RM Bench: Benchmarking Deep Reinforcement Learning for Robotic Manipulator Control

Yanfei Xiang¹, Xin Wang²,³,§, Shu Hu, Bin Zhu⁵, Xiaomeng Huang¹, Xi Wu⁶,§, Siwei Lyu³

Abstract—Reinforcement learning is used to tackle complex tasks with high-dimensional sensory inputs. Over the past decade, a wide range of reinforcement learning algorithms have been developed, with recent progress benefiting from deep learning for raw sensory signal representation. This raises a natural question: how well do these algorithms perform across different robotic manipulation tasks? To objectively compare algorithms, benchmarks use performance metrics. Benchmarks use objective performance metrics to offer a scientific way to compare algorithms. In this paper, we introduce RM Bench, the first benchmark for robotic manipulations with high-dimensional continuous action and state spaces. We implement and evaluate reinforcement learning algorithms that take observed pixels as inputs and report their average performance and learning curves to demonstrate their performance and training stability. Our study concludes that none of the evaluated algorithms can handle all tasks well, with soft Actor-Critic outperforming most algorithms in terms of average reward and stability, and an algorithm combined with data augmentation potentially facilitating learning policies. Our code is publicly available at https://github.com/xiangyanfei212/RMBench-2022, including all benchmark tasks and studied algorithms.

I. INTRODUCTION

Robotic manipulations involve a robot, usually a robotic arm, interacting with objects in its surroundings. These interactions may include elevating an object above a certain height, moving an end effector to a specific location, or placing an object on top of another object. To complete these tasks intelligently, the robot must control its arms. Reinforcement Learning (RL) aims to teach an agent a good strategy for maximizing cumulative rewards by interacting with the environment. In robotic manipulations, RL algorithms offer the promise of machines with human-like abilities by learning dexterous manipulations directly from observed raw pixels [7], with both model-based [40], [41] and model-free [42]–[45] approaches available.

Model-based RL was historically used in robotics due to its high sample efficiency [46], enabling agents to complete tasks with minimal iterations. However, learning precise models for complex robotics control can be challenging, leading to poor model-based performance. To address issues with continuous state and action spaces, the Deep Deterministic Policy Gradient (DDPG) algorithm [13] was developed, opening doors for other model-free RL algorithms to solve robotic manipulation tasks with continuous state and action spaces. Since then, the Twin Delayed Deep Deterministic Policy Gradient (TD3) [15] and Soft Actor Critic (SAC) [16] algorithms have been developed, leading to significant advancements in this field. Since robotic manipulation tasks involve continuous states and actions, sensory states (i.e., observations) usually carry meaningful physical attributes such as position, velocity, and force, which vary over time. Impressive results have also been achieved using raw sensory inputs combined with deep learning advances [1]–[4].

Advances in deep reinforcement learning (Deep RL) have enabled autonomous agents to perform well on Atari games, often outperforming humans, using only raw pixels to make their decisions. However, most of these games take place in 2D environments that are fully observable to the agent. Because of the variations in shadow and light intensity, a robotic agent could not be able to fully comprehend the current state of the 3D environment from the 2D observed states. A critical insight in solving control tasks in real-world is learning better low-dimensional representations through autoencoders [50] and data augmentations [4], [22]. Therefore, it is necessary to find whether RL algorithms have human-like abilities by directly learning dexterous manipulation from observed raw pixels.

Benchmarks provide a systematic way to evaluate the strengths and limitations of existing algorithms. The Arcade Learning Environment (ALE) [5] offers a set of standard benchmarks for evaluating and comparing RL algorithms for tasks with high-dimensional state inputs and discrete actions. However, these benchmarks are not suitable for comparing RL algorithms designed for tasks with continuous actions, such as robotic manipulations. Duan et al. [6] propose a benchmark suite for continuous control tasks, including tasks with very high-dimensional states and actions such as 3D humanoid locomotion. However, their benchmark does not cater to robotic manipulation tasks. It is widely recognized that developing continuous control algorithms capable of observing high-dimensional images in RL is challenging, and such algorithms are actively studied in robotic manipulations using reinforcement learning. Therefore, it is critical to establish a consistent and rigorous testbed to benchmark these algorithms, providing a simulation framework and environment benchmark to facilitate the study and improvement of robotic manipulation solutions.
This paper introduces a benchmark, called RMBench, for evaluating the human-like abilities of RL algorithms in directly learning dexterous manipulations from observed raw pixels in a 3D environment. The benchmark includes five types of representative robotic manipulation tasks: lifting, placing, reaching, stacking, and reassembling. To evaluate RL algorithms, we implement state-of-the-art methods from two categories: policy optimization and actor-critic, including Vanilla Policy Gradient (VPG), Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), Delayed DDPG (TD3), and Soft Actor-Critic (SAC). We also use DrQ-v2 with autoencoders and data augmentation to improve environment learning. To address the high variance of vanilla policy gradient updates, we utilize General Advantage Estimation (GAE) for policy gradients in VPG, TRPO, and PPO. RMBench directly uses observed raw pixels as input for each task to evaluate an algorithm’s training policy performance and stability. Our results indicate that none of the tested algorithms can handle all tasks well, and actor-critic methods outperform policy-optimization methods overall. VPG, the simplest algorithm, performs well in most tasks, while SAC is effective for training deep neural network policies in terms of average reward and stability. We also find that data augmentation techniques and autoencoders used in DrQ-v2 can help agents gain richer environment information, even under unstable learning conditions.

Our main contributions can be summarized as follows:

- We present the first benchmark for RL-based robotic manipulation tasks with high dimensional continuous states and actions, including five common types of environments: lifting, placing, reaching, stacking, and reassembling.
- We provide a standardized set of RL-based benchmarking experiments as baselines for visual robotic manipulation tasks, with the aim of facilitating the investigation and improvement of robot manipulation solutions. We believe that agents trained in a 3D simulation environment can be transferred to real-world tasks.
- Novel findings are presented based on the performance and learning stability. We found that SAC is the most effective method, while VPG, despite being the simplest algorithm, performs satisfactorily in most tasks. Additionally, we show that data augmentation and autoencoders can help agents gain richer information about environments.

II. RELATED WORK

In many benchmarks, agents receive low-dimensional physical states as input to the policy, such as in robosuite [26], ROBEL [27], and RLLib [30]. Similarly, several RL competitions with low-dimensional actions have been conducted [31], [32]. In contrast, RMBench involves tasks with high-dimensional continuous state and action spaces. Here, we implement and evaluate RL algorithms that train policies directly using raw pixels as input.

Benchmarks have been proposed for Atari games with 2D state spaces, such as the ALE [5], which provides a benchmark for hundreds of Atari 2600 game environments. Liang et al. [29] propose a benchmark to evaluate the importance of key representational biases encoded by DQN’s network and provide a generic representation for ALE, significantly reducing the burden of learning a representation for each game. Ankesh et al. [28] systematically evaluate these biases by proposing simple linear representations and provide a simple, reproducible benchmark for the ALE, significantly reducing the burden of learning a representation for each game. While deep RL has enabled agents to perform well on Atari games, most of these games occur in fully observable 2D environments. A robotic agent cannot have full knowledge of the current state of the 3D environment, so building intelligent agents for real-world environments requires the ability to capture an environment’s latent characteristics. This paper focuses on robotic manipulation tasks in 3D simulation environments.

Benchmarks for robotic manipulations can be conducted using either real-world or simulated manipulations. Dasari et al. [35] proposed a benchmark for the standard 2-finger gripper based on real-world manipulations, which includes four tasks: pouring, scooping, zipping, and insertion. They used the benchmark to compare five algorithms that can be divided into two categories: open-loop behavior cloning and a model-based offline RL algorithm (MORel [36]). However, this benchmark is insufficient for comparing current state-of-the-art RL algorithms. Simulation provides a lower-cost alternative to sampling data from the real world. Aumjaud et al. [37] proposed a benchmark that trains policies in simulation before transferring them to an actual robotic manipulator. They used the benchmark to compare the performance of various model-free RL algorithms by solving the reaching task with a robot manipulator. However, evaluating an algorithm’s effectiveness based on a single task type is insufficient. In contrast, RMBench implements five different task types: lifting, placing, reaching, stacking, and reassembling, allowing for a more comprehensive evaluation of RL algorithms.

III. OUR ROBOTIC MANIPULATION BENCHMARK

RMBench provides a challenging benchmark for robotic manipulation tasks that serves as a performance indicator for current RL algorithms. It consists of five types of robotic manipulations: lifting, placing, reaching, stacking, and reassembling, with some types having multiple tasks. In total, nine tasks are included, which are visualized in Figure 1. Each manipulation environment returns a reward $r(s,a) \in [0,1]$ per time-step. The state and action spaces in robotic manipulation tasks are high dimensional and continuous. We selected seven RL algorithms based on their published time, type of policy, policy optimization method, and applicability to continuous action spaces. The agents were trained using raw pixels as inputs, without any auxiliary data pre-processing enhancements, to explore the ability of RL algorithms for high-dimensional state and action spaces.
To accelerate the training process, classic neural networks such as Feed-forward neural networks (FFN) and Convolutional neural networks (CNN) were implemented.

**Problem Definition.** We introduce the following notation, which will be used later in this work. Consider a standard infinite-horizon discounted Markov decision process (MDP) [49], defined by the tuple \((S,A,P,r,\rho_0,\gamma,T)\). Here, \(S\) is a (possibly infinite) set of states, \(A\) is a set of actions, \(P: S \times A \times S \to \mathbb{R} \geq 0\) is a transition function that defines a probability distribution over the next state given the current state and action, \(r: S \times A \to \mathbb{R}\) is a reward function, \(\rho_0: S \to \mathbb{R} \geq 0\) is an initial state distribution, \(\gamma \in (0,1)\) is a discount factor, and \(T\) is a finite horizon.

A policy is a rule used by an agent to decide what actions to take. In deep RL, we work with parameterized policies that we can improve using an optimization technique to change the agent’s behavior. Stochastic policies with parameter \(\theta\) are often defined by \(a_t \sim \pi_{\theta}(\cdot \mid s_t)\). For deterministic policies, it is usually denoted by \(a_t = \mu_{\theta}(s_t)\). Let \(J(\pi)\) denote its expected discounted reward: \(J(\pi) = \mathbb{E}_\tau \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right]\), where \(\tau = (s_0, a_0, s_1, a_1, \ldots)\) represents the whole trajectory, \(s_0 \sim \rho_0(s_0)\), \(a_t \sim \pi(a_t | s_t)\), and \(s_{t+1} \sim P(s_{t+1} | s_t, a_t)\). For whatever choice of return measure and whatever choice of policy, the objective of RL is to train a policy which maximizes the expected return. We use Generalized Advantage Estimation (GAE) [10] for computing the policy gradient. Let \(V\) be an approximate value function. Define the temporal difference (TD) residual \(\delta_t^V\) of \(V\) with discount \(\gamma\): \(\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)\). The sum of \(k\) of these \(\delta\) terms is defined as \(\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V\). The generalised advantage estimator \(\hat{A}_t^{GAE(\gamma,\lambda)}\) is defined as the exponentially-weighted average of these \(k\)-step estimators:

\[
\hat{A}_t^{GAE(\gamma,\lambda)} := (1 - \lambda) \left( \hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots \right) = \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}^V,
\]

where \(\gamma \in [0,1]\) and \(\lambda \in [0,1]\).

**Benchmarked Algorithms.** In RMBench, we have chosen representative RL algorithms based on their publication date (from 2000 to 2021), policy type (stochastic or deterministic), policy learning strategy (on-policy or off-policy), policy optimization method (policy optimization, Q-learning, or a combination of both), and ability to handle continuous action spaces. Table I compares their key characteristics and differences.

- **Vanilla Policy Gradient (VPG)** [8] [9]. VPG directly optimizes the stochastic policy by gradient ascent to maximize...
performance. The optimization is performed on-policy, which implies that each update uses experiences collected by the most recent policy version. This approach provides a nice reformulation of the derivative of the objective function, which simplifies gradient computation significantly. Since a large number of actions and states need to be estimated, this policy-based algorithm is particularly useful in continuous spaces.

- **Trust Region Policy Optimization (TRPO)** [11]. TRPO updates policies satisfying a specific constraint to ensure monotonic improvement during training. This constraint is the KL divergence, which measures the distance between probability distributions. By enforcing a small distance between old and new policies, TRPO avoids the policy collapse that can occur when using large step sizes to update policy parameters, as in VPG. As a result, TRPO can guarantee stable improvement through policy iteration.

- **Proximal Policy Optimization (PPO)** [12]. PPO is a group of first-order methods that aim to avoid the performance collapse caused by large updating steps by keeping new policies close to old ones. In contrast, TRPO uses a complex second-order method to address the same issue. PPO simplifies TRPO by using a clipped surrogate objective without KL-divergence, while maintaining performance. PPO-Clip is a variation of PPO that imposes the constraint by constraining the distance between new and old policies to a small, pre-determined interval.

- **Deep Deterministic Policy Gradient (DDPG)** [13]. Unlike the on-policy algorithms mentioned earlier, such as VPG, TRPO, and PPO, DDPG is an off-policy algorithm that updates a deterministic policy. It combines the Deterministic Policy Gradient (DPG) [14] and Deep Q-Network (DQN) [1] algorithms and simultaneously learns a Q-function and a policy in the actor-critic framework. To encourage exploration, noise is added to DDPG’s policies.

- **Twin Delayed DDPG (TD3)** [15]. TD3 employs three key strategies to prevent overestimation of the value function in DDPG, which is a common problem in that algorithm. First, Clipped Double Q-learning was developed to favor underestimation bias. Second, the Delayed update of the Target and Policy Networks strategy was implemented to reduce estimation variance and stabilize the training process. Finally, TD3 uses a smoothing regularization strategy on the value function to avoid estimating local peaks in value.

- **Soft Actor-Critic (SAC)** [16]. SAC is an on-policy actor-critic algorithm that trains a stochastic policy with entropy regularization. It is based on the entropy-regularized RL framework [19], optimizing the policy and value function networks separately. Unlike TD3, SAC aims to maximize both the expected return and the entropy simultaneously when training the policy, which leads to more exploration. Similar to TD3, SAC uses the clipped double-Q trick and takes the minimum Q-value between the two Q approximators. The entropy regularization coefficient is fixed in our benchmark.

- **DrQ-v2** [4]. Data augmentations are crucial to multiple recent visual RL algorithms [21–23]. DrQ-v2 applies image augmentation by randomly shifting pixel observations of states to solve tasks directly from these observations. The algorithm uses DDPG [13] as its backbone actor-critic RL algorithm with n-step returns [18], which leads to faster reward propagation and overall learning progress [20]. Furthermore, DrQ-v2 handles exploration noise variance by using a linear decay exploration schedule [24].

IV. EXPERIMENTS

A. Experimental Settings

**Setup.** We use the dm_control software package [47], which provides task suites for reinforcement learning agents in an articulated-body simulation. The software relies on Multi-Joint dynamics with Contact (MuJoCo) [38], a fast and accurate physics simulator [48]. Our focus is on manipulation tasks with a 3D robotic arm, including lifting, placing, reaching, stacking, and reassembling. We learn policies directly from raw state pixels, and each observed state is a visual image of $84 \times 84$ pixels. All the manipulation environments have a time limit of 10 seconds and return a reward $r(s, a) \in [0, 1]$ per time step.

**Implementation Details.** We train the agent for 200 episodes using five different random seeds, with each episode comprising 2000 steps. The implementation of each algorithm is based on existing public code\(^1\) with moderate modifications. We conduct experiments using the same training configuration and default hyper-parameter values as described in the original papers. To overcome the high variance issue of vanilla policy gradient updates, we employ the General Advantage Estimation (GAE) [10] method (Eq.(1)) for on-policy algorithms (VPG, TRPO, and PPO) with a $\lambda$ value of 0.95. For actor-critic algorithms (DDPG, TD3, and SAC), we use several techniques to improve exploration and stability during training. Firstly, we include a few steps (10000) of random action selection before running the actual policy. Secondly, we perform gradient descent updates until 1000 interactions to ensure that the replay buffer is adequately filled for useful updates. Thirdly, we update the networks every 50 environment interactions, allowing the actor and critic networks to change slowly and improve learning stability. Additionally, we add Gaussian exploration noise (standard deviation=0.1) to the policy during training.

**Network Architecture.** We employ a feed-forward neural network policy with three hidden layers, consisting of 512, 256, and 128 hidden units with tanh activation at each hidden layer, for all algorithms except DrQ-v2. In all cases, agents are trained using raw flattened pixels in RGB format. For the DrQ-v2 algorithm, we adopt the original setting of four convolution layers (without pooling) with 32 filters at each layer, followed by two fully connected layers with 1024 units. The inputs to DrQ-v2 are obtained through image

\(^1\)https://github.com/openai/spinningup and https://github.com/facebookresearch/drqv2
TABLE II: The MER performance (mean and the standard deviation) of the seven RL algorithms with the nine robotic manipulation tasks, which is averaged over 200 episodes and five different random seeds. The best, second, and third best results are highlighted by different shades of blue, with the darkest blue indicating the best.

| Task                              | VPG          | TRPO         | PPO          | DDPG         | TD3          | SAC           | DrQ-v2        |
|-----------------------------------|--------------|--------------|--------------|--------------|--------------|---------------|---------------|
| Place brick                       | 4.360 ± 0.085| 4.723 ± 0.162| 4.371 ± 0.090| 4.314 ± 0.096| 4.372 ± 0.086| 5.03 ± 0.058 | 4.429 ± 0.101 |
| Place cradle                      | 4.319 ± 0.059| 4.552 ± 0.121| 4.385 ± 0.058| 4.305 ± 0.070| 4.339 ± 0.17  | 4.942 ± 0.102| 4.308 ± 0.178 |
| Reach duplo                       | 2.832 ± 1.778| 7.296 ± 1.465| 2.399 ± 1.501| 0.589 ± 0.553| 0.923 ± 1.765| 9.297 ± 0.635| 1.906 ± 1.135 |
| Reach site                        | 1.217 ± 0.26 | 3.975 ± 0.719| 1.161 ± 0.270| 1.380 ± 0.090| 1.218 ± 0.148| 4.519 ± 0.172| 1.904 ± 0.770 |
| Lift large box                    | 0.003 ± 0.001| 0.002 ± 0.000| 0.003 ± 0.001| 0.003 ± 0.001| 0.004 ± 0.002| 0.002 ± 0.000| 0.002 ± 0.000 |
| Lift brick                        | 0.002 ± 0.000| 0.001 ± 0.000| 0.001 ± 0.000| 0.001 ± 0.000| 0.002 ± 0.001| 0.001 ± 0.000| 0.001 ± 0.001 |
| Stack 2 breaks                    | 0.148 ± 0.011| 0.144 ± 0.013| 0.145 ± 0.013| 0.148 ± 0.011| 0.145 ± 0.012| 0.147 ± 0.013| 0.142 ± 0.008 |
| Stack 2 breaks movable base       | 0.147 ± 0.009| 0.144 ± 0.013| 0.143 ± 0.009| 0.150 ± 0.008| 0.147 ± 0.012| 0.143 ± 0.013| 0.142 ± 0.018 |
| Reassemble 5 bricks random order  | 60.543 ± 0.779| 59.893 ± 1.590| 60.347 ± 2.280| 60.244 ± 1.712| 59.018 ± 2.068| 56.769 ± 1.180| 61.131 ± 2.912 |

Fig. 2: Training convergence curves of seven RL algorithms on the nine robotic manipulation tasks: (a) place brick, (b) lift brick, (c) reach duplo, (d) place cradle, (e) lift large box, (f) reach site, (g) stack 2 bricks, (h) stack 2 bricks movable base, (i) reassemble 5 bricks random order. Shaded regions correspond to one standard deviation.

Augmentation of random shifts to pixel observations of states.

B. Performance Metrics

To measure the performance of an episode during training, we use Episode Reward (ER):

\[ ER^i = \sum_{t=1}^{T} R^i_t \]  

(2)

where \( T \) is the number of steps in an episode, and \( R^i_t \) is the undiscounted return for the \( t \)-th step of the \( i \)-th episode. The average performance for a specific task is measured by the Mean Episode Reward (MER):

\[ MER = \frac{1}{T} \sum_{i=1}^{I} ER^i = \frac{1}{T} \sum_{i=1}^{I} \sum_{t=1}^{T} R^i_t \]  

(3)

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
where $I$ is the number of episodes.

![Fig. 3: The average performance of each algorithm for nine benchmark tasks in RMBench. The y-axis demonstrates Mean Episode Reward (MER). The x-axis means RL algorithms. Green bars mean algorithms using policy optimization. Blue bars mean algorithms using the actor-critic mechanism. Bars filled with slanted lines indicate algorithm using stochastic policy, and bars not filled indicate algorithm using deterministic policy.](image-url)

C. Results and Discussion

We present the results in Table II, Fig. 2, and Fig. 3. Table II shows the average performance of all episodes, with each entry displaying a pair of numbers representing the mean and standard deviation of MER. Fig. 2 shows the training convergence curves for the five initialization seeds at each episode, with the $x$-axis denoting the training episode and the $y$-axis denoting the Episode Reward (ER). Solid lines represent the average values over five random runs, and the shaded regions correspond to one standard deviation. Note that we smooth the curve by averaging across ten episodes, as in [29], [51]. Fig. 3 displays the average performance of each algorithm for all tasks. Based on these results, we make the following observations:

**Algorithm Level.** 1) **VPG:** Although VPG is a simple algorithm, it performs acceptably for most robotic manipulation tasks. However, it can sometimes converge too quickly to a local optimum, as shown in Fig. 2(a)(c)(d)(f), which is also noted by Peters et al. [33]. 2) **TRPO:** TRPO performs competitively on tasks such as placing and reaching, indicating that updating the policy when satisfying the KL-Divergence constraint is beneficial for ensuring consistent improvement over policy iteration. 3) **PPO:** PPO can be seen as a simplified version of TRPO. Our simulations show that PPO performs worse than TRPO, possibly due to the instability of clipped-PPO when rewards vanish outside bounded support on continuous action spaces. PPO is also sensitive to policy initialization when there are locally optimal actions close to the initialization [34]. 4) **DDPG:** DDPG’s performance is mediocre. Large replay buffers can help DDPG benefit from learning over a collection of uncorrelated transitions [13], and appropriate noises that facilitate temporally correlated exploration are critical for physical control problems [52]. 5) **TD3**: TD3, based on DDPG, applies a couple of tricks, such as Clipped Double Q-learning and "Delayed" Policy Updates. The simulation results show that TD3 performs similarly to DDPG, indicating that these tricks do not significantly improve performance in these tasks. 6) **SAC:** SAC shows the best performance in terms of average reward and stability. A central feature of SAC is entropy regularization, which has a better trade-off between exploration and exploitation, encouraging greater exploration and preventing the policy from prematurely converging to an inferior local optimum. 7) **DrQ-v2:** DrQ-v2 is an extension of DDPG that adds data augmentation and a convolutional encoder. DrQ-v2 performs better than DDPG in terms of MER, likely due to the benefits of data augmentation and the convolutional encoder, which provide richer environmental information and improve the agent’s ability to learn policies efficiently. However, the training phase is unstable, as indicated by the standard deviations and fluctuating training curves. This instability may be due to the loss of state details caused by convolution operations.

**Category Level.** 1) **On-policy and off-policy:** Our benchmark includes three on-policy algorithms (VPG, TRPO, and PPO) and four off-policy algorithms (DDPG, TD3, SAC, and DrQ-v2). From Fig. 3, we can see that the performances of the two types of algorithms are similar. Additionally, TRPO is the most potent on-policy algorithm, and SAC is the best off-policy algorithm. SAC is slightly better than TRPO, suggesting that the sample collection approach used in off-policy algorithms can lead to better exploration. 2) **Only policy optimization and actor-critic:** Policy optimization algorithms can optimize policy directly, which makes them more principled and stable. Q-learning, on the other hand, optimizes policy indirectly to improve performance, resulting in less stability but better sample efficiency. Actor-critic methods fall between policy optimization and Q-learning. From Fig. 3, actor-critic algorithms such as DDPG, TD3, SAC, and DrQ-v2 perform comparably to policy optimization methods, indicating that actor-critic approaches can provide a satisfactory compromise between policy optimization and Q-learning. 3) **Deterministic policy and stochastic policy:** To explore the entire state and action spaces, a stochastic policy is often necessary, where the policy gradient is computed by integrating both state and action spaces. A deterministic policy only integrates over the state space, so noise is often added to it for more exploration. According to Silver et al. [14], deterministic policies provide significant performance benefits for high-dimensional tasks. From Fig. 3, the performance of algorithms based on the stochastic policy such as VPG, TRPO, PPO, and SAC is comparable to that of algorithms based on a deterministic policy. Therefore, the stochastic policy is more capable of producing satisfactory results, indicating the importance of exploration during the learning phase. 4) **Data Augmentation:** DrQ-v2 is an RL algorithm based on DDPG that uses data augmentation and convolutional encoder. The simulations show that DrQ-v2 outperforms DDPG in most tasks, including placing, reaching, and reassembling.
DrQ-v2 achieves the best performance in the task of reassembling 5 bricks in random order. Therefore, data augmentation and convolutional encoder can help agents understand pixel-level states. Since the data augmentation strategy can be easily applied to all RL algorithms in visual control tasks, a combination of advanced RL algorithms, such as SAC, and data augmentation can achieve more satisfactory results. A larger replay buffer is crucial, especially in tasks with a wider variety of starting state distributions [53]. Therefore, a substantial replay buffer is necessary when using data augmentation.

**Task Level.** Different tasks require different algorithms to produce favorable outcomes. For instance, SAC and TRPO demonstrate superior performance in placing manipulation tasks, such as brick and cradle. Furthermore, SAC is highly effective in reaching tasks like duplo and reaching site, followed by TRPO. In the task of reassembling five bricks, DrQ-v2 achieves the highest mean episode reward. However, all the algorithms exhibit disappointing performance in lifting and stacking tasks due to unstable training curves and mean episode rewards. This may be because these tasks demand more precise and complex coordination of arm and finger joints.

V. CONCLUSION AND FUTURE WORK

In this paper, we introduce RMBench, the first-ever benchmark for robot manipulations that evaluates the human-like abilities of seven RL algorithms by directly learning dexterous manipulations from observed raw pixels. RMBench comprises nine robotic manipulation tasks, including lifting, placing, reaching, stacking, and reassembling. We implement and evaluate seven model-free RL algorithms and provide a standardized set of RL-based benchmarking experiments as baselines for visual robotic manipulation tasks. The simulations reveal that none of the algorithms perform well across all tasks. Despite being the simplest algorithm, VPG performs satisfactorily in most benchmark tasks. SAC is an effective method for training deep neural network policies in terms of average reward and stability among the tested algorithms. Although the learning of DrQ-v2 is unstable, data augmentation and autoencoder used in DrQ-v2 can help agents gain richer information about environments. However, all the implemented RL algorithms exhibit poor performance on the lifting tasks, emphasizing the need for developing new and improved algorithms. Since agents trained in the 3D simulation environment can be transferred to real-world tasks, the results of this study provide a useful guideline for leveraging RL algorithms in real-world robotic manipulation tasks. Overall, the objective of RMBench is to simplify the study and enhancement of robot manipulation solutions by providing a simulation framework and environment benchmark.

**Future Works.** To enhance the diversity of tasks, RMBench should include additional robotic manipulation tasks. It is crucial to evaluate whether agents trained in simulators can be seamlessly transferred to real-world situations, given the inherent discrepancies between simulator dynamics and the real world. Since robotic simulation environments are more complex than 2D games due to 3D distances and variations in shadow and light intensity, it is necessary to consider enhancing the understanding and extraction of state information. Applying deeper neural networks can improve the understanding of states, and data augmentation methods used in the Computer Vision field can enhance sample efficiency during the learning phase. Additionally, future research should compare the time cost of different algorithms by fixing the CPU and GPU configurations to record wall-clock time during training. We leave this as a potential area for future work.

**References**

[1] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, Art. no. 7540, Feb. 2015, doi: 10.1038/nature14236.
[2] X. Guo et al., “Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning,” NIPS, 2014, vol. 27.
[3] M. Watter et al., “Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images,” NIPS, 2015, vol. 28.
[4] D. Yarats et al., “Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning,” arXiv:2107.09645 [cs], Jul. 2021.
[5] M. G. Bellemare et al., “The Arcade Learning Environment: An Evaluation Platform for General Agents,” JAIR, vol. 47, pp. 253–279, Jun. 2013, doi: 10.1613/jair.3912.
[6] Y. Duan et al., “Benchmarking Deep Reinforcement Learning for Continuous Control,” ICML, Jun. 2016, pp. 1329–1338.
[7] H. Nguyen and H. La, “Review of Deep Reinforcement Learning for Robot Manipulation,” IRC, Feb. 2019, pp. 590–595.
[8] R. S. Sutton et al., “Policy Gradient Methods for Reinforcement Learning with Function Approximation,” NIPS, 1999, vol. 12.
[9] J. Schulman, “Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs,” UC Berkeley, 2016.
[10] J. Schulman et al., “High-Dimensional Continuous Control Using Generalized Advantage Estimation,” arXiv, arXiv:1506.02438, Oct. 2018.
[11] J. Schulman et al., “Trust Region Policy Optimization,” ICML, Jun. 2015, pp. 1889–1897.
[12] J. Schulman et al., “Proximal Policy Optimization Algorithms.” arXiv, Aug. 28, 2017.
[13] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” arXiv, arXiv:1509.02971, Jul. 2019.
[14] D. Silver et al., “Deterministic Policy Gradient Algorithms,” ICML, Jan. 2014, pp. 387–395.
[15] S. Fujimoto et al., “Addressing Function Approximation Error in Actor-Critic Methods,” arXiv, arXiv:1802.09477, Oct. 2018.
[16] T. Haarnoja et al., “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” arXiv:1801.01290 [cs, stat], Aug. 2018.
[17] T. Haarnoja et al., “Reinforcement Learning with Deep Energy-Based Policies,” arXiv:1702.08165 [cs], Jul. 2017.
[18] G. Barth-Maron et al., “Distributed Distributional Deterministic Policy Gradients,” arXiv, arXiv:1804.08617, Apr. 2018.
[19] B. D. Ziebart et al., “Maximum entropy inverse reinforcement learning,” in Aaai, 2008, vol. 8, pp. 1433–1438.
