynamic programming. It enables both learning strategies from real data without the need for environmental models and single-step updates. Its updated formula is

$$Q(s, a) \leftarrow Q(s, a) + \alpha (r(s, a) + \gamma V(s') - Q(s, a)) \quad (8)$$

The off-policy form of the TD algorithm is the most widely used (Q -Learning) [32]. Often a network is used to fit the optimal Q . Therefore, during the iterative process, the TD-Target can be used as the update target for each set of state transfer samples:

$$y_i = r(s_i, a_i) + \gamma \max_{a'_i} Q_\theta(s'_i, a'_i) \quad (9)$$

3.2. State Space

As the state S_t in the Markov model, the method in this paper uses observations from vision sensors with RGB-D images of the robot's working area acquired by the depth camera at each moment. Pre-processing of camera-captured scene images using known camera parameters. After converting the RGB-D image to a 3-dimensional point cloud, the point cloud is projected vertically to the gravity direction to obtain a color mapping map and a height mapping map in the top view direction, and the point cloud is filled with 0 values to account for missing pixel point values in the mapping map calculation. Both mapping maps have the same resolution and are 224×224 in size, with pixel locations corresponding to each other. The value corresponding to each pixel point in the color mapping map in the depth mapping map is the depth information of that point in 3D space.

In this paper, the set of grasping actions of a robot is called the action space, and the position of an object in the space can be represented by six variables $(x, y, z, \alpha, \beta, \gamma)$. Therefore each grasping action in the action space should contain two elements: movement position (x_r, y_r, z_r) and movement gestures $(\alpha_r, \beta_r, \gamma_r)$. The grasping of the robot is set to grip in the vertical direction facing the table. The action posture is simplified to the rotation angle θ of the robot's end-effector along the z -axis, which is the direction of the robot's action. The range of rotation angles of the robot end gripper is $(0, 2\pi)$. The robot is set to have 16 different directions of grasping action at the same position, i.e., the range of rotation angles is discrete into 16 parts. The action space is parameterized according to the above settings. At time t , the robot's action space A contains $16 \times 224 \times 224$ grasping actions $a_t = (q, \theta)$, which are represented in the robot coordinate system as (x_r, y_r, z_r, θ) .

3.3. Rewards

There are two parts to the common reward design, reward r . The distance reward function, r_1 , represents the distance between the robotic arm's end and the target region's center point. When the end of the robotic arm is within the target region, the sparse reward function (r_2) is a single-step reward with an ambient feedback value of 1.

$$r_1 = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2} \quad (10)$$

$$r_2 = \begin{cases} 0 & \text{not at the target point} \\ 1 & \text{otherwise} \end{cases} \quad (11)$$

$r = r_1 + r_2$ is the algorithm's original reward function, which is used as the simulation robot arm's default reward function when the reward function is constant. This section designs a segmented reward function based on action effects for robot push-grasp learning. Different action effects are considered in the calculation of the reward for the current action, and environmental changes are taken into account in the calculation of the cumulative desired reward. The current award is defined as

$$R_{a_t}(s_t, s_{t+1}) = r + \Gamma(\phi_t)I_{a_t}(s_t, s_{t+1}) \quad (12)$$

The current reward consists of a basic movement reward $I_{a_t}(S_t, S_{t+1})$ and a movement effect reward $\Gamma(\phi_t)$. The basic action reward is the baseline and the action effect reward indicates the degree of reward or punishment. Andrew Hundt et al. proposed an exponential reward mechanism to determine the reward for different actions of the robot based on the type of action and the importance of the different actions [33]. The method in this paper involves two types of robots actions and expects to achieve the effect of pushing actions to assist grasping actions, so an exponential reward mechanism is used to generalize the basic action reward for pushing and grasping actions, defined as:

$$\Gamma_{exp}(\phi_t) = \eta_0 + \eta_1 2^{s - s_{\max}} \quad (13)$$

where η_0 and η_1 are action-independent parameters, $S_\varphi \leq S_{\max}$ is the reward parameter associated with the action type φ . For the two actions involved in this paper, the parameters of the push action $s_c = 0$ and the parameters of the grasp action $s_g = s_{\max} = 1$.

The designed action effect reward I_{a_t} is a segmentation function. During different training periods, the robot is rewarded with a penalty for the different results of the two actions during grasping. The action effect bonus consists of a grab action effect reward and a push action effect reward:

$$I_{a_t}(s_t, s_{t+1}) = \begin{cases} I_g(s_t, s_{t+1}), & \text{if } a_t = a_g \\ I_c(s_t, s_{t+1}), & \text{if } a_t = a_c \end{cases} \quad (14)$$

For grasping actions, the action effect reward I_g is defined as follows:

$$I_g(s_t, s_{t+1}) = \begin{cases} -1.0, & \text{if do nothing} \\ -0.5, & \text{if grasp empty} \\ -0.25, & \text{if touch but not grasp} \\ 0.0, & \text{if grasp but drop} \\ 1.0, & \text{if grasp and lift} \end{cases} \quad (15)$$

As the first two actions have no effect on the object being grasped, they can be classified as the same type of effect, dividing the effect of robot grasping into 4 cases: (1) empty grasp where there is no object; (2) gripper touches an object but fails to grasp it successfully; (3) grasps an object but fails to hold it and drops it; (4) successful grasp and removal of the object in four different situations, giving a bonus ranging from -1 to 1 . The gripping

effect of the robot can be judged by the distance between the fingers of the robot when the gripper is initially closed, the distance between the fingers after the gripper is closed and moved, and the change in state before and after the action. The change in the state before and after the movement is determined by calculating the difference between the depth mapping maps before and after the movement.

The push assists in the gripping action. The definitions are as follows:

$$I_c(s_t, s_{t+1}) = \begin{cases} -1.0 & \text{if } \phi_g(s_{t+1}) \leq \phi_g(s_t) \\ 1.0 & \text{otherwise} \end{cases} \quad (16)$$

Instead of only rewarding successful grasping, the robot is rewarded or punished for various grasping effects. The robot learns the coordination and synergy between the pushing and grasping actions by rewarding or punishing it with a churn based on the predicted outcome of the next training session.

3.4. Network

In designing the network architecture, we draw inspiration from the success of Zeng et al. [2]. In proposing an Actor-Critic approach that incorporates deep networks, there are several key differences that contribute to the superiority of our proposed framework over theirs.

DenseNet is a deep convolutional neural network with dense connections [34]. It helps to back-propagate the gradient during training and solves the problem of gradient disappearance or gradient explosion of traditional deep convolutional neural networks to some extent. The vision module of our framework is based on DenseNet-121's full convolutional network (FCN) model (Figure 2). Each depth height map and the corresponding color image are rotated 16 times at an angle of $\pi/8$ radians, and together they form the state input. It is important to note that the single-channel depth map is channel cloned and processed into the same three-channel (DDD) as the color image and normalized. After channel cascading the outputs of the three networks, the outputs are interleaved using two additional 1×1 convolutional layers, a nonlinear activation function (ReLU) [35] and spatial batch normalization [36], followed by bilinear up sampling, resulting in an output image with the same resolution as the input image, which predicts a dense pixel mapping with the same q-value as the output size. Each pixel in the output Q-value map represents a push or grasp action primitive at the corresponding 3D position, and the rotation angle of the height map corresponds to the rotation angle of the end-effector. The dense pixel approach parameterization simplifies the action space and thus speeds up the convergence.

In our framework, the core of the action module framework is the Actor-Critic algorithm. Our algorithm inherits the successful experience of DQN while (experience replay and target network freeze), adding Policy network for outputting continuous action values. The Actor learns the policy function π , the Critic learns the action value function Q, and the Actor makes an action according to the policy. The Critic gives Actor a score, and the Actor improves his strategy based on this score. The Critic relies on the rewards given by the environment to improve himself and make his scoring more and more accurate. Main implementation process: the Actor makes the robot arm perform the action according to the behavior strategy a_t , return r_t and the new state s_{t+1} . The Actor stores this state transition (s_t, a_t, r_t, s_{t+1}) in the replay memory buffer R as the dataset for training the online network. From the replay memory buffer R, N transitions are sampled according to the policy as a mini-batch training data for the online policy network and the online Q network. The target value network is fixed in the algorithm, and after updating the online Q and online policy networks, a Target_Q network with the same structure and parameters are copied from the online Q network again every C steps and samples data from the experience replay and outputs a stable target value Q_target. The specific algorithm is described in greater detail in the following section, with detailed steps listed in Algorithm 1.

The system has two components, an action module based on the Actor-Critic algorithm framework and a vision module based on the FNC framework. The visual 3D data observed

by the statically mounted RGB-D camera is reprojected onto the orthogonal RGB-D height map and entered into the FCN Vision Model as the current state s_t . A value network is the effect of performing a push or grab action based on the position of each pixel in the image corresponding to the space of the input Q network, visualizing the value of the action output from the network as a heat map with the same resolution as the input image. At the same time, we sample the mini-batch data from the replay memory buffer, update the policy network, and the value network.

In general, we want to optimize a policy with parameters so that it can learn a value function (Critic) from data interacting with the environment, which will help the policy function (Actor) learn a better policy. It is worth noting that the original push or grip action is chosen based on the acquired strategy. Also, the visual changes in the workspace are examined by comparing the workspace images in two subsequent states. If no significant visual change is detected, the robot will perform a pushing action. Otherwise, a grasping action will be executed. In self-supervised learning, this process is repeated over and over again. The grasping and pushing actions are trained using a single shared network, the Fully Connected Network.

Algorithm 1. Detailed steps of the improved algorithm CSAC-PER

```

1 : Initializing of the network  $Q(s, a | \theta^Q), \mu(s | \theta^\mu)$ 
2 : Initializing the parameters of the target network  $(Q', \mu') \theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$ 
3 : Initializing the noise of exploration  $\varepsilon$ 
4 : Initializing experience replay  $R_1, R_2$ 
5 : for episode = 1, M do
6 :   Initialize the environment and initial state  $S_1$ 
7 :   for  $t = 1, T$  do
8 :     for candidate  $j = 1, J$  do
9 :       Sampling the  $a_j$  according to the distribution  $W$ 
10 :    end for
11 :    Select the optimal  $K$  actions  $\mathcal{K} \leftarrow \text{argsort}\left(Q(s, \mathbf{a}_j)_{j=1}^J\right)_{1:K}$ 
12 :     $\mu = \frac{1}{K} \sum_{k \in \mathcal{K}} \mathbf{a}_k, \sigma = \frac{1}{K-1} \sum_{k \in \mathcal{K}} |\mathbf{a}_k - \mu|$ , Update  $W$ 
13 :    Output optimal action  $a_t = \mu + \varepsilon$ 
14 :    Store the samples  $(s_t, a_t, r_t, s_{t+1})$  into  $R_1$ 
15 :    Sort the data according to the reward function and store it in  $R_2$ 
16 :    Priority sampling of  $N$  samples from  $R_2$ 
17 :    Calculate the value function :  $\text{softmax}_\beta(Q'(s_{i+1}, a_{i+1}))$ 
18 :    Calculate the objective function :
19 :     $y_i \leftarrow r + \gamma(1 - d) \text{softmaxmax}_\beta(Q'(s_{i+1}, a_{i+1}))$ 
20 :    Updating the value network :
21 :     $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2$ 
22 :    Updating the policy network :
23 :     $\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a | \theta^Q) \Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s | \theta^\mu) \Big|_{s_i}$ 
24 :    Update target networks :
25 :     $\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$ 
26 :     $\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$ 
27 :  end for
28 : end for

```

where R_1, R_2 have experienced replay buffers of a size much smaller than the time step T ; d is an integer discrete value in the range of $[0, 1]$ to indicate whether the intelligence reaches the termination state during the learning process; ε is the action of adding noise, used to calculate the value function and the exploration of the agent, $\varepsilon \sim \text{clip}(N(0, \delta), -c, +c)$; The softmax operation for the continuous action space involves an integral that is difficult

to compute, so its unbiased estimate can be obtained by importance sampling; τ is the discount factor used to smooth the update process in the network update.

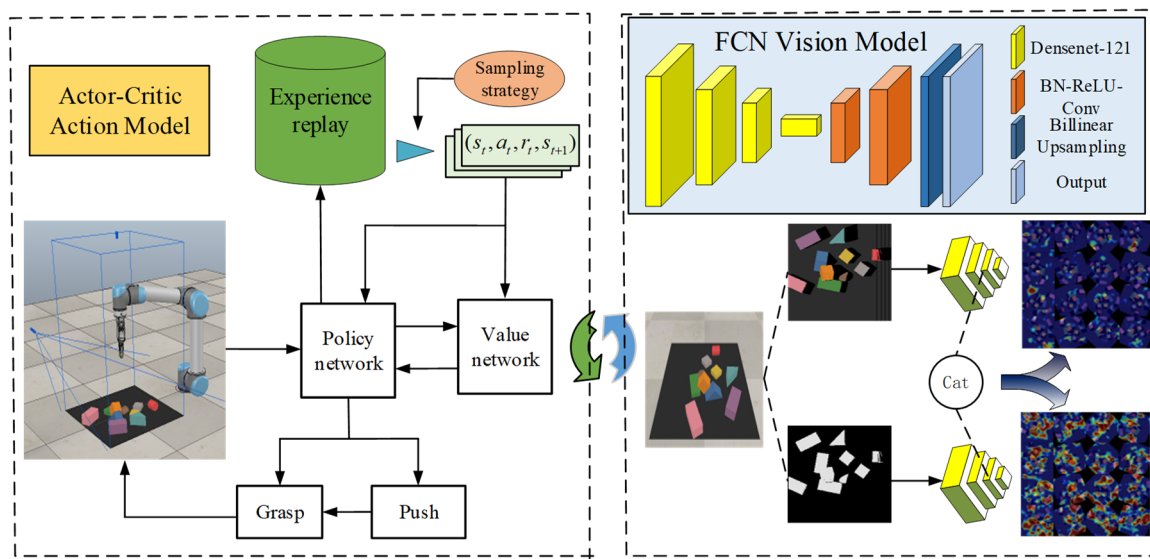


Figure 2. The framework of our system.

3.5. Dynamic Prioritized Experience Replay

The beta distribution, as an important class of underlying distributions in multivariate statistical analysis, has a wide range of applications in fields such as mathematical statistics [37]. In Bayesian inference, the Beta distribution is a conjugate prior distribution of the Bernoulli, binomial, negative binomial, and geometric distributions. Beta distributions come in many shapes, which can give the magnitude of the probability of occurrence of all probabilities. The expression of its probability density function is given by:

$$f(x) = \frac{x^{a-1}(1-x)^{b-1}}{B(a,b)} \tag{17}$$

The denominator in Equation (15) is the beta function, the expression of which is:

$$B(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1} dt \tag{18}$$

When the parameters a, b takes different values, the probability density function can be fitted to different function shapes (Figure 3). When $a = b = 1$, it happens to be uniformly distributed. Bayes says that knowing nothing also means that any probability is the same and is possible, and in the earliest papers, it was most common to use uniform distribution pairs for sampling in experience replay.

To avoid the high probability of samples with large TD errors being sampled, samples with low TD errors are never sampled, making the sampled samples lack diversity and leading to the overestimation of the training process. We use the Bata distribution function to sample the sample data after sorting the sample data in the experience reply according to their TD errors from largest to smallest based on priority experience playback so that samples with large and small TD errors have the same probability of being sampled, and the rest can be sampled approximately uniformly. As the number of training steps increases and when $\alpha = \beta = 1$ metamorphoses into common uniform sampling. Then α reaches the threshold value and β continues to increase, gradually reaching the effect of preferential experience replay.

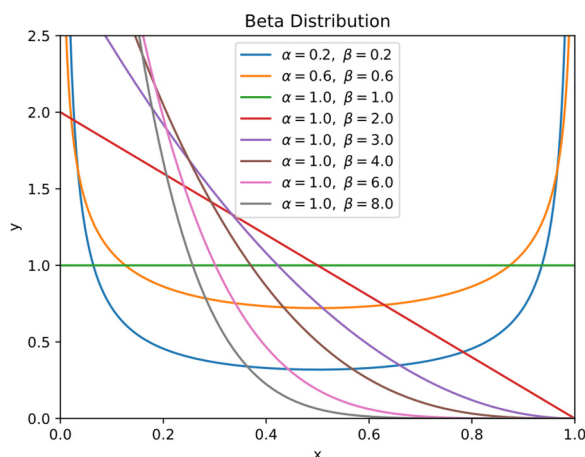


Figure 3. Beta distribution with different parameters.

Finally, using Beta distribution sampling can achieve the effect of priority sampling while also enriching the sampling samples and giving the machine more curiosity to explore unexpected strategies (Figure 4). The method avoids compensating for probability distribution errors and is easier to implement because it does not directly calculate the priority weights and sampling probabilities of the samples.

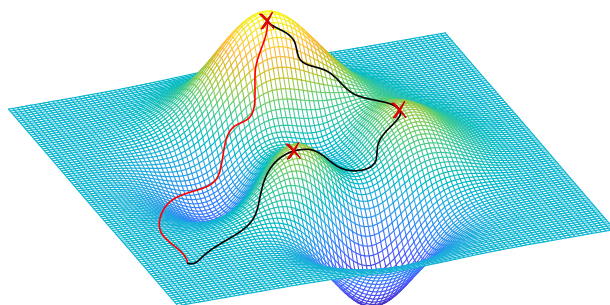


Figure 4. Two paths to explore the optimal strategy.

4. Experimental Results

The system uses an i7 9700k processor and an NVIDIA RTX 2070s for computing. Our system adopts the loss function using the Huber loss function and the optimizer is the Adam optimizer. Our simulation environment consists of a UR5 robot arm with an RG2 gripper in CoppeliaSim (shown in Figure 5).

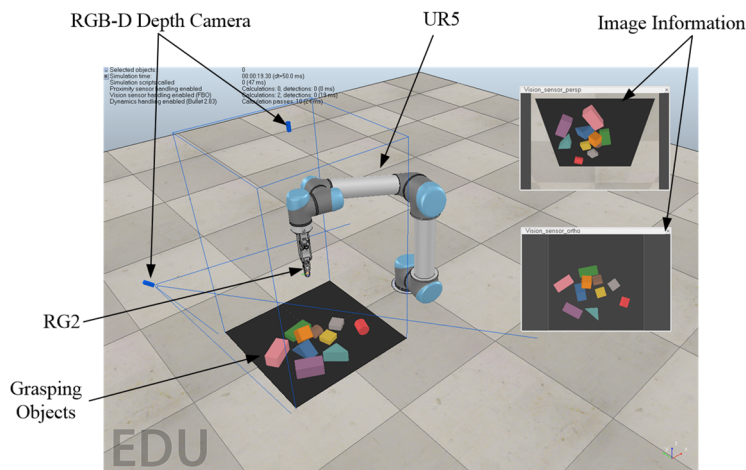


Figure 5. Simulation environment.

4.1. Baseline Methods

In this section, we perform a series of experiments to evaluate our system. The objectives of the experiments are: (1) prove that our training method is effective in speeding up the exploration and training process. (2) Verify that the proposed CSAC-PER algorithm is effective in improving the action synergy and achieving effective grasping of the target object with higher learning efficiency than the traditional sampling algorithm.

To verify the effect of exploration strategy and network structure on training speed and grasping accuracy, several baseline models were used as described below:

- (1) Grasping-only is a greedy deterministic grasping strategy that uses a fully convolutional network to predict the action and uses a greedy strategy to select the next action to be executed [38].
- (2) VPG maps action Q values by two action full convolutional networks and uses reinforcement learning methods to learn the synergy between “push” and “grasp” to achieve target grasping.

4.2. Evaluation Metrics

The proposed CSAC-PER and other baseline methods were evaluated in the test cases. The robot needs to pick up and clear all objects from the workspace. For each test case, n test runs ($n = 10$) were executed. The number of objects in the workspace varies in the range of 1–10 objects. Three evaluation metrics are used to assess the performance of the model. For all these metrics, the higher the value, the better. These metrics are as follows:

Grasping success rate: The ratio of the number of successful grasps to the total number of grasping actions performed in n test runs per test case.

Completion rate: A robot is successful if it achieves the grasp of the target within the action execution threshold, which measures the ability to grasp all objects in each test case.

Action efficiency: the proportion of successful grasping actions to all grasping actions is counted, and the average value of m groups of experiments is calculated, and this index measures the efficiency of the robot in completing the grasping task.

4.3. Simulation Experiments

The training phase sets the maximum threshold for the number of executed actions and defines that when the action number threshold is exceeded, the grasping environment is reset, and the next round of grasping training is implemented. If the target object is successfully grasped, a new target object is specified for the next training, and if there is no target object in the entire training area, the environment is reset for the next round of grasping training. The CSAC-PER algorithm is compared with other baseline methods for the training performance of the robot with 2500 training sessions. And the performance of the robot under different methods is plotted as shown in Figure 6.

It can be seen from the training performance graph that the Grasping-only method performs the worst during the training process. The training performance graph shows that the Grasping-only method performs the worst during the training process. Because it only uses the grasping method to achieve target grasping, it ignores the impact of the pushing action on the unknown environment, resulting in its worst training performance for target grasping, with a low success rate and a training performance of about 50%. Reinforcement learning is used in the VPG method to train the pushing action, which can change the unknown environment structure, better expose the target to the workspace, and achieve the grasping operation on the target. The VPG training efficiency for pushing is slightly higher than before, but the performance of pushing only to change the structure of the environment is not stable enough for grasping, and the training performance is around 75%. The average grasping success rate of the robot trained with the CSAC-PER algorithm continues to rise, but after 1500 training cycles, there is a tendency to converge, and the average success rate's growth rate slows. Although there were fluctuations during the last 1000 training sessions, the model gradually converged and the average grasping success

rate hovered around 80%, and the overall performance of training was better than other methods.

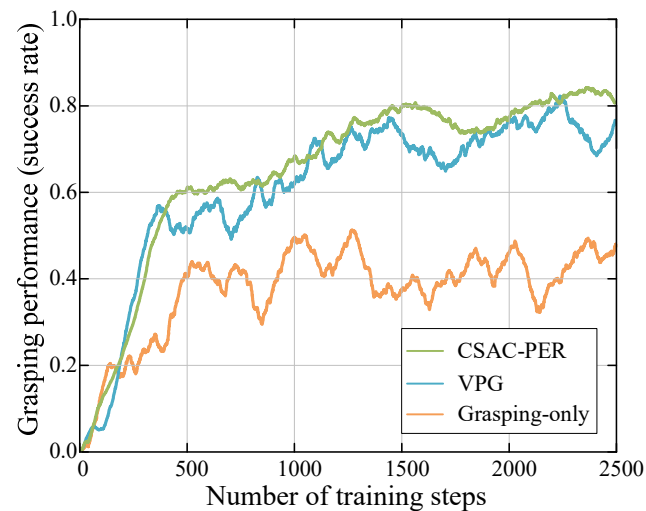


Figure 6. Grasping performance of different baselines methods.

We further conduct simulation experiments on the dynamic sampling of experience replay based on the CSAC algorithm. Firstly, two parameters, $\alpha = 0.5$ and $\beta = 0.5$, are initialized with thresholds of 1 and 12. And set the number of training steps to iteratively update α, β . For experience replay on training performance, the original CSAC algorithm and CSAC-PER, CSAC- β are compared, and the training performance is shown in Figure 7.

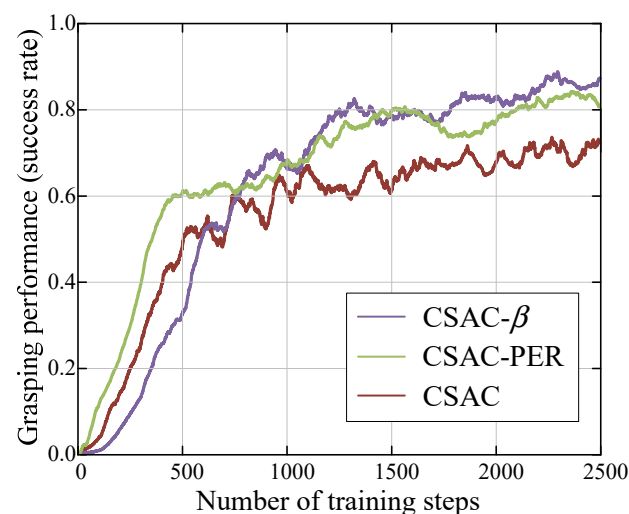


Figure 7. Grasping performance of different sampling algorithms.

At first, the robot will fully utilize the sampled samples and increase its exploration of the complex environment, so CSAC- β performance is low in the early stages. However, as the training progresses, the training performance improves, and the final grasping success rate is around 90%. Prioritized experience replay (PER) initially improves the algorithm's efficiency in utilizing samples, but updating the importance of sampling weights and priorities after each training increases the algorithm's complexity, particularly in robot grasping training, which can significantly increase training time. Although performance is higher in the early stages, and the final grasping success rate of CSAC-PER is around 80%, the number of trials is likely insufficient. The original CSAC uses a uniform distribution pair for sampling, which will flood the samples with higher values among all samples, and there is a problem of inefficient utilization, and the convergence is expected to be slower than the previous two.

4.4. Evaluation Tests

During the testing phase, four different test scenarios are selected from four different test scenarios to validate the training model’s performance (Figure 8). The test cases compare CSAC- β to other baseline methods in which the robot manipulates target objects with clutter effects by placing ordered irregularly shaped blocks against the environment. To test the task completion rate and action efficiency, 20 rounds of experiments were run for each test case to obtain an average value, and it should be noted that we programmed the robot to grasp the target object in 5 consecutive actions at a time.

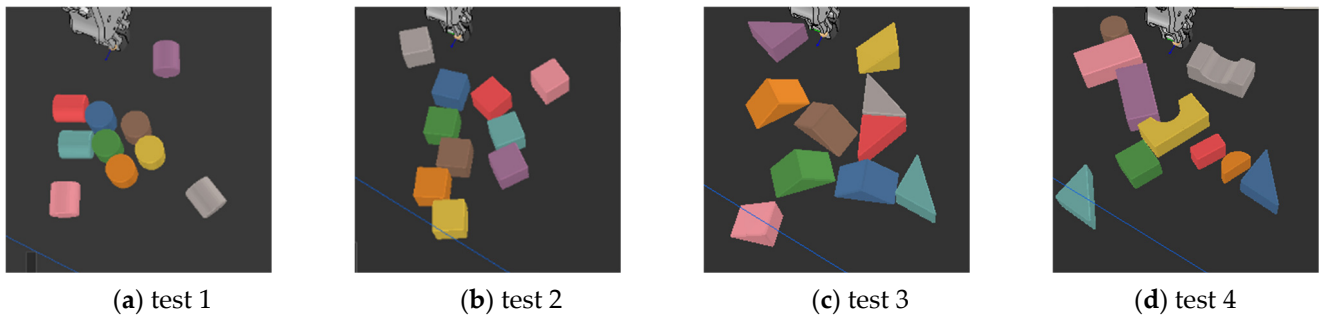


Figure 8. Four different test cases.

As shown in Figure 9, grasping-only has a relatively low action efficiency and completion rate for the grasping test task. For the VPG method with added pushing action, the action is selected only by the Q value of the predicted action, and the effect of action synergy is not obvious, and the efficiency and completion rate of the task action in the test cases are only about 70% and 60%. We propose a priority playback approach (CSAC-PER) to break uniform sampling and give greater sampling weights to samples with high learning efficiency, resulting in further improvement in action efficiency and task completion rate. Based on the a priori of beta distribution, we propose the dynamically prioritized experience replay of beta distribution (CSAC- β), which significantly improves task action efficiency and task completion rate, essentially reaching over 90%.

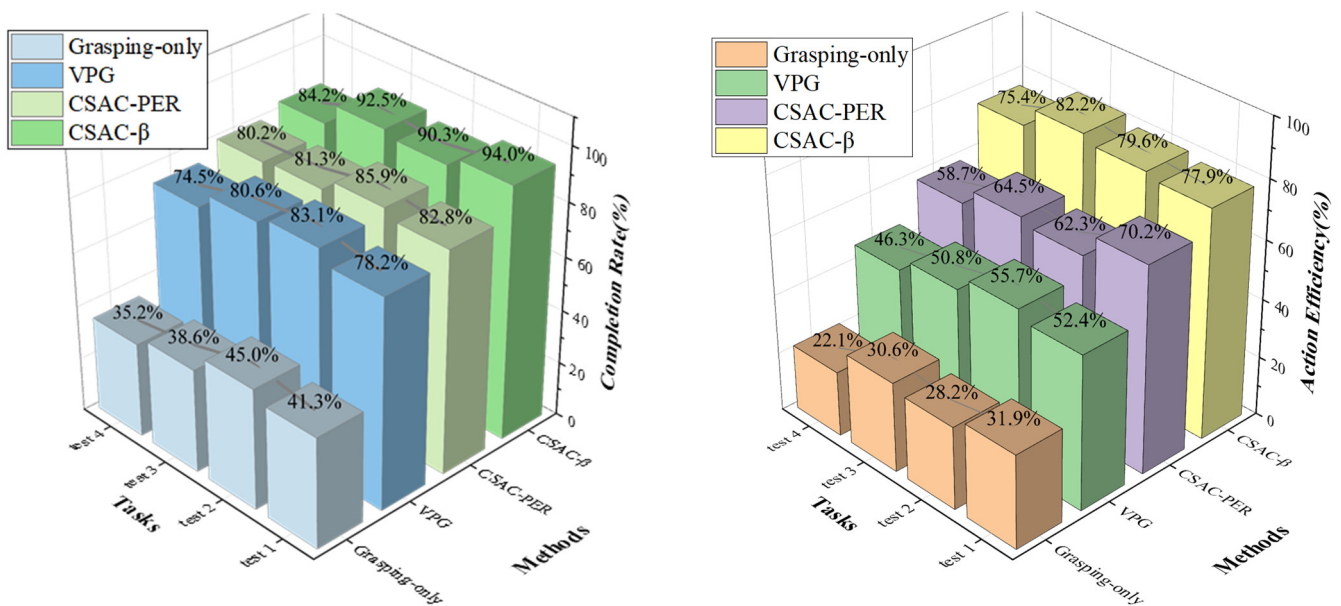


Figure 9. Evaluating the Performance of Grasping on Test Cases.

5. Conclusions

One of the challenges in robotics is to perform grasping tasks in unstructured environments. In this paper, we propose an improved a-c algorithm, consider a further dynamic sampling algorithm (CSAC- β) based on beta distribution based on prioritized experience replay, and experimentally investigate the robot's grasping performance in test scenarios involving random clutter and obscured objects. Our solution relies on self-supervised learning-based learning, which eliminates the need for extensive manual data collection and requires no human involvement throughout the process, and the control strategy involves a series of optimizations from sampling methods to sample selection to obtaining the optimal behavioral strategy for the value network. The simulation results show that the objects can be effectively removed from the workspace, that the grasping task can be successfully completed, that the grasping success rate is improved from the original (up to 90%), and that there is a significantly high completion rate (up to 90%) and action efficiency (up to 80%), indicating that the strategy is effective in terms of synergy push and grasping behavior. Although the efficiency of our proposed approach is not ideal, to begin with, as we weaken the idea of greed and focus more on future returns, the actual results also prove that the best returns in the present may not be the best in the future.

6. Discussion

We also note the following limitations of our approach. In the visual module, we follow DenseNet as the backbone network for feature extraction. With the development of computer vision, we prepare the attention mechanism to join the visual network of deep Q-network, especially channel attention (SE, CBAM) has a significant effect on improving the model performance. For the action module, the dynamic sampling of the experience pool requires us to determine the number of update steps to be set by ourselves according to the specific project scale and experience, which is uncontrollable. And due to the diversity of sampling methods, the preliminary efficiency is low. And inefficient upfront due to the diversity of sampling methods. We plan to address this issue in future work by incorporating dynamic sampling distribution of distributed adaptive experience pools into the framework. There is no set standard for training hyperparameters; instead, it is more empirical and thus a significant challenge. Inverse reinforcement learning by strategy or expert demonstration is a solution and one of the future research directions. Despite significant progress in DRL research, there are still issues such as insufficient generalization ability and failure to consider the robustness of the robot itself that limit the use of DRL in real-world applications. As part of future work, we will also improve the robot's robustness and resilience by improving its interaction with its environment.

Author Contributions: Conceptualization and methodology, T.Y. and H.W.; software, validation, formal analysis, investigation, data curation and writing—original draft preparation, T.Y.; writing—review and editing, T.Y., H.W. and X.X.; supervision, P.B.P. and A.R.; project administration, X.X. and H.W.; funding acquisition, H.W. and X.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Anhui Provincial Natural Science Foundation (2108085ME166), Natural Science Research Project of Universities in Anhui Province (KJ2021A0408), the Open Project of China International Science and Technology Cooperation Base on Intelligent Equipment Manufacturing in Special Service Environment (ISTC2021KF08).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhang, W.J.; Yang, G.; Lin, Y.; Ji, C.; Gupta, M.M. On definition of deep learning. In Proceedings of the 2018 World Automation Congress (WAC), Stevenson, WA, USA, 3–6 June 2018; pp. 1–5.
2. Zeng, A.; Song, S.; Welker, S.; Lee, J.; Rodriguez, A.; Funkhouser, T. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 4238–4245.

3. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [[CrossRef](#)] [[PubMed](#)]
4. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
5. Bahdanau, D.; Brakel, P.; Xu, K.; Goyal, A.; Lowe, R.; Pineau, J.; Courville, A.; Bengio, Y. An actor-critic algorithm for sequence prediction. *arXiv* **2016**, arXiv:1607.07086.
6. Rubinstein, R.Y.; Kroese, D.P. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*; Springer: New York, NY, USA, 2004.
7. Pan, L.; Cai, Q.; Huang, L. Softmax deep double deterministic policy gradients. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 11767–11777.
8. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952.
9. Zhao, D.; Wang, H.; Shao, K.; Zhu, Y. Deep reinforcement learning with experience replay based on SARSA. In Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece, 6–9 December 2016; pp. 1–6.
10. Fang, M.; Li, Y.; Cohn, T. Learning how to active learn: A deep reinforcement learning approach. *arXiv* **2017**, arXiv:1708.02383.
11. Lenz, I.; Lee, H.; Saxena, A. Deep learning for detecting robotic grasps. *Int. J. Robot. Res.* **2015**, *34*, 705–724. [[CrossRef](#)]
12. Redmon, J.; Angelova, A. Real-time grasp detection using convolutional neural networks. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 1316–1322.
13. Mahler, J.; Liang, J.; Niyaz, S.; Laskey, M.; Doan, R.; Liu, X.; Ojea, J.A.; Goldberg, K. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv* **2017**, arXiv:1703.09312.
14. Chu, F.J.; Xu, R.; Vela, P.A. Real-world multiobject, multigrasp detection. *IEEE Robot. Autom. Lett.* **2018**, *3*, 3355–3362. [[CrossRef](#)]
15. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In Proceedings of the Advances in Neural Information Processing Systems 28 (NIPS 2015), Montreal, QC, Canada, 7–12 December 2015; pp. 91–99.
16. Han, M.; Pan, Z.; Xue, T.; Shao, Q.; Ma, J.; Wang, W. Object-agnostic suction grasp affordance detection in dense cluster using self-supervised learning. *Docx. arXiv* **2019**, arXiv:1906.02995.
17. Wang, C.; Xu, D.; Zhu, Y.; Martín-Martín, R.; Lu, C.; Fei-Fei, L.; Savarese, S. Densefusion: 6d object pose estimation by iterative dense fusion. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 3343–3352.
18. Pinto, L.; Gupta, A. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 3406–3413.
19. Levine, S.; Pastor, P.; Krizhevsky, A.; Ibarz, J.; Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *Int. J. Robot. Res.* **2017**, *37*, 027836491771031. [[CrossRef](#)]
20. Kalashnikov, D.; Irpan, A.; Pastor, P.; Ibarz, J.; Herzog, A.; Jang, E.; Quillen, D.; Holly, E.; Kalakrishnan, M.; Vanhoucke, V.; et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation (2018). *arXiv* **2018**, arXiv:1806.10293.
21. Quillen, D.; Jang, E.; Nachum, O.; Finn, C.; Ibarz, J.; Levine, S. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018; pp. 6284–6291.
22. Breyer, M.; Furrer, F.; Novkovic, T.; Siegwart, R.; Nieto, J. Comparing task simplifications to learn closed-loop object picking using deep reinforcement learning. *IEEE Robot. Autom. Lett.* **2019**, *4*, 1549–1556. [[CrossRef](#)]
23. Clavera, I.; Held, D.; Abbeel, P. Policy transfer via modularity and reward guiding. In Proceedings of the 2017 IEEE/RISJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 1537–1544. [[CrossRef](#)]
24. Haarnoja, T.; Pong, V.; Zhou, A.; Dalal, M.; Abbeel, P.; Levine, S. Composable deep reinforcement learning for robotic manipulation. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018; pp. 6244–6251.
25. Singh, A.; Yang, L.; Hartikainen, K.; Finn, C.; Levine, S. End-to-end robotic reinforcement learning without reward engineering. *arXiv* **2019**, arXiv:1904.07854.
26. Liang, H.; Lou, X.; Choi, C. Knowledge induced deep q-network for a slide-to-wall object grasping. *arXiv* **2019**, arXiv:1910.03781.
27. Joshi, S.; Kumra, S.; Sahin, F. Robotic grasping using deep reinforcement learning. In Proceedings of the 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), Hong Kong, China, 20–21 August 2020; pp. 1461–1466.
28. Xu, K.; Yu, H.; Lai, Q.; Wang, Y.; Xiong, R. Efficient learning of goal-oriented push-grasping synergy in clutter. *IEEE Robot. Autom. Lett.* **2021**, *6*, 6337–6344. [[CrossRef](#)]
29. Yang, Y.; Liang, H.; Choi, C. A deep learning approach to grasping the invisible. *IEEE Robot. Autom. Lett.* **2020**, *5*, 2232–2239. [[CrossRef](#)]
30. Jaakkola, T.; Singh, S.; Jordan, M. Reinforcement learning algorithm for partially observable Markov decision problems. In Proceedings of the 7th International Conference on Neural Information Processing Systems, Denver, CO, USA, 28 November–1 December 1994; pp. 345–352.
31. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
32. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.

33. Hundt, A.; Killeen, B.; Greene, N.; Wu, H.; Kwon, H.; Paxton, C.; Hager, G.D. “Good robot!”: Efficient reinforcement learning for multi-step visual tasks with SIM to real transfer. *IEEE Robot. Autom. Lett.* **2020**, *5*, 6724–6731. [[CrossRef](#)]
34. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
35. Nair, V.; Hinton, G.E. Rectified linear units improve restricted boltzmann machines. In Proceedings of the Icml, Haifa, Israel, 21–24 June 2010.
36. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the International Conference on Machine Learning, PMLR, Lille, France, 6–11 July 2015; pp. 448–456.
37. Ferrari, S.; Cribari-Neto, F. Beta regression for modelling rates and proportions. *J. Appl. Stat.* **2004**, *31*, 799–815. [[CrossRef](#)]
38. Zeng, A.; Song, S.; Yu, K.T.; Donlon, E.; Hogan, F.R.; Bauza, M.; Ma, D.; Taylor, O.; Liu, M.; Romo, E.; et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018; pp. 3750–3757.