Achieving Sample-Efficient Learning of Long-Horizon Sparse-Reward Robotic Tasks with Base Controllers

Minjian Xin, Guangming Wang, Zhe Liu and Hesheng Wang

***Abstract*— Application of Deep Reinforcement Learning (DRL) algorithms in robotic tasks faces many challenges. On the one hand, reward-shaping for complex tasks that involve multiple sequences is difficult and may result in sub-optimal performances. On the other hand, a sparse-reward setting renders exploration inefficient, and exploration using physical robots is of high-cost and unsafe. In this paper we propose a method of learning long-horizon sparse-reward tasks utilizing one or more existing controllers. Built upon Deep Deterministic Policy Gradients (DDPG), our algorithm incorporates the controllers into stages of exploration, policy update, and most importantly, learning a heuristic value function that naturally interpolates along task trajectories. Through experiments ranging from stacking blocks to cups, we present a straightforward way of synthesizing these controllers, and show that the learned state-based or image-based policies steadily outperform them. Compared to previous works of learning from demonstrations, our method improves sample efficiency by orders of magnitude. Overall, our method bears the potential of leveraging existing industrial robot manipulation systems to build more flexible and intelligent controllers.**

I. INTRODUCTION

Deep Reinforcement Learning (DRL) has been applied extensively to complicated decision-making problems in the past few years. Meanwhile, a lot of research has been done on applying DRL to robot control problems [1]–[4]. However, DRL algorithms are often data-hungry, requiring millions of interactions with the environment to learn a single task. Exploration is even more inefficient when the reward is sparse and delayed. While random exploration of a robot in the real world is of high-cost and unsafe, many DRL algorithms lack the consideration of exploration cost and assume a simple learner-controlled exploration strategy.

Previous works have used demonstrations [5]–[8] to speed up DRL. Although this method could successfully learn otherwise unlearnable long-horizon sparse-reward tasks, it still is not a perfect solution to the efficiency problem. And learning tasks like pick-and-stack still take millions of interactions with the environment. Furthermore, additional hardwares like virtual reality equipments or 3D motion controllers are required to collect demonstrations.

*This work was supported in part by the Natural Science Foundation of China under Grant U1613218, 61722309 and U1913204, in part by Beijing Advanced Innovation Center for Intelligent Robots and Systems under Grant 2019IRS01, and in part by grants from NVIDIA Corporation. Corresponding Author: Hesheng Wang.

M. Xin, G. Wang, and H. Wang are with Department of Automation, Insititue of Medical Robotics, Key Laboratory of System Control and Information Processing of Ministry of Education, Key Laboratory of Marine Intelligent Equipment and System of Ministry of Education, Shanghai Engineering Research Center of Intelligent Control and Management, Shanghai Jiao Tong University, Shanghai 200240, China. Z. Liu is with the Department of Computer Science and Technology, University of Cambridge.

In robotic tasks, there are many shared task structures and common action primitives. Also, non-DRL based control policies may serve as priors for learning. Considering these, we propose a hybrid method for robotic task learning exploiting existing controllers, which we refer to as *base controllers*. The main contributions of this paper include:

- We present a novel learning scheme that incorporates base controllers for resolving the challenges of sparse rewards and long task horizons. The proposed approach greatly relives the exploration cost during learning.
- We extend our approach to an ensemble of base controllers, and show our generalizabilities to various state based to raw-sensor-data based tasks, in all of which the final learned policies outperform the base controllers.
- The results validate that, compared with learning from demonstrations, our approach accelerates the learning speed by orders of magnitude and is also effective in handling difficult tasks which can not be solved by demonstration learners.

II. RELATED WORK

Imitation Learning (IL) is broadly adopted in robotics, ranging from automated driving [9], [10] to Unmanned Aerial Vehicle control [11]. The simplest way of doing IL is Behavior Cloning (BC) [9], which intends to match actions of the learner to those of an expert. BC is not safe due to overfitting and distribution drift issues. IL algorithms like Dataset Aggregation (DAGGER) [12], [13] address this by collecting data with annealed ratio of expert control, which helps achieve low-cost learning, or regret-minimization. Nevertheless, a huge amount of expert data needs to be collected. Recent algorithms can also learn implicitly from demonstration data, such as Inverse Reinforcement Learning (IRL) [14] and Generative Adversarial Imitation Learning (GAIL) [15].

State-of-the-art DRL algorithms have been applied in the robot learning field, ranging from robot arm control using Deep Q-Network (DQN) [16], [17], dexterous robot hand manipulation using Proximal Policy Optimization (PPO) [4], [18], up to applications of Deep Deterministic Policy Gradient (DDPG) [19], [20], and Soft Actor Critic (SAC) [21]. For simple sparse-reward settings, Hindsight Experience Replay (HER) [22] proposes the notion of goal-conditioned tasks. But HER is hard to apply to more complex tasks with bottlenecks [8].

Researchers have been trying to combine IL with RL to get the best of both worlds. [23] puts demonstration in the replay buffer of DDPG to learn peg insertion tasks, [5] augments DDPG with a Q-filtered BC loss, and shows that the robot is

able to learn a multi-step sparse-reward block-stacking task. [6] employs GAIL to learn from demonstrations, and uses state prediction as auxiliary tasks to learn a deep visuomotor policy for a robot hand. [8] uses demonstrations to overcome bottlenecks in HER. [7] samples demonstrations according to a curriculum of increasing difficulty to help learn a visuomotor policy. All of the above works use demonstrations but learning tasks like pick-and-stack still take millions of environment interactions. Also, additional hardwares are used. Specifically, [5] uses virtual reality equipments, [6] uses 3D motion controllers, and [7] uses a 3D mouse. Finally, the agent is prone to making mistakes in the initial stage when learning from demonstrations, which is unfavorable when mistakes are expensive.

In contrast to learning from demonstrations, learning with existing controllers often directly uses a controller to aid exploration. Guided Policy Search (GPS) [24], [25] optimizes a model-based controller and then learns a policy in the supervised manner. Residual RL [26]–[28] learns a correction policy using RL on top of a feedback controller. One major characteristic of residual RL is that it only learns a compensation policy that is strongly dependent on the controller, and it is unclear how well such policy generalizes to different scenes or different controllers, while our method learns a complete policy and the controller can be discarded once learning is done. Also, our method can utilize an ensemble of controllers, while residual RL can only use one controller and cannot cope with non-deterministic controller policies. The crucial difference between our work and above works is that both GPS and Residual RL focus on tasks like insertion where the robot moves an already-grasped object to the goal position, while we focus on more sequentially involved tasks with longer task horizons.

There are also previous works that apply a "mixed" exploration strategy when learning with existing controllers. [29] uses a simple proportional controller as "training wheel", and trains a Deep Q-Network to decide which action to execute between the proportional controller and agent's policy. [30] learns a plane cruising task using a proportional controller, whose control ratio is linearly decayed to ensure safety in the start. Our method uses a more principled exploration strategies compared to these works.

III. METHOD

We formulate the Markov Decision Process (MDP) as per standard RL. For state set $\mathcal{S}$ and action set $\mathcal{A}$ in environment E, a policy μ is a mapping $\mathcal{S} \rightarrow \mathcal{A}$, and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition probability. For every time step t, the agent receives observation s_t, takes an action a_t, then receives next observation s_{t+1} and reward r_t. The goal is to maximize the accumulated discounted reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$ where γ is the discount factor. The optimal Q-value is defined as the expected return from a state after taking an action and acting optimally thereafter: $Q^*(s_t, a_t) = \max_{\mu} \mathbb{E}_{s_i, r_i \sim E, a_i \sim \mu}[R_t | s_t, a_t]$. A base controller is denoted as $\mu_b(s)$ and can be queried for actions regarding arbitrary states. Our algorithm builds on DDPG [19], a standard actor-critic algorithm for continuous control which is often preferred in past works [5], [23] for its utilization of off-policy data. The actor network is denoted as $\mu(s|\theta^\mu)$, and the critic network is denoted as $Q(s, a|\theta^Q)$, where θ^μ and θ^Q are corresponding network weights. In the subsections below, we will describe our algorithm, DDPG with Base Controllers (DDPGwB, Alg. 1), including methods during exploration, value estimation and policy update to incorporate a base controller. And we will show how the algorithm scales from one base controller to an ensemble of base controllers.

A. Mixed Q-Control

Exploration is challenging in robotic tasks, and even more so when the reward is sparse. By utilizing a base controller to explore in the initial stage and linearly decaying its control ratio like in [30], we help the agent to experience a regular positive reward. Also, we help the agent to avoid unsafe states and unnecessary explorations. However, simply annealing the ratio poses a hard constraint and could result in abandoning the base controller too early or too late, when in the former case the agent might crumble due to insufficient guidance, and in the latter case the agent might not outperform the base controller since it cannot explore on its own. To address this, we propose a mechanism that allows the agent to flexibly adjust the control ratio. Specifically, the probability that an agent must execute the base controller's action, ϵ, is 1 at the start of learning, and decreases by δ every environment step. With $1 - \epsilon$ probability, the agent selects the best action as the critic predicts:

$$a_i = \underset{a \in \{\mu(s_i|\theta^\mu), \mu_b(s_i)\}}{\arg\max} Q(s_i, a|\theta^Q). \tag{1}$$

The intuition behind this is that, in Q-learning [31], the learned policy is $\arg\max$ over all possible actions given arbitrary state. Since the critic in DDPG performs the same role as the Q-network in DQN, it is natural to adopt the same fashion of choosing actions via $\arg\max$.

B. Base Controller Bootstrap

In the critic update step, DDPG, like many other RL algorithms, uses Bellman equation:

$$Q^*(s_i, a_i) = \mathbb{E}_{s_{i+1}, r_i \sim E}[r_i + \gamma \max_{a_{i+1}} Q^*(s_{i+1}, a_{i+1})] \tag{2}$$

to estimate the Q-value, where the asterisk denotes the optimal policy. Unlike Q-learning, DDPG deals with continuous actions and thus loses the ability of efficiently computing the maximum Q-value of next state. Hence, DDPG uses the Q-value of its current policy. In practice, target networks (indicated with apostrophe below) are used for stabilized learning, whose weights are slowly updated. We use y_i to represent the bootstrap target:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}). \tag{3}$$

This comes with certain caveats besides possible instability. Consider exploration with a base controller in sparse reward scenario, the only reward comes at the end of successfully completing the task. Upon learning the Q-value of that final

Algorithm 1: DDPG with Base Controllers (DDPGwB)

Input: actor network $\mu(s|\theta^\mu)$, critic network $Q(s,a|\theta^Q)$, target network μ' and Q', base controller $\mu_b(s)$, replay buffer R, Gaussian noise $\mathcal{N}$

Output: learned policy $\mu(s|\theta^\mu)$

Initialize $\epsilon = 1$;
for *episode*=1, M **do**
  Get initial observation s_1;
  for t=1, T **do**
    if $random < \epsilon$ **then**
      $a_t = \mu_b(s_t)$;
    else
      Get a_t according to Eq. 1;
    $a_t \leftarrow a_t + \mathcal{N}$;
    Execute action a_t, observe reward r_t and next state s_{t+1};
    Store transition (s_t, a_t, r_t, s_{t+1}) in R;
    $\epsilon \leftarrow \epsilon - \delta$;
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R;
    Get y_i according to Eq. 4;
    Update critic by minimizing the Bellman error;
    Update actor according to Eq. 6;
    Update the target networks:
    $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$,
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$;

step, it may not propagate instantaneously backwards since the action to reach that reward is very likely different from agent's own policy in the start. This would possibly contribute to slow-downs and inaccurate Q-value estimates. Hence we propose taking a maximum operation over the Q-values of agent's own policy and base controller's policy to compose the bootstrap target:

$$y_i = r_i + \gamma \max[Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}), Q'(s_{i+1}, \mu_b(s_{i+1})|\theta^{Q'})]. \quad (4)$$

This improvement, referred as base controller bootstrap, combined with mixed Q-control, can also be viewed as learning with a macro policy in the form of Eq. 1. And Eq. 4 can be obtained by plugging the macro policy into the standard bootstrap target form like Eq. 3.

C. Regularized Policy Update

The policy update in DDPG is done by back-propagating through the critic network:

$$\nabla_{\theta^\mu} J = \frac{1}{N} \sum_i \nabla_a Q(s,a|\theta^Q)|_{a=\mu(s|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu). \quad (5)$$

Due to inaccurate value estimates and a deterministic policy, such unbounded update tends to be very brittle as discussed in [32], as the actor will often try to exploit peaks in the Q-function and diverge. To relieve this, we apply behavior cloning w.r.t. the base controller as a regularization tool that guides the gradient update direction:

$$\nabla_{\theta^\mu} \mathcal{L}_a = -\nabla_{\theta^\mu} \mathcal{L}_{BC} + \lambda \nabla_{\theta^\mu} J, \quad (6)$$

where λ is a coefficient balancing two losses. Specifically, the behavior cloning gradient is imposed only on actions that are worse than those of the base controller as decided by the critic. Note that this does not violate the optimization goal of maximizing the accumulated reward. This kind of filtering is similar to [5], except in [5] BC loss was imposed on a demonstration buffer.

D. Ensemble of Base Controllers

For complex tasks, it is often that a good base controller may not exist, but one can find some weak controllers or controllers that each specializes in certain action primitives. It is then ideal to learn a policy from an ensemble of base controllers. To this end, our algorithm easily scales to multiple base controllers. Consider K base controllers: $\mu_{b_1}(s), \mu_{b_2}(s), ..., \mu_{b_K}(s)$, we define the ensemble action set:

$$\mathcal{B}_i = \{\mu_{b_1}(s_i), \mu_{b_2}(s_i), ..., \mu_{b_K}(s_i)\}, \quad (7)$$

the macro base controller action during exploration is then a randomly drawn sample from $\mathcal{B}_i$. And Eq. 1 becomes:

$$a_i = \underset{a \in \mathcal{B}_i \cup \{\mu(s_i|\theta^\mu)\}}{\arg\max} Q(s_i, a|\theta^Q). \quad (8)$$

The bootstrap target form in value learning now also extends to:

$$y_i = r_i + \gamma \max[Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}), \max_{a \in \mathcal{B}_{i+1}} Q'(s_{i+1}, a|\theta^{Q'})]. \quad (9)$$

Finally, $\mathcal{L}_{BC}$ in Eq. 6 becomes:

$$\mathcal{L}_{BC} = \frac{1}{N} \sum_i || \underset{a \in \mathcal{B}_i \cup \{\mu(s_i|\theta^\mu)\}}{\arg\max} Q(s_i, a|\theta^Q) - \mu(s_i|\theta^\mu)||^2. \quad (10)$$

At an algorithmic level, using an ensemble of base controllers helps the agent to avoid mistakes caused by a single controller during exploration, as in Eq. 8. It also helps the critic to further benefit from a diverse range of strategies, as reflected in the bootstrap target form. In turn, those learned values help the agent to shape a better policy as in Eq. 10. In this way, even if individual base controllers are weak or incomplete, a learned policy absorbs the strengths in each of them.

The complete pseudo-code is presented in Alg. 1. We unified exploration, value learning and policy learning into a single framework to take the best advantage of one or more base controllers. The core of our method is value learning that resembles the idea behind Q-learning. The critic is in charge of both exploration and policy update, while base controller bootstrap helps the critic to avoid suffering from a sub-optimal actor. The effect of actor brittleness is greatly

eased, as we will show that abandoning the use of a target actor network does not create drastic decay to our algorithm's performance, which is unusual for variants of DDPG.

IV. EXPERIMENTS

In this section, we first provide a description of the tasks we evaluate our algorithm on. Then we describe the neural network architectures, along with a straightforward template for designing base controllers. Finally, we give a quantitative evaluation of our algorithm in terms of final policies' success rates, learning efficiency and cost, performances on visual data based tasks and the ability to utilize an ensemble of base controllers. And we compare to previous methods.

A. Task Description

We use PyBullet [33] simulator and a KUKA robot arm for our experiments. In all tasks, the action dimension is 5, consisting of 3-dimensional (3-d) change of the end-effector position, 1-d change of end-effector yaw angle, and 1-d change of the gripper open angle. All actions have the range of $(-1, 1)$. The observation consists of proprioceptive data of the robot arm (8-d, including end-effector position, orientation, gripper open angle and the force on gripper fingers), and either state information of two objects (postions and orientations, 12-d) or images (binocular RGB images from two sides of the table, 128×128).

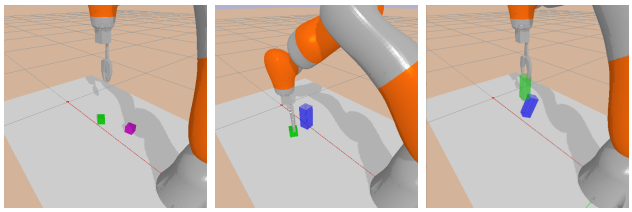
a) Stacking: This task involves two cubes initialized at random positions on the table, each with a side length of 5cm. The goal is to stack the green cube onto the purple cube.

b) Block-cup: This task involves a cube and a cup initialized at random positions on the table. The cube has a side length of 5cm. The cup's height is 15cm and has a square opening with a side length of 6cm. The goal is to place the green cube into the blue cup while not tipping the cup over.

c) Cup-cup: This task involves two cups initialized at random positions on the table. The green cup's opening has a side length of 5cm. The blue cup's opening has a side length of 6cm. Both cups' height is 15cm. The goal is to place the green cup into the blue cup while not tipping either cup over.

Screenshots of all tasks are presented in Fig. 1. In all tasks, we employ a truly sparse reward, that is, a reward of 1 if the goal is achieved, and a reward of 0 for other scenarios. An episode is terminated if it reaches 100 steps or the agent achieves the goal. In block-cup or cup-cup tasks, an episode is also terminated when a cup is tipped, and the reward is 0. All tasks are designed so that they reflect the challenges of sparse rewards and long task horizons, that is, the robot must accomplish a sequence of actions before receiving a proper reward signal. For example, the robot must first reach for the green cup, grasp the cup, then align the green cup with the blue cup, and finally place it into the blue cup.

We highlight the increasing difficulties in these tasks as multiple bottlenecks have to be overcome: First, the agent has to pick up the object. Second, it cannot tip the cup over when moving the object. Finally, it has to match the object precisely with the opening. The risk of tipping over the cup makes the tasks significantly harder and mimics a kind of safety requirements, e.g., when the robot manipulates fragile objects like glasses. A proper learning scheme should then avoid making those expensive mistakes. This is however, not considered in traditional setups where the object is placed into a fixture [34].



(a) Stacking (b) Block-cup (c) Cup-cup

Fig. 1. Screenshots of all tasks in different stages described in Sec. IV-A. The screenshot of the cup-cup task depicts a failed trial, where the blue cup is tipped over by the robot.

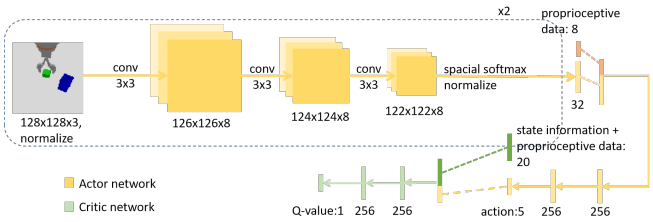


Fig. 2. Neural network architecture for an image-input actor and a state-input critic. The left-most image shows camera input from one side of the table.

B. Network Structure

a) State-input: The actor network receives 20-d input, and outputs 5-d action, with 2 hidden layers each having 256 units and ReLU activation. Activation of last layer is Tanh. The critic network receives 25-d input and outputs 1-d Q-value, with 2 hidden layers each having 256 units and ReLU activation. Activation of last layer is Sigmoid, which naturally fits the sparse-reward scenario. To this end, the bounded Q-value can also be viewed as a learned heuristic probability regarding task completion. It turns out that although DRL algorithms like SAC [21] or TD3 [32] perform better than DDPG on standard benchmarks, they exhibit inferior performances in the case of our long-horizon sparse-reward tasks due to the extra entropy reward introduced or underestimation biases. Hence we do not adopt them in building our algorithm.

b) Image-input: The normalized image first passes through a Convolutional Neural Network (CNN) with 3 convolution layers of 3×3 kernel size, stride 1, 8-channel output and ReLU activation, then passes through channel-wise spacial softmax [24] to get feature points (16-d). The binocular images share one CNN, and their output features are concatenated, resulting in a 32-d feature. This image feature concatenated with proprioceptive data is then passed through a fully connected network same as the state-input

Algorithm 2: Base Controller Template

Input: `function` MoveAndAlign: Execute $\mathbf{a} = \tanh\left(K_p(\mathbf{p}_{dist} - \mathbf{p}_{curr})\right)$

```
while goal not achieved do
    MoveAndAlign;
    if close enough then
        Close/Open Grip;
        if in grasp then
            Switch p_dist;
```

actor, except for different input dimensions. A state-input critic is coupled with an image-input actor during training as in [6]. Note that our method does not assume perfect state information and the use of state information here is for learning efficiency only. And the trained image-based policies do not rely on state information either. Fig. 2 outlines the entire neural network architecture.

C. Base Controllers

Given state information and proprioceptive data, we provide a simple controller template in Alg. 2 that covers a variety of tasks. $\mathbf{p}_{dist}$ represents the desired end-effector (object) position and yaw angle. $\mathbf{p}_{curr}$ represents the current end-effector (object) position and yaw angle. The controller first tries to reach and grasp an object, once it holds the object the controller will place it to the goal position. We use simple proportional controllers to calculate actions for the end-effector. And K_p is 5 and 2 for linear and angular actions respectively. Force on gripper fingers is used to tell whether there is a successful grasp. We find that following Alg. 2 produces decent success rates.

For an ensemble of base controllers, we break the sequentially-connected logic order and take out each action primitive as an independent controller, i.e., a MoveAndAlign controller, a controller that only does CloseGrip and a controller that only does OpenGrip. The action of macro base controller is randomly drawn from all 3 controllers. This mimics the scenario where no good single controller exists. And the agent must learn the best sequence of executing these actions in order to complete the task.

Note that the base controllers described here are for illustrative purposes. Since our method does not assume the nature of base controllers, they may take visual data as input and do not rely on state information. This is left for future work.

D. Training Details

We use Adam optimizer and a learning rate of 10^{-3} for both actor and critic. Exploration noise during training is Gaussian with 0.1 standard deviation and 0 mean. Discount factor γ is 0.99. Soft update parameter τ for target networks is 5×10^{-3}. Batch size is 256, and replay buffer size is 10^5. We set λ as 2×10^{-2} and δ as 2×10^{-5}.

TABLE I

SUCCESS RATES DURING FINAL EVALUATION

Method	Stacking	Block-cup	Cup-cup
base controller	92.2%	82.8%	49.9%
DDPGwB w/o BB	92.7%	90.5%	57.6%
DDPGwB w/o MQ_{arg}	93.1%	89.5%	59.4%
DDPGwB	**94.6%**	**92.1%**	**61.9%**

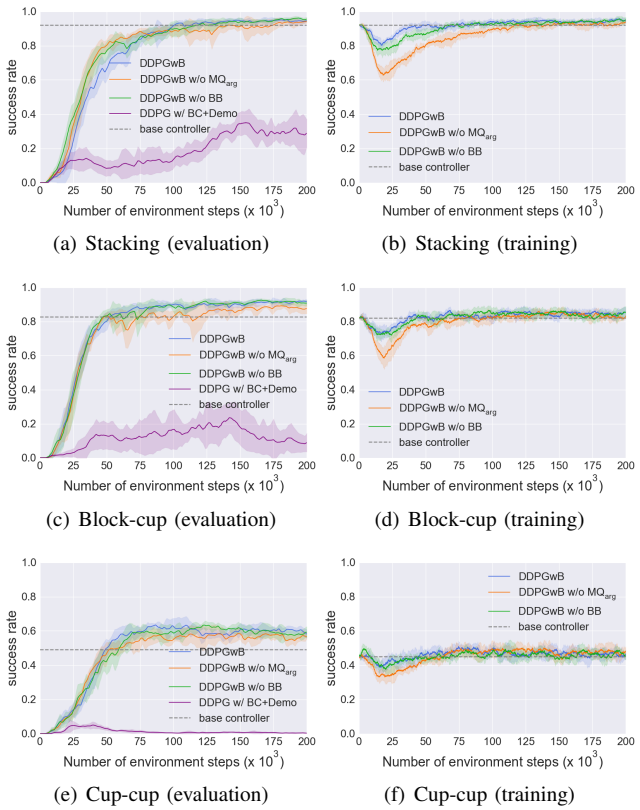
E. Results

Learning curves (success rates during evaluation) with state-input are summarized in Fig. 3(a, c, e). DDPGwB w/o BB represents ablation of base controller bootstrap, and DDPGwB w/o MQ_{arg} means we take out the $\arg\max$ part of mixed Q-control. In other words, ϵ is still 1 in the start, and decreases by δ every step, except that now for $1 - \epsilon$ probability the agent must choose its own action. Results show that taking out the BC loss from algorithm would result in a complete failure of learning, that is, a flat line of 0. So we omit it for reducing redundancy.

In addition, we compare to the method of learning from demonstrations [5], [23]. A total of 500 successful demonstrations are collected via the base controller to compose a demonstration buffer (5 times larger than the buffers in [5], [23]). The demonstration buffer is then sampled every gradient update, with a Q-filtered BC loss imposed on the policy, as per [5], and with a batch size N_D of 128. The learning curves are also presented (DDPG w/ BC+Demo).

a) Final performances compared to base controllers and ablations: We evaluate the performances of final learned policies, as summarized in Tab. I. In all tasks, the agent achieves a performance better than the base controller. Ablation study shows that final policies' performances degrades if base controller bootstrap or the $\arg\max$ part of mixed Q-control is taken out of the algorithm. Generally speaking, taking out the $\arg\max$ part of mixed Q-control will result in more instabilities, as reflected in Fig. 3. We omit comparison to learning from demonstrations as those agents never achieve a comparable performance with the base controller.

b) Learning efficiency and cost: In all tasks, our method requires only about 0.1 million environment steps or less to converge, while learning from demonstrations struggles to converge in limited environment steps. In previous works of learning sparse-reward tasks from demonstrations, [5] and [8] both learned a much easier pick-and-place task: the robot arm had to pick up a block on the table and move it to a position in space, where neither orientation nor object-interaction is considered. According to the results, [5] used about 1 million environment steps to converge, while [8] used about 0.5 million environment steps to converge. [8] also learned a slightly easier version of the stacking task where orientation is not considered, and it took 14 million environment steps to learn a policy with a success rate around 80%. According to our results, as the difficulty level of the task increases (more bottlenecks), it becomes harder for demonstration learners to find a sensible policy, and performances even start to degrade as the training progresses. For the cup-cup task, the final



(a) Stacking (evaluation) (b) Stacking (training)

(c) Block-cup (evaluation) (d) Block-cup (training)

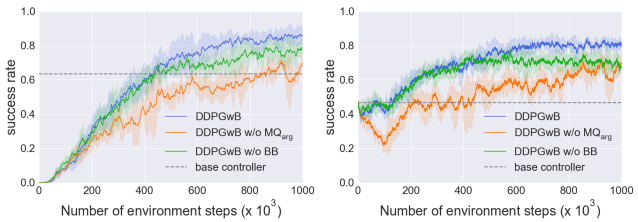
(e) Cup-cup (evaluation) (f) Cup-cup (training)

Fig. 3. (a, c, e) Learning curves (success rates during evaluation) of three tasks. (b, d, f) Success rates in actual training of three tasks. Solid lines represent the mean, and shaded areas represent standard deviation. Number of environment steps is used as x-axis label to convey sample complexity. Curves are averaged from 6 runs with different random seeds and smoothed for clarity. The performance of base controller is denoted by the horizontal dashed line.

performance is almost down to zero.

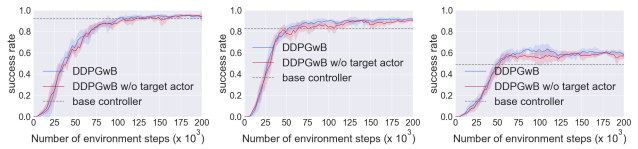
To better characterize the cost of learning, we present the success rates during training in Fig. 3(b, d, f), which are the performances the agents hold while actually interacting with the environment. Note that all policies' performances are naturally lower than those during evaluation because of Gaussian noise. Results show that our method learns at a stable success rate throughout the training process. And the base controller helps prevent the agent from making mistakes like tipping over the cup in the initial stage. Without the $\arg\max$ part of mixed Q-control, the algorithm is not able to flexibly decide control ratio, hence creating a performance drop in the initial stage. Taking out base controller bootstrap also introduces a performance drop, which could be attributed to inaccurate Q-value estimates. Again, we omit discussion of learning from demonstrations since they start learning with a success rate of 0.

c) Learn from images: We run a set of experiments that use binocular RGB images as actor network's input instead of state information. We train the agent for 0.4 million environment steps. The final learned image-based policies achieve success rates of 93.0%, 88.4%, and 58.2% respectively for stacking, block-cup and cup-cup task, all of which exceed the performances of base controllers, proving that our method also applies to image-based tasks.



(a) (b)

Fig. 4. (a) Learning curves when using an ensemble of base controllers. (b) Success rates during training when using an ensemble of base controllers.



(a) Stacking (b) Block-cup (c) Cup-cup

Fig. 5. Effect of abandoning the use of target actor networks.

d) Learn with an ensemble of base controllers: The macro base controller composed of an ensemble of controllers as described in Sec. IV-C is naturally much weaker due to randomness in the actions. For the stacking task, the success rate of macro base controller is 63.5% (no Gaussian noise) and 46.7% (in the presence of Gaussian noise) respectively. In Fig. 4, we show the learning curves and training success rates of DDPGwB as well as ablations under this setting. The full algorithm is the only method that achieves a performance comparable with that under the single base controller setting (90%+), reaffirming the results in Sec. IV-E.a and Sec. IV-E.b. Also, it proves the validity of learning from an ensemble of base controllers.

e) Abandon the target actor: We further validate our algorithm by abandoning the use of a target actor network, which is an unusual ablation since most algorithm variants of DDPG need both target actor and target critic networks for stabilized learning. As shown in Fig. 5, abandoning the use of a target actor network does not create drastic decay to our algorithm's performance, and policies better than base controllers are still stably learned, which highlights that our algorithm greatly eases the effect of actor brittleness and differentiates our algorithm from DDPG and its variants.

V. CONCLUSIONS

In this paper we propose DDPGwB, an algorithm that utilizes one or more base controllers to efficiently learn long-horizon sparse-reward robotic tasks, with the learned state-based or image-based policies exceeding the performances of these base controllers. One possible direction of future work is trying to automatically synthesize base controllers for tasks requiring multi-step planning. We also believe that our algorithm has the potential of being applied to a range of tasks where learning complex policies could benefit from one or more traditional policies.

REFERENCES

[1] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[2] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[3] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *arXiv preprint arXiv:1806.10293*, 2018.

[4] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[5] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6292–6299.

[6] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, *et al.*, "Reinforcement and imitation learning for diverse visuomotor skills," *arXiv preprint arXiv:1802.09564*, 2018.

[7] L. Hermann, M. Argus, A. Eitel, A. Amiranashvili, W. Burgard, and T. Brox, "Adaptive curriculum generation from demonstrations for sim-to-real visuomotor control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6498–6505.

[8] Y. Ding, C. Florensa, P. Abbeel, and M. Phielipp, "Goal-conditioned imitation learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 298–15 309.

[9] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.

[10] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[11] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2015.

[12] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[13] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer, "Ensembledagger: A bayesian approach to safe imitation learning," *arXiv preprint arXiv:1807.08364*, 2018.

[14] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, 2000, p. 2.

[15] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in neural information processing systems*, 2016, pp. 4565–4573.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[17] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," *arXiv preprint arXiv:1511.03791*, 2015.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[20] M. Vecerik, O. Sushkov, D. Barker, T. Rothörl, T. Hester, and J. Scholz, "A practical approach to insertion with variable socket position using deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 754–760.

[21] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[22] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in neural information processing systems*, 2017, pp. 5048–5058.

[23] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[24] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.

[25] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, A. Tamar, and P. Abbeel, "Reinforcement learning on variable impedance controller for high-precision robotic assembly," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3080–3087.

[26] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6023–6029.

[27] T. Davchev, K. S. Luck, M. Burke, F. Meier, S. Schaal, and S. Ramamoorthy, "Residual learning from demonstration," *arXiv preprint arXiv:2008.07682*, 2020.

[28] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," *arXiv preprint arXiv:1906.05841*, 2019.

[29] L. Xie, S. Wang, S. Rosa, A. Markham, and N. Trigoni, "Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6276–6283.

[30] M. Xin, Y. Gao, T. Mou, and J. Ye, "Online hybrid learning to speed up deep reinforcement learning method for commercial aircraft control," in *2019 3rd International Symposium on Autonomous Systems (ISAS)*. IEEE, 2019, pp. 305–310.

[31] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[32] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[33] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," *GitHub repository*, 2016.

[34] I. Akinola, J. Varley, and D. Kalashnikov, "Learning precise 3d manipulation from multiple uncalibrated cameras," *arXiv preprint arXiv:2002.09107*, 2020.