International Journal of Control, Automation, and Systems 20(10) (2022) 3296-3311
http://dx.doi.org/10.1007/s12555-021-0511-4
ISSN:1598-6446 eISSN:2005-4092
http://www.springer.com/12555

Hierarchical End-to-end Control Policy for Multi-degree-of-freedom Manipulators

Cheol-Hui Min and Jae-Bok Song*

Abstract: In recent years, several control policies for a multi-degree-of-freedom (DOF) manipulator using deep reinforcement learning have been proposed. To avoid complexity, previous studies have applied a number of constraints on the high-dimensional state-action space, thus hindering generalized policy function learning. In this study, the control problem is addressed by in-troducing a hierarchical reinforcement learning method that can learn the end-to-end control policy of a multi-DOF manipula-tor without any constraints on the state-action space. The proposed method learns hierarchical policy using two off-policy methods. Using human demonstration data and a newly proposed data-correction method, controlling the multi-DOF manipu-lator in an end-to-end manner is shown to outperform the non-hierarchical deep reinforcement learning methods.

Keywords: Deep reinforcement learning, demonstration-based learning, end-to-end robot control, hierarchical reinforcement learning.

1. INTRODUCTION

Reinforcement learning (RL)-based robot control is typically performed through policy search methods that learn a policy function by receiving measured robot information and outputting a robot control command. The control function of RL finds the innate rule of control system by itself, while conventional Fuzzy-sytem based control methods [1–3] requires control rules. Further, studies investigating generalized robot control policies using function approximation with deep neural networks are becoming mainstream. However, three significant difficulties still exist in learning robot control policies through RL. Firstly, the state-action space of a robot is high-dimensional and continuous [4]. Secondly, such learning requires a substantial amount of data, but it is difficult to obtain noiseless data using real robots. Thirdly, as a robot agent performs workspace exploration based on stochastic trajectories to identify an optimal policy, there is a risk of damage to the robot and the surrounding environment during operation [5].

In attempts to address these difficulties, many studies have proposed a number of methods, including the discretization of state-action space [6] and the design of a poly function specified for an individual task or learning from the demonstrated trajectory of a human [7]. Furthermore, employing prior knowledge of tasks, like human demonstration [8] and model-based guidance [9,10] have also been shown to be promising methods to solve the difficulties.

To be specific, many works have addressed the complexity of the state-action space with a structural composition of the policy network. One stream of those studies is to design the neural networks of the agent in a systematic method. A neural programming that learns to decompose demonstrated robotic tasks into hierarchy in a supervised way was proposed in [11]. A compositional policy architecture, where a task-conditioned high-level policy infers an embedded representation that chooses the combination of sub policies was proposed to solve a complex task in [12]. A compositional reasoning of actions with dynamic assembly of module networks was performed in [13]. The structured decomposition concept was applied to one-shot imitation learning in [14]. While all of these studies can be categorized as hierarchical abstract machine [15–17], which requires an expert demonstration or predefined task primitives [18].

Another line of work focuses on the hierarchical policy learning in either the model-free or model-based manner, with hierarchical manipulation of the resolution of the agent's trajectory [19–24]. The underlying idea, which has been the basis of many recent architectures, is the hierarchical temporal abstraction of the trajectory that the agent experiences. Such abstraction leads to the design of

Manuscript received June 19, 2021; revised November 27, 2021 and January 16, 2022; accepted February 9, 2022. Recommended by Associate Editor Jiuxiang Dong under the direction of Editor Guang-Hong Yang. This research was supported by the MOTIE under the Industrial Foundation Technology Development Program supervised by the KEIT (No. 20008613).

Cheol-Hui Min and Jae-Bok Song are with the School of Mechanical Engineering, Korea University, 145, Anam-ro, Seongbuk-gu, Seoul, Korea (e-mails: {mch5048, jbsong}@korea.ac.kr).
* Corresponding author.

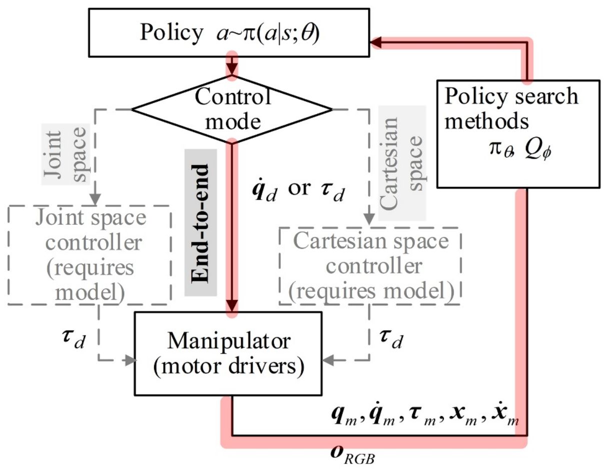

Policy $a \sim \pi(a|s;\theta)$
Control mode
Joint space
Cartesian space
Policy search methods π_θ Q_ϕ
End-to-end
$\dot{q}_d$ or τ_d
Joint space controller (requires model)
Cartesian space controller (requires model)
τ_d
τ_d
Manipulator (motor drivers)
$q_m, \dot{q}_m, \tau_m, x_m, \dot{x}_m$
o_{RGB}

Fig. 1. End-to-end control policy of a manipulator.

higher-level policies, allowing more abstract transition dynamics of the environment. For example, Vezhnevets *et al.* [25] proposes an on-policy hierarchical RL agent, whose low-level agent is incentivized with the cosine similarity reward to the goal. Co-Reyes *et al.* [26] adopts a representation learning to learn latent representation of a trajectory of the agent, to run the model predictive control (MPC) agent. Levy *et al.* [27] utilizes the hindsight experience replay [28] to exploit the data-efficiency of off-policy learning. Similar to this, Jiang *et al.* [30] proposes an off-policy hierarchical deep RL method with off-policy correction to increase the sample efficiency of the agent. In addition, LI *et al.* [31] proposed a language-based abstraction of the goal state inferred from the high-level policy, and Jain et al. [32] adopted the hierarchical agent for navigation in complex indoor environments. While these methods are not validated on real-world robots, several works have successfully learned real-world robot tasks. In [33,34], the locomotion policy was trained for quadrupled robots whose state-action space is more complex than robot arms. In addition to this, Levy *et al.* [35] trained a mixture of policies conditioned on a specific task, alternating between non-parametric policy search and supervision of parameterized policy in a hierarchical manner, resulting in learning complex robot manipulation tasks in real world.

Our work also investigates the hierarchical reinforcement learning approach to tackle the complexity of state-action space. By exploiting both the deterministic and stochastic policy, we expand [31] in theory. In addition, other techniques are adopted to train an end-to-end control policy of a robot to solve various manipulation tasks, as depicted in Fig. 1.

In this study, our method, hierarchical hard soft actor-critic (H-HSAC) algorithm, learns a policy that can control a multi-DOF manipulator through hierarchical RL. H-HSAC represents high-dimensional state-action space using a hierarchical structure, which allows for the learning problem to be defined without limitation ns in terms of dimensional complexity. Accordingly, control policies that are more generalized than previously proposed methods, such as ones that involve confining action space to be Cartesian space or using analytic information adoption like Euclidean distance between a robot's end-effector and an object while training, can be learned. Furthermore, as H-HSAC learns policy in an off-policy manner, human demonstration data can be used to improve its sample efficiency.

The main contribution of the proposed H-HSAC scheme is the newly proposed off-policy correction method for the hierarchical RL whose low-level policy has stochasticity. The resulting policy can estimate the robot control command from the information obtained from the camera and joint encoder, which are unrelated to the kinematics and dynamics of a robot. In addition, we propose a method to adopt human demonstration as a hierarchical data to increase the sample efficiency in training both high-level and low-level policies. The performance of the proposed method on various robot tasks are demonstrated by solving it in an end-to-end manner. It is shown that the proposed approach outperforms the state-of-the art hierarchical RL agents in [31,35].

The rest of the paper is organized as follows: Section 2 introduces the preliminaries of RL and the two algorithms comprising the proposed hierarchical policy: twin-delayed deep deterministic policy gradient (TD3) [36] and soft actor-critic (SAC) [37]. Next, the structure of H-HSAC is de-scribed in Section 3 along with a method to improve its off-policy learning. The leveraging human demonstration and the real-world transfer learning of simulation are discussed in Sections 4 and 5, respectively. Finally, Section 6 describes the experimental results of H-HSAC.

2. BACKGROUND

2.1. Reinforcement learning

The reinforcement learning framework can generally be divided into agents and environments. For each time step t, the agent first observes a state s_t from the environment and subsequently performs an action a_t determined by a policy π, which is conditioned on s_t. The agent then observes the next state s_{t+1}, which is a result of the transition through the probabilistic dynamics of the environment $p(s_{t+1} \mid s_t, a_t)$, and this transition is quantitatively evaluated by a scalar reward t_t that is obtained from the reward function R and is defined as follows:

$$r_t = R(s_t, a_t, s_{t+1}). \tag{1}$$

The function R in (1) describes the desired goal of each task, and is generated over the whole time sequence. Consequently, the agent accumulates a sequence of data $\tau = [s_0, a_0, r_0, ..., s_T]$, which is referred to as the trajectory.

The objective of reinforcement learning is to identify an optimal policy π^* that maximizes the return $G_t = \Sigma\gamma^i/r_i$, which is a cumulative sum of the reward r_t discounted at every step by γ. To find π^*, we first define a policy function π_θ parameterized by θ and then formulate the corresponding objective function $J_\pi(\theta)$, which optimizes θ in the direction of maximum G_t. During optimization, θ is updated with the gradient of the objective function J, $\nabla_\theta J$, formulated as follows:

$$\theta_{k+1} = \theta_k + \nabla_\theta J_\pi(\theta_k), \tag{2}$$

where $J_\pi(\theta) = E[G_t]$ for each gradient step k. The updated policy of (2) is called the policy gradient method, and the computation of the gradient is affected by the properties of the policy.

2.2. Goal-conditioned reinforcement learning

The proposed H-HSAC algorithm is a goal-conditioned, hierarchical reinforcement learning method [38]. Its low-level policy function estimates primitive actions to achieve the goals estaimated by high-level policy function that achieves a single learning objective, where in return maximization is implicitly set by the reward function $R(s_t, a_t, s_{t+1})$. By contrast, in goal-conditioned RL, the policy function $\pi_\theta(a_t \mid s_t, g_t)$ considers both the state s_t and the goal g_t that an agent should achieve at every step. Thus, the reward function $R_g(s_t, a_t, s_{t+1}, g_t)$ is dependent on g_t, meaning that the agent behaves differently based on the given g_t as it is optimized with the objective function described by

$$J = E_g\left[E_{\pi_\theta(a|s,g)}\left[\Sigma_t R_g(s_t, a_t, s_{t+1}, g_t)\right]\right]. \tag{3}$$

2.3. High-level policy for goal estimation: TD3

The H-HSAC algorithm has a hierarchical RL structure, and its high-level policy function is learned based on the twin delayed deep deterministic policy gradient (DDPG) algorithm [36]. The twin-delayed DDPG (TD3) is based on the deep deterministic policy gradient algorithm [39] in which the return G_t in (2) is approximated by the Q function. The policy function μ_θ from the deterministic policy gradient and the Q function Q_ϕ adopted from a deep Q-network (DQN) [40] are learned alternately. The updates are implemented in an off-policy manner using data samples obtained from a replay buffer D. The deterministic policy function μ_θ is optimized based on the following objective function:

$$J(\theta) = E_D\left[Q_\phi(s, \mu_\theta(s))\right]. \tag{4}$$

DDPG trains μ_θ by taking the gradient of (4) and updating the neural network θ similar to (2). Further, Q_ϕ is optimized through the loss function given by

$$L(\phi) = E_D\left[\{Q_\phi(s,a) - y\}^2\right], \tag{5}$$

where y represents the target function described by, $y = r + \gamma Q_\phi^{\text{targ}}(s', \mu_\theta^{\text{targ}}(s'))$, where Q_ϕ^{targ} and μ_θ^{targ} are the target networks used to stabilize the learning procedure in a manner similar to DQN. The TD3 algorithm, along with the DDPG, further stabilizes learning by addressing the overestimation issue of the Q-values in DDPG. To begin, the target policy network in (5) is modified as follows:

$$a'(s') = \mu_\theta^{\text{targ}}(s') + \text{clip}(\varepsilon, -c, c), \text{ where } \varepsilon \sim N(0, \sigma), \tag{6}$$

where ε is the random noise following a normal distribution N added in the process of estimating Q^{targ} and the clip() operator truncates ε in the range $(-c, c)$. Equation (6) prevents the maximum Q^{targ} from being estimated at all times. Furthermore, TD3 trains the two separate the Q-function networks of ϕ_1 and ϕ_2 with the following loss function of Q-functions:

$$L(\phi_i, \mathcal{D})_{i=1,2} = E_D\left[\left(Q_{\phi_i}(s, a) - y\right)^2\right], \tag{7}$$

where $y = r + \gamma\min_{i=1,2} Q_{\phi_i}^{\text{targ}}(s', a'(s'))$.

For the function approximation error in the neural networks to yield different estimations from the same input, the $\min Q_{\phi_i}$ operation in (7) reduces the overestimation error of the Q values by considering the smaller of the Q value estimates of Q_{ϕ_1} and Q_{ϕ_2}. Finally, TD3 performs a delayed policy function update, compared to Q functions, which also promotes the robustness of the algorithm. In H-HSAC, TD3 behaves as a high-level policy that estimates a deterministic goal for the low-level policy.

2.4. Low-level policy for primitive action: SAC

The low-level policy of H-HSAC is learned through the soft actor-critic (SAC) algorithm [37]. To begin with, this reparameterizes the stochastic policy function π_θ as a Gaussian policy $f_\theta(s, \xi)$ (i.e., $\pi_\theta = f_theta(s, \xi)$). Further, SAC aims to maximize both the return and entropy $H(\pi_\theta)$ that represents the randomness of π_θ. This behavior is learned by considering the objective function of the policy network θ in (8), which helps the agent regulate the trade-off between exploitation and exploration.

$$J(\theta) = E_D\left[Q_\phi(s, f_\theta(s, \xi)) + \alpha H(\pi_\theta)\right]. \tag{8}$$

In (8), $H(\pi_\theta)$ equals $-\log \pi_\theta$ and Q_ϕ is the approximated return using the Q-function network ϕ. Like DDPG, SAC also alternates between updating θ and ϕ. As a result, ϕ is optimized as follows:

$$L(\phi) = E_D\left[(Q_\phi(s,a) - y)^2\right], \tag{9}$$

where $y = r + \gamma\left\{Q_\phi^{targ}(s', f_\theta(s', \xi)) + \alpha H(\pi_\theta)\right\}$.
Equation (9) is a loss function that learns Q_ϕ to fit the target function y augmented with the entropy $H(\pi_\theta)$.

Therefore, SAC learns a policy that approximately soft-maximizes the Q values by optimizing the two objective functions of (8) and (9) in an actor-critic, off-policy manner. Furthermore, the double Q-network method of (7) was borrowed during implementation for the purpose of stabilization. Further, the variable α for adjusting the entropy maximization degree can also be learned using the following objective function

$$J(\alpha) = E_a\left[-\alpha \log \pi_\theta(a|s) - \alpha H_0\right], \tag{10}$$

where H_0 represents the entropy, which should be kept to a minimum. In summary, the SAC algorithm adjusts the trade-off between exploration and exploitation in the learning process so that the agent can better find meaningful reward signals in a complex state-action space.

3. HIERARCHICAL-HARD-SOFT ACTOR-CRITIC (H-HSAC)

This section proposes the hierarchical-hard-soft actor-critic (H-HSAC) algorithm, which is a hierarchical and goal-conditioned RL algorithm. H-HSAC consists of the high-level policy μ_θ^{hi} learned with TD3, which is based on (hard-) $\max_a Q$, and the low-level policy π_θ^{lo} learned with soft-$\max_a Q$-based SAC, as described in Section 2.

3.1. Hierarchical architecture for goal-conditioned policy

In this study, a goal-conditioned RL is performed, where μ_θ^{hi} estimated a goal g that π_θ^{lo} should achieve at every c step, as illustrated in Fig. 2. At this point, H-HSAC borrows the structure of subgoal-based deep RL algorithms [29,31]. In addition, only the partial state $s_{part,t}$ and observation o_t are fed to the high-level and the low-level policies of H-HSAC, respectively, despite the environment, at

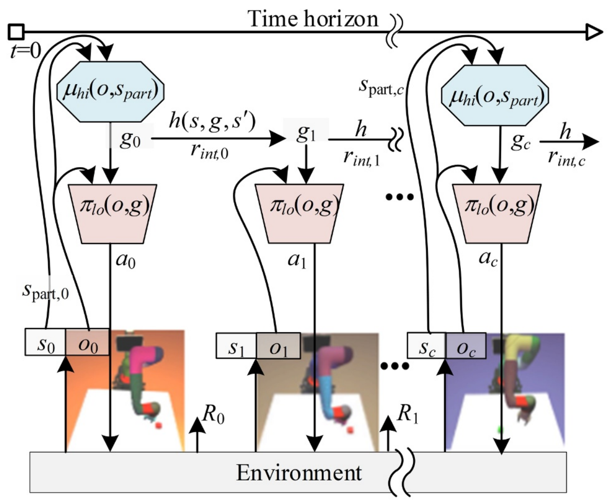


Fig. 2. Hierarchical interaction of the policy in H-HSAC with the environment.

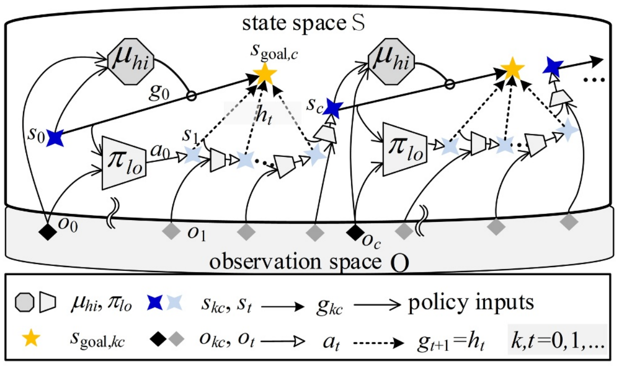


Fig. 3. Goal-conditioned interaction of hierarchical policy in the state space.

the time of training, providing the full state s_t. This is done intentionally to train an end-to-end control policy that can estimate the control command of a manipulator, a_t, independently of $s_{part,t}$ and o_t. The partial state $s_{part,t}$ denotes a subset of the full state, and it consists of information obtained without using the robot models (i.e., kinematics and dynamics) and the pose of target object. Further, the observation data indicates the raw visual information of the environment.

To describe the proposed method in detail, the agent-environment interaction, depicted in Fig. 2, is projected into the state space of the learning problem, as shown in Fig. 3. In this sense, the agent-environment interaction of a hierarchical policy is carried out using the following procedure. During training, the agent begins by acquiring the initial full state s_0 and observation o_0 from the environment during each episode.

Subsequently, the high-level policy $\mu_\theta^{hi}(o,\, s_{part})$ takes the partial state $s_{part,0}$ and the observation o_0, and then estimates a subgoal g_0 that amounts to the difference between the partial goal states, which the agent visit after c steps, and its current partial state. Next, the low-level policy $\pi_\theta^{lo}(a \mid o,\, g)$ estimates an action a_0 from g_0 and o_0, and the agent then applies a_0 to the environment. Finally, the agent observes the next state s_1 and the observation o_1 that follows the environment dynamics $p(s_{t+1} \mid s_t,\, a_t)$ and the observation dynamics $p(o_t \mid s_t)$. Meanwhile, μ_θ^{hi} should estimate g_t at every c step in the proposed method, where g_t should be adjusted over the intervening time steps. The adjustment is conducted through a goal transition function h which is defined by

$$g_{t+1} = h\left(s_{part,t},\, g_t,\, s_{part,t+1}\right) = s_{part,t} + g_t - s_{part,t+1}. \tag{11}$$

The transition function in (11) is depicted in Fig. 3 as a sequence of black dashed lines, and $\underline{g}_t$ points to $s_{goal,kc}$ at every time step in between the goal estimations. Further, an intrinsic reward [41] to train the low-level policy π_θ^{lo}

using h is defined as follows:

$$r_{int,t} = -\|s_{part,t} + g_t - s_{part,t+1}\|_2 = -\|h\|_2. \quad (12)$$

As $r_{int,t}$ in (12) equals the negative Euclidian distance of the vector g_t, a higher value denotes increased satisfaction of g_t. The low-level policy π_θ^{lo} in H-HSAC is then improved to satisfy the subgoal g_t as it is trained on $r_{int,t}$. Simultaneously, the high-level policy μ_θ^{hi} is trained on c step cumulative reward $\Sigma R_{t:t+c-1}$, where R_t is the reward given by the environment. In this manner, μ_θ^{hi} is improved toward achieving the episodic objective.

3.2. Assymetric actor-critic method

In this study, an end-to-end control policy of a robot is proposed. To be specific, the policy outputs a robot's joint velocity and gripper commands by observing an RGB image and partial state that is irrelevant to the kinematics and the dynamics of a robot. This form of training has certain advantages and disadvantages. One advantage is that the well-trained policy can be generalized due to the unconstrained state-action space. The disadvantage is that the training becomes challenging when the full state cannot be exploited. To address this, an asymmetric actor-critic [42] architecture is adopted for both the high- and low-level actor-critics as the value function for policy evaluation is necessary only during the training.

Fig. 4 illustrates the input-output relationship between the high- and low-level policy functions μ^{hi} and π^{lo}, respectively, as well as the value functions Q^{hi} and Q^{lo}. As shown in the figure, μ^{hi} and π^{lo} only require the observation o and the partial state s_{part}, while Q^{hi} and Q^{lo} receive a full state s, where s_{part} is augmented with the auxiliary state s_{aux}. This asymmetric design leads to the policy function being learned indirectly on the full state s received by the value function although it only receives only s_{part} and o. Consequently, the policy function can better estimate the action a in an end-to-end manner within the test time by simply observing s_{part} and o.

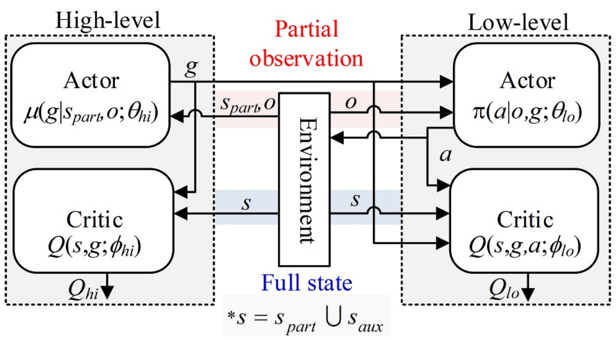


Fig. 4. Overall asymmetric actor-critic architecture of H-HSAC.

3.3. Off-policy correction for the reparameterized high-level policy

The off-policy characteristic of H-HSAC, induced by employing a replay buffer D, allows us to consistently utilize training samples in a sample-efficient way. Nevertheless, in hierarchical RL when the high-level policy μ_θ^{hi} is trained in an off-policy manner, the non-stationary dynamics issue occurs [43]. In H-HSAC, this is attributed to the fact that because the dynamics of high-level policy $p^{hi}(s_{t+c} \mid s_t, g_t; \theta_{lo})$ is dependent on the c step state-action trajectory of low-level policy $p(s_t, o_t, a_t, ..., s_c, o_c \mid \theta_{lo})$, as described below.

$$\begin{aligned} &p^{hi}(s_{t+c}|s_t, g_t; \theta_{lo}) \\ &\propto p(s_t, o_t, a_t, \ldots, s_c, o_c \mid \theta_{lo}) \\ &= p(s_t) \prod_{i=t}^{t+c-1} \pi_\theta^{lo}(a_i|o_i, g_i) p(o_i|s_i) p(s_{i+1}|s_i, a_i). \quad (13) \end{aligned}$$

The connection between p^{hi} and π_θ^{lo} in (13) implies that p^{hi} is transient as the low-level policy network θ_{lo} is updated (i.e., $p^{hi}(s_{t+c} \mid s_t, g_t; \theta_{lo,old}) \neq p^{hi}(s_{t+c} \mid s_t, g_t; \theta_{lo,new})$). To address this issue, we propose an off-policy correction method [29] that reflects the stochastic nature of the low-level policy of H-HSAC. During the learning process, this method corrects the data $\tau_{hi,old}$ sampled from the replay buffer of high-level policy D_{hi}, which has been previously generated along the dynamics $p^{hi}(s_{t+c} \mid s_t, g_t; \theta_{lo,old})$, represented as sky-blue dashed lines in Fig. 5(a). Accordingly, $\tau_{hi,old}$ is corrected to $\tau_{hi,new}$, which satisfies the current dynamics $p(s_{t+c} \mid s_t, g_{t,corr}; \theta_{lo,new})$, as depicted by the blue dashed line in Fig. 5(b). The corrected subgoal $g_{t,corr}$ that fits the current dynamics is chosen with the maximum likelihood estimation (MLE) on a set of subgoal candidates C. The MLE of the subgoal is found over c step length data $a_{t:t+c-1}$ and $o_{t:t+c-1}$, and the estimator is the current policy $\pi_{\theta,new}^{lo}$; it can therefore be formulated by

$$g_{t,corr} = \arg\max_{\tilde{g}_t \in C} \log \pi_{\theta,new}^{lo}(a_{t:t+c-1}|o_{t:t+c-1}, \tilde{g}_{t:t+c-1}). \quad (14)$$

C in (14) consists of goal candidates g_{cands} as follows:

$$\begin{aligned} C = \{&g_{t,\theta old}, s_{t+c} - s_t, \\ &g_{rand,1} \sim N_{corr}(s_{t+c} - s_t, (\alpha \cdot \sigma_{\theta,\text{lo}})^2 I), \ldots, g_{rand,n}\}. \end{aligned} \quad (15)$$

The elements of C are illustrated in Fig. 5 where the elements $g_{t,old}$, $s_{c+t} - s_t$, and n sampled goals g_{rands} from the distribution $N(s_{c+t} - s_t, (\alpha \cdot \sigma_\theta)^2)$. $\log \pi_{\theta,new}^{lo}$ in (14), which is a diagonal Gaussian with mean μ_θ^{lo} and standard deviation σ_θ^{lo}, where action a is k-dimensional, and is calculated as follows:

$$\log \pi_{\theta,new}^{lo}(a_{t:t+c-1} \mid o_{t:t+c-1}, \tilde{g}_{t:t+c-1})$$

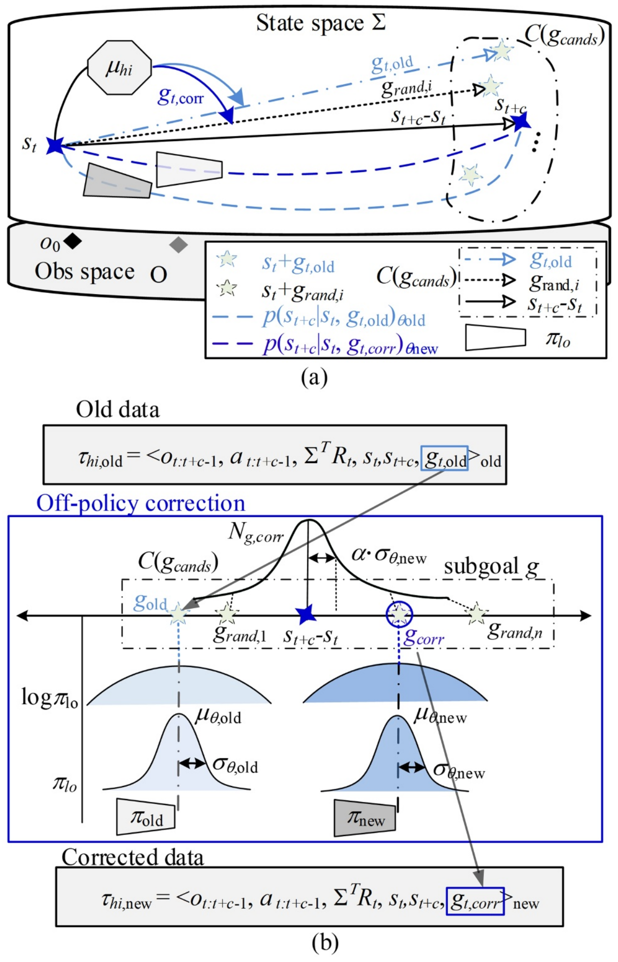


Fig. 5. Off-policy correction of H-HSAC.

$$= -\frac{1}{2}\left[\Sigma_{i=t}^{t+c-1}\Sigma_{j=1}^{k}\left\{\frac{(a_{i,j}-\mu_{i,j})^2}{\sigma_{i,j}^2}+2\log\sigma_{i,j}\right\}\right.$$
$$\left.+kc\log 2\pi\right]. \tag{16}$$

Fig. 5(a) shows the distribution of goal candidates in the state space S. The off-policy correction aims to correct $g_{t,old}$ to fit the current low-level policy's dynamics (blue dashed line) by selecting a new goal that maximizes the current policy's probability; this procedure is shown in Fig. 5(b), which illustrates that $g_{t,old}$, which has the maximum likelihood under $\pi^{lo}_{\theta,old}$, is being corrected with the new subgoal $g_{t,corr}$ that has the maximum likelihood under $\pi^{lo}_{\theta,new}$. The proposed method, in contrast to [29], reflects the learnability of the stochasticity of low-level policy σ^{lo}_{θ} and α in composing the goal candidates. The proposed off-policy correction reflects the stochastic characteristics of low-level policy better than compared to the one proposed in [29], which is formulated on the deterministic policy.

3.4. Learning procedure of H-HSAC algorithm

Algorithm 1 outlines the overall interaction and learning of H-HSAC. First the agent accumulates the trajectories τ_{hi} and τ_{lo} in the replay buffers for each of the high- and low-level policies, as shown in Fig. 3. Next, in the learning phase, the low-level actor-critics θ_{lo}, $\phi_{lo,1}$, and $\phi_{lo,2}$ are trained based on 1-step temporal difference (TD) learning, whereas the high-level actor-critics θ_{hi}, $\phi_{hi,1}$, and $\phi_{hi,2}$ are trained based on c step TD learning using the off-policy correction described in Fig. 5. By using this scheme, H-HSAC learns the control policy of a robot with-

Algorithm 1: Hierarchical-hard soft actor-critic (H-HSAC).

```
Initialize high-level actor-critics $\mu^{hi}_{\theta}$, $Q^{hi}_{\phi 1}$, $Q^{hi}_{\phi 2}$ and their target networks.
Initialize low-level actor-critics $\mu^{lo}_{\theta}$, $Q^{lo}_{\phi 1}$, $Q^{lo}_{\phi 2}$ and their target networks.
Initialize $\alpha$, $H_0$, $c$, goal candidate number $n_{cand}$, $\lambda$ (learning rate), $v$(Polyak averaging rate).
Initialize high-level and low-level buffers $D_{hi}$, $D_{lo}$.
for each episode do
    Observe $s_t$, $o_t$ $(t=0)$.
    $g_t \sim \mu_{hi}(s_{part,t}, o_t)+\sigma(0,\mathbf{I})$
    for each env. step $t$ do
        if $t$ $(>0)$ mod $c$ then
            Store< $[o_{t:t+c-1}]$, $[a_{t:t+c-1}]$, $\Sigma_\tau R_t$, $s_t$, $s_{t+c}$, $g_{t,old}$ $>_{hi}$ in $D_{hi}$.
            $g_t \sim \mu_{hi}(s_{part,t}, o_t)+\sigma(0,\boldsymbol{I})$.
        end if
        Execute $a_t \sim \pi_{lo}(a \mid o_t, g_t)$ and get $Rt$, $s_{t+1}$, $o_{t+1}$
        Append< $[o_t]$, $[a_t]$, $R_t$, $s_t$, $\cdot$, $g_t$ $>_{hi}$.
        $g_{t+1} = h(s_{part,t}, s_{part,t+1}, g_t)$
        Compute intrinsic reward $t_{int,t} = -\|h\|_2$.
        Store< $s_t$, $o_t$, $g_t$, $a_t$, $t_t$, $s_{t+1}$, $o_{t+1}$, $g_{t+1}$ $>_{lo}$ in $D_{lo}$.
    end for
    for each gradient step $s$ $(=$ env. step) do
        if $s$ mod $c$ then
            do off-policy correction on $D_{hi}$.
            Update high-level critics. $\rightarrow$(7)
        end if
        if $s$ mod $2c$ then
            Update high-level actor. $\rightarrow$(4)
            Update high-level target networks.
        end if
        Update low-level critics. $\rightarrow$(9)
        if $s$ mod 2 then
            Update low level actor. $\rightarrow$(8)
            Update low level target networks.
            Update $\alpha$. $\rightarrow$(10)
        end if
    end for
end for
```

out any constraints in the state-action space, thus resolving the first difficulty of RL for robotics that was discussed to in the introduction section.

4. LEVERAGING HUMAN DEMONSTRATION

Although the proposed method can learn the end-to-end control policy of a robot from scratch, this learning is still difficult as a substantial number of training samples are required to train six neural networks along with their performances μ_θ^{hi} and π_θ^{lo} which are dependent on each other. Thus, the use of training samples that have a good reward signal can facilitate learning.

In this study, a robotic teleoperation system was set up, as shown in Fig. 6, to obtain the necessary human demonstration data for the desired task. The teleoperation was used to acquire RGB videos of a robot that performs the task without the human demonstrator appearing in the image, making the training data identical to those obtained by the robot agent's own interaction. The hierarchical policy of H-HSAC eventually estimated the joint velocity command $\dot{\boldsymbol{q}}_d$ as action a_t. A 7-DOF robot was teleoperated using Jacobian pseudo-inverse matrix-based joint velocity control, which converted the target Cartesian commands $\dot{\boldsymbol{x}}_d$ and $\boldsymbol{x}_d$ to $\dot{\boldsymbol{q}}_d$.

As a human demonstrator can be deemed as an expert policy π^*, it is assumed that the data $\tau_{demo} =< s_t, o_t, a_t, t_t, R_t, s_{t+1}, o_{t+1} >$ generated from demonstration always satisfies c step goal-conditioning g_{kc} between high- and low-level policies. Hence, the expert subgoal g_{kc} can be derived from the difference $s_{(k+1)c} - s_{kc}$, as depicted by red arrows in Fig. 7. Next, the goal-transition of (11) is applied to obtain the interim goals up to g_{2kc-1} (black arrow of Fig. 7). Finally, if a terminal state s_T exists over the intervening time steps, the last g_{kc} is computed from $g_{kc} = s_T - s_{kc}$ (blue arrow in Fig. 7). After obtaining subgoals from this approach, the whole demo data is reorganized as τ_{demo}^{hi} and τ_{demo}^{lo} for high- and low-level policy learning, respectively. The restructured data are then stored in the replay buffers and used to facilitate hierarchical policy learning in H-HSAC, thus making it possible to overcome the second difficulty of RL, which requires a substantial amount of data.

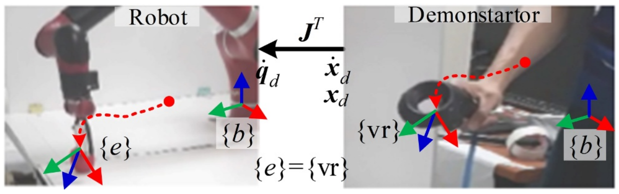
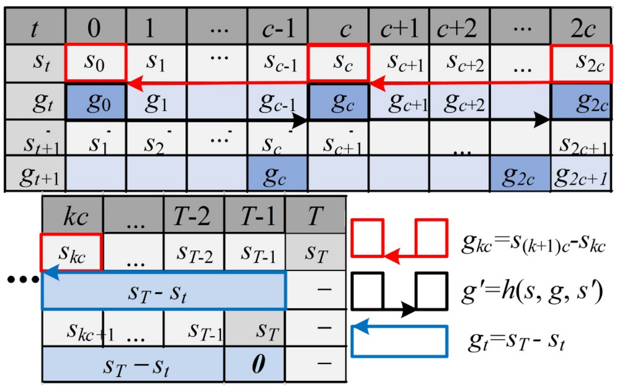


Fig. 6. Human demonstration collection via VR device.


t | 0 | 1 | … | c-1 | c | c+1 | c+2 | … | 2c
s_t | s_0 | s_1 | … | s_{c-1} | s_c | s_{c+1} | s_{c+2} | … | s_{2c}
g_t | g_0 | g_1 | g_{c-1} | g_c | g_{c+1} | g_{c+2} | g_{2c}
s_{t+1} | s_1 | s_2 | … | s_c | s_{c+1} | … | s_{2c+1}
g_{t+1} | g_c | g_{2c} | g_{2c+1}
kc | … | T-2 | T-1 | T
s_{kc} | … | s_{T-2} | s_{T-1} | s_T
$s_T - s_t$ | –
s_{kc+1} … | s_{T-1} | s_T | –
$s_T - s_t$ | 0 | –
$g_{kc}=s_{(k+1)c}-s_{kc}$
$g'=h(s, g, s')$
$g_t=s_T - s_t$
Fig. 7. Transition data generation from demonstration.

5. SIM-TO-REAL TRANSFER OF LEARNED POLICY

One of the aims of this study is to learn the end-to-end control policy function in simulation environments alone and then directly apply the results to a real robot. In this way, it is possible to execute rapid learning without any risk of collision caused by stochastic exploration in the real robot learning process.

However, there are certain discrepancies between the real world and simulation environments, such as visual information and physical modeling of objects, which deteriorates the performance of simulation to real-world (sim-to-real) transfer learning. Therefore, domain randomization (DR) [44] is employed to bridge the gap between the simulation environment and the real-world test setup. The components listed in Table 1 were randomized by virtue of the H-HSAC policy that estimates the control command from the vision and the joint sensor policy that estimates the control command from the vision and the joint sensor information.

Fig. 8(a) illustrates the randomized visual domains centered in the default simulation environment (Fig. 8(b)). As the hierarchical policy with DR being learned, the policy gradually becomes more generalized, ultimately expanding its estimation accessible domain to the real-world domain (Fig. 8(c)). Thus, a real robot can be controlled in the target test domain (Fig. 8(c)) with a policy network trained solely in randomized simulation environments. Note that the visual properties of the robot and its surrounding environment are randomized for each time step, and that the robot's joint properties are changed for every episode.

Table 1. Randomized properties in simulation environment.

State type	Randomized properties
Visual, o	Color and position of target objects, robot link materials, ambient lights, environmental lights
Measured, s_{part}	Joint damping, friction coefficients, proportional gain of the joints

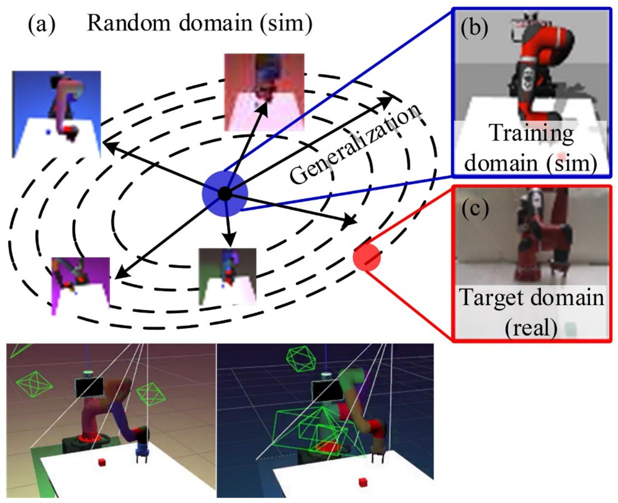


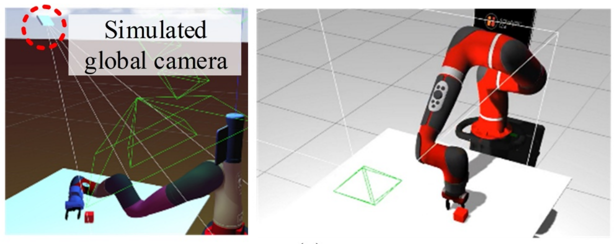
Fig. 8. Sim-to-real transfer learning via domain randomization.

6. EXPERIMENTAL ANALYSIS

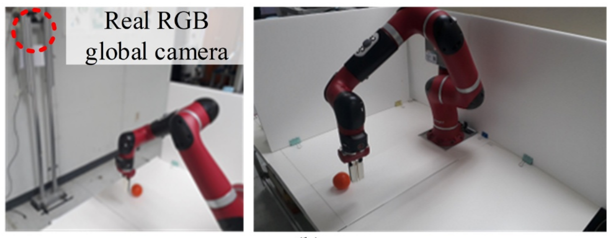
In this section, the training and test performances of H-HSAC are evaluated in a reach-and-pick task of a 7-DOF manipulator. The training evaluation is conducted by examining the stability of the proposed method, the effectiveness of using human demonstration, and the efficacy of the newly proposed off-policy correction. Further, H-HSAC is compared to other deep RL algorithms. To assess the test performance, H-HSAC was used to perform reach-and-pick task on both simulation and real environments, with the results shown in Figs. 9(a) and 9(b), respectively. The reward function R_t given by the environment for learning the reach-and-pick task is defined as

$$R_t(s,a) = -\|x_e - x_o\|_2 + b_{\text{picked}}, \tag{17}$$

where $\boldsymbol{x}_e$ and $\boldsymbol{x}_o$ represent the Cartesian position of the robot's end-effector and the position of the target object, respectively, and b_{picked} represents the positive sparse reward that is only given if the robot's gripper successfully picks the target object. Hence, the negated distance between the two incentivizes the robot to reach the target object.



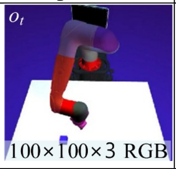
(a)



(b)

Fig. 9. Experimental environments of reach-and-pick task: (a) virtual environment, and (b) real world.

Table 2. State, observation, and action space defined for the experiments.

State space Σ	Observation space O	Action space A
$s_t = s_{\text{full}} = s_{part} \cup s_{aux}$ $s_{part} = [\boldsymbol{q}, \dot{\boldsymbol{q}}, \tau]$ $s_{aux} = [x_e, x_o, b_{grip}]$	 o_t 100×100×3 RGB	$a = [\dot{\boldsymbol{q}}_d \parallel b_{grip}]$

As proposed in the introduction section, the hierarchical policy estimates seven joint velocities and gripper state of a robot by observing the global camera and joint encoder information. Accordingly, the state space, observation space, and action space of the agent and environment are defined in Table 2. In this study, the full state s_{full} of the environment is a union of the partial state s_{part} and the auxiliary state s_{aux}, where s_{part} is a set of joint positions q, velocities $\dot{q}$, and torques τ, whereas s_{aux} consists of x_e, x_o, and the binary state of the gripper, b_{grip}. Further, the observation o represents an RGB video of the robot performing the tasks. Finally, the action consists of the seven joint velocities and gripper commands of a robot.

The experimental conditions are as follows: The control period of the robot was fixed to 30 Hz for both training and testing, and the maximum episode length was set at 1,000 steps. The training performance measure was an undiscounted episodic return $\Sigma^T R_t$. In (17), b_{picked} was set to 5. Furthermore, if the end-effector of the robot went out of the valid workspace, the agent was penalized with a scalar value of -1. The experimental conditions are as follows. The control period of the robot was fixed to 30 Hz for both training and test, and the maximum episode length was 1,000 steps. The training performance measure was an undiscounted episodic return $\Sigma^T R_t$. In (17), b_{picked} was set to 5. Furthermore, the agent was penalized with a scalar value of -1 if the end-effector of the robot went out of the valid workspace.

6.1. Training performances in a simulation environment

The efficacy of the off-policy correction discussed in Subsection 3.3 was compared with that of the correction presented in [29], as shown in Fig. 10(b). The off-policy

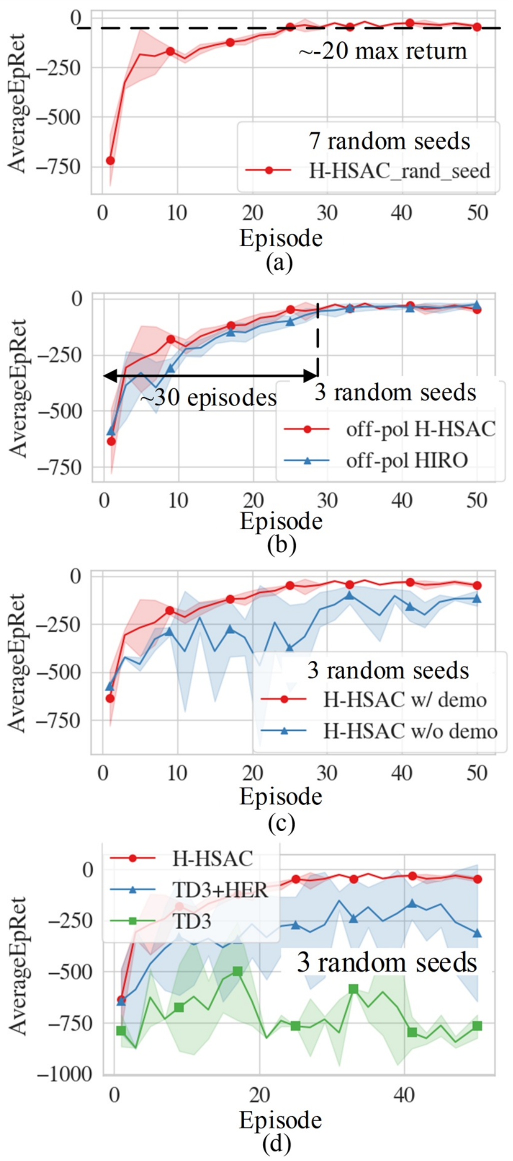


Fig. 10. Training performance curves for experiments: (a) H-HSAC over 7 different random seeds, (b) efficacy of the proposed off-policy correction in H-HSAC compared with that of HIRO [13], (c) efficacy of human demonstration on the task, and (d) comparison of H-HSAC, TD3 + HER, and pure TD3.

correction method (red solid line) outperformed that of [29] (solid blue line) in the early- and mid-stage of learning, and both methods ultimately converged to a similar performance, which indicates that the proposed method off-policy correction method is more suitable for stable learning with the entropy regularized low-level policy. A method for improving training performance using human demonstrations was also investigated. Fig. 10(c) shows the effect of using 20 human demonstration episodes. The performance and stability of the learning with demonstration data surpassed those of learning with purely exploratory data, indicating that the use of demonstration increases the sample efficiency and improves the policy.

Finally, H-HSAC was compared to TD3+HER (hindsight experience replay, [28]) as well as vanilla TD3. These three algorithms have similar state, observation, and action spaces, and their replay buffers were augmented with identical demonstration data. As shown in Fig. 10(d), vanilla TD3 did not improve the policy at all, whereas goal-conditioning TD3 with HER could facilitate policy improvement, but was still not comparable to H-HSAC in terms of both performance and stability. Consequently, the goal-conditioned hierarchical approach of H-HSAC is a reasonable choice for control policy learning in high-dimensional state-action space.

6.2. Test performances in a real-world environment

As the action space is involved in the joint velocities of a robot, the learned end-to-end control policy should have the ability to estimate continuous joint velocity commands that do not have a detrimental effect on the robot. In Fig. 11, which shows a part of recorded control commands estimated for a real robot, the overall shape of the trajectories was continuous aside from a few small jerks, which implies that the policy is qualitatively well trained. The learned policy was quantitatively evaluated over 50 episodes of the reach-and-pick task in the real world. The success rate was checked for the three cases of reach, reach-and-pick, and reach-and-pick with a valid pose, as listed in. In evaluating the reaching performance, it was determined that the reaching was successful if the robot end-effector and the object were within 3 cm, although the position of the target object was randomly initialized for every episode. As presented in Table 3, the control policy of a real-world robot could be learned without the need for exploratory data acquisition using real robots.

In the simulation environment, the success rates based on the three criteria are 100%, 54%, and 20%, respectively, and Fig. 12(a) shows one of the test episodes in the simulation environment. This indicates that the learned

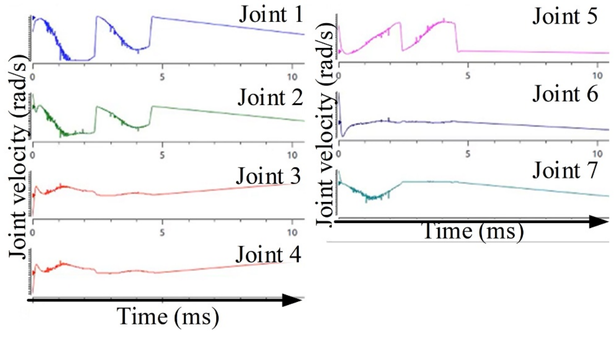


Fig. 11. Joint velocity commands inferred from the control policy.

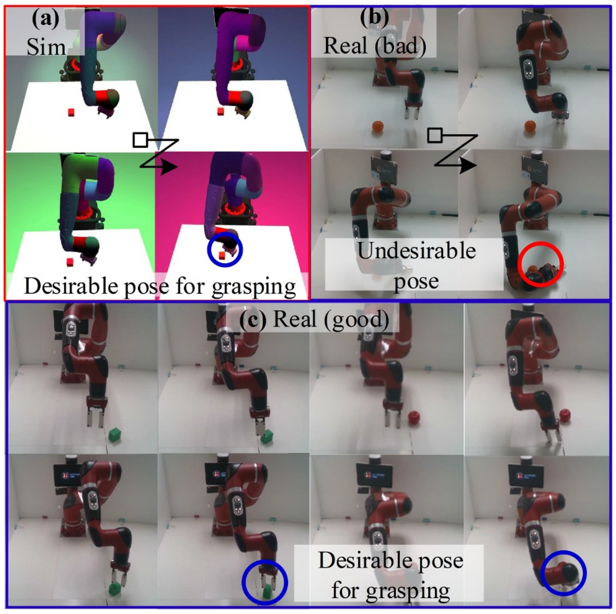


Fig. 12. Sampled test episodes of reach-and-pick task over simulated and real world environments, executed with the same policy network. (a) Succeed in the simulated environment. (b) Failure in the real-world environment. (c) Successfully achieved tasks in real-world environment: this verifies the sim-to-real transfer learning of a control policy.

policy of H-HSAC can independently control the robot using camera and joint encoder information, although it lacks some generalization over the various positions of the target object in the tested workspace.

The success rates were compared for the two real-world cases of the policy learned with and without DR. As shown in Table 3, the test performance of the learned policy in the simulation environment was substantially deteriorated as it was transferred to the real world, but applying DR could prevent the performance degradation to some extent. In the real-world, for a few episodes, the agent showed an undesirable pose despite the fact that it had grasped the target object (Fig. 12(b)), while for some other episodes, the agent successfully picked the target object with a desirable pose (Fig. 12(c)).

Furthermore, H-HSAC was compared with TD3+HER and TD3 for the reach-and-pick task, with the results shown in Table 4. Note that the test performances among the three are similar in terms of training performance, as shown in Fig. 10(d). From this result, we further verified that the goal-conditioned hierarchical policy of H-HSAC is a suitable approach for end-to-end control policy learning.

In summary, H-HSAC can learn the end-to-end control policy of a robot for achieving a specific task. In summary, H-HSAC can learn the end-to-end control policy of a robot

Table 3. Reach-and-pick task success rate comparison over different domains.

	Simulated environment	Real environment	
		with DR	w/o DR
Reach (within 3 cm)	50/50 (100%)	32/50 (64%)	11/50 (22%)
Reach & pick	27/50 (54%)	14/50 (28%)	0/50 (0%)
Valid grasping pose	10/50 (20%)	2/50 (4%)	-

Table 4. Reach-and-pick task success rate comparison among different algorithms.

	H-HSAC (proposed)		TD3+HER		TD3 [8]	
	Sim	Real	Sim	Real	Sim	Real
Reach (within 3cm)	50/50	32/50	35/50	10/50	2/50	0/50
Reach & pick	27/50	14/50	9/50	0/50	0/50	-
Valid grasping pose	10/50	2/50	0/50	-	-	-

for achieving a specific task. However, the generalization performance should be increased to guarantee the practicality of the proposed method.

6.3. Applications to complex tasks

To further exemplify the performance of H-HSAC, we conducted additional experiments on more complex tasks than a reach-and-pick task. The additional tasks are shown in Fig. 13. For the pick & place task involving multiple objects (Fig. 13(a)), the reward function is a modification of (17), which is designed in a curriculum fashion.

Before grasping the object, the reward is the sum of the negated Euclidean distance to all targets, with a bonus of 5 if the agent picks the target object. If the object is grasped, the reward changes to the summation of negated Euclidean distance to the corresponding placing goal, which is categorized as the same color (blue, green, yellow). If the agent successfully places the object, it also gets a bonus reward of 5. Each episode has a maximum length of 5,000 steps (about 150 seconds). We conducted these experiments and compared H-HSAC to the state-of-the-art model-free, off-policy HRL algorithms (HIRO, hierarchical actor-critic (HAC)) [29,35], which fall into the same category with H-HSAC, and the state-of-the-art non HRL agents (SAC, TD3).

For more challenging tasks: peg-in-hole and block stacking task, each episode has a maximum length of 1,000 steps. The reward function of a peg-in-hole task is also designed in a similar way to that of a pick-and-place

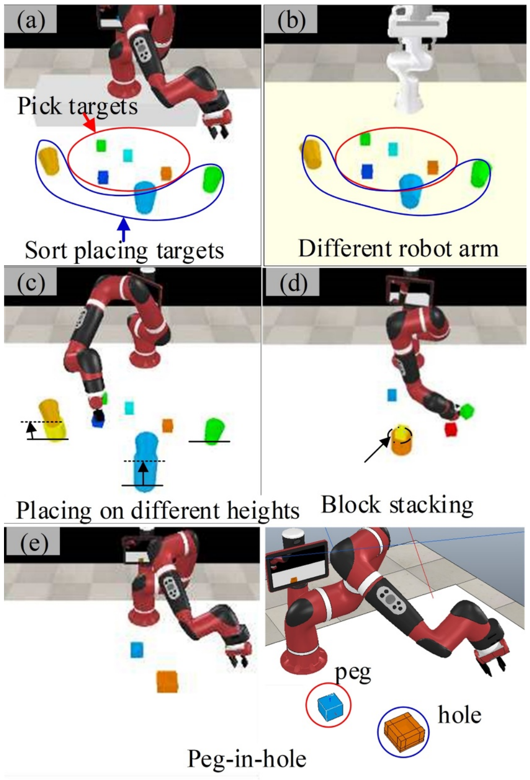


Fig. 13. Simulated environment for solving complex tasks. (a) Solving a pick-and-place task of multiple objects. The agent should sort blue, green, and yellow objects into the blue, green, and yellow cups, respectively. For effective sim-to-real transfer, while applying domain randomization, the range of color randomiza-tion of target objects was limited. (b) The same task with (a) but with a kinematically different robot arm (Panda). (c) Variants of (a). The height of the cups is different from before. (d) Block stacking task. The agent should stack cubic objects on the or-ange cylinder. (e) Peg-in-hole task. The agent should insert a blue peg into the orange assembly hole. The clearance is about 1 mm.

task but has additional bonus of 10 for successful assembly. For a block stacking task, the reward function is similar to that of a pick-and-place task, with the difference of goal position being set at the top of lastly stacked block.

6.4. Training results on complex tasks

Fig. 14 shows the training performance of H-HSAC and other RL algorithms in various task environments discussed in the previous section. From Figs. 14(a) to 14(d), the agents were trained for 10,000 episodes and evaluated every 100 episodes with the averaged return over 3 runs. Moreover, all experiments were conducted for three different random seeds with additional samples from the demonstration data. First, H-HSAC was compared with its variants whose combination of high- and low-level policies were different from H-HSAC. To begin, for all comparisons with other methods, H-HSAC (in purple curve) shows the best stability for the random seed which is demonstrated by less fluctuation of curves over the procedure of learning and small shaded area that implies low performance variance for multiple experiments. It was observed from Fig. 14(a) that, over the course of training, H-HSAC (in blue curve) having a deterministic high-level policy and a stochastic low-level policy outperformed the other algorithms with better stability (lower variance indicated by the shadow). This result justifies the choice of the deterministic high-level and low-level policies with the newly proposed off-policy correction method.

Figs. 14(b) and 14(c) show the training performance among various algorithms for Sawyer and Panda robots. In both experiments, H-HSAC (in purple) outperformed all other algorithms. In addition, it was found that the hierarchical RL agents (H-HSAC, HIRO, HAC) outperformed its counterparts that had a flat policy (SAC, TD3) by large margin. Next, Fig. 14(d) demonstrates that for a more kinematically complex task, the performance of H-HSAC was better than other RL algorithms.

Fig. 14(e) shows the training performance of two more complex tasks. The agent should infer the trajectory-force relationship in an end-to-end manner in the peg-in-hole task and should adapt to a varying kinematic goal for a block stacking task. Only the performance of H-HSAC was reported since the seemingly promising training results did not correspond to the successful achievement of the two tasks. Thus, further experiments for other algorithms were not conducted, and this will be discussed in the next section.

6.5. Test results on complex tasks

With the trained agent discussed in the previous section, the trained agents for solving the complex tasks were evaluated. Fig. 15 shows the sampled trajectories from the test in the real world. Figs. 15(a) and 15(b) show the experimental setup that the simulated environment in Fig. 13 emulates. Fig. 15(c) demonstrates the sampled trajectory of a real-world pick-and-place task. The agent sorted the blocks on the table according to its category of color. Furthermore, Figs. 15(d) and 15(e) show the peg-in-hole and block stacking tasks conducted in the real world.

The qualitative results are summarized in Tables 5-7. The '#' in the table indicates the number of blocks that have been successfully picked and placed at the target. Table 5 lists the test results of the pick-and-place task for various settings. In the leftmost column, the first two rows compare the pick-and-place operation between Sawyer and Panda in simulated environments, as illustrated in Figs. 13(a) and 13(b). The result describes that H-HSAC works well for both robot arms with different kinematic configurations. This proves that the proposed method is

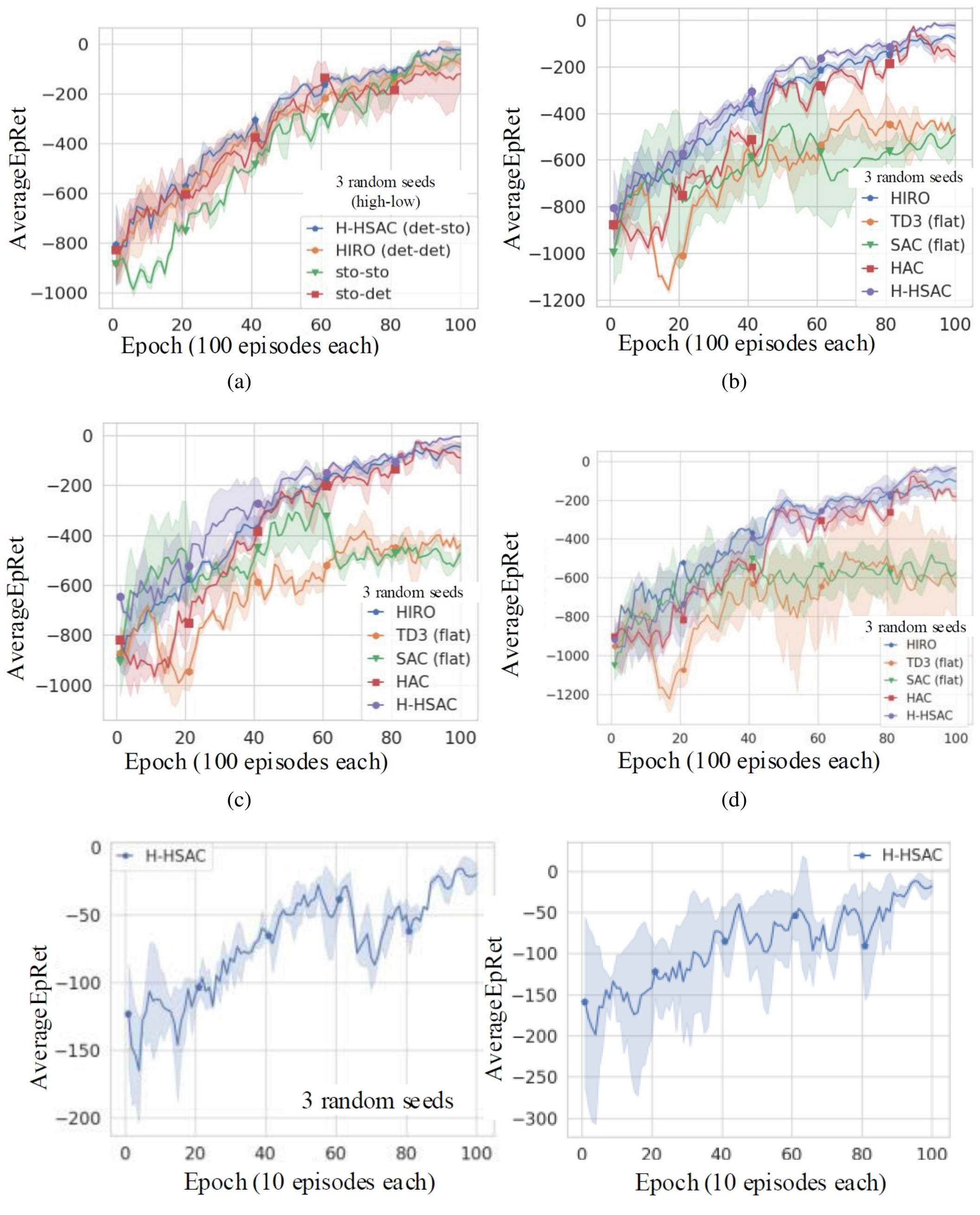


Fig. 14. Training performance on complex tasks. (horizontal axis: pro-cedure of training. vertical axis: cumulative return value for a single episode. solid line denotes average over 3 runs and shaded area is the variance): (a) Pick-and-place task of multiple targets for different combination of hierarchy. (b) Pick-and-place task of multiple targets with Sawyer arm. (c) The same task with Panda arm. (d) Pick-and-place task at varying heights. (e) Left: peg-in-hole task; right: block stacking task.

applicable to general robot arms. Moreover, the 3rd row showing the test result on the pick-and-place task at different goal heights demonstrates that H-HSAC can solve complex kinematic problems in an end-to-end manner.

Next, Table 6 compares H-HSAC to two different HRL algorithms (i.e., HIRO, HAC) on a pick-and-place task trained in Fig. 14(b). The result demonstrates that H-HSAC is competitive with those two state-of-the-art algorithms.

Finally, Table 7 lists the resulting performance on the peg-in-hole and block stacking tasks. As mentioned in Subsection 6.3, the test results do not comply with the

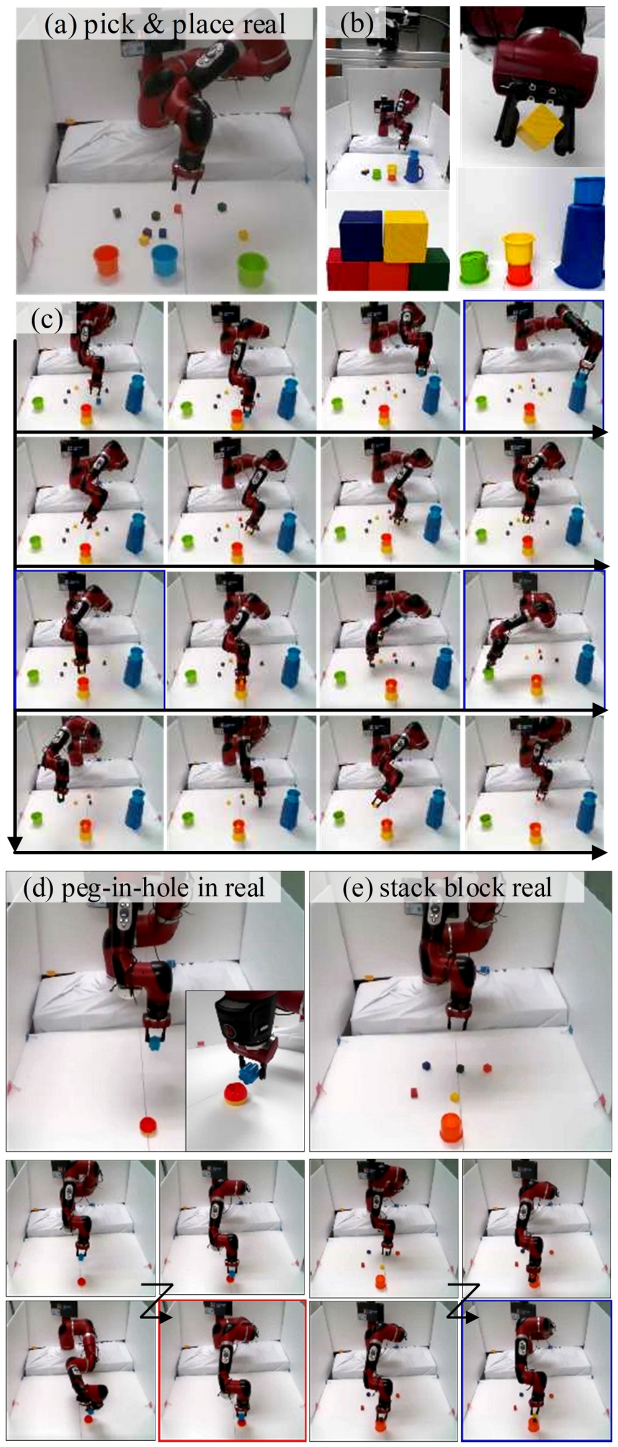


Fig. 15. Qualitative test results of complex tasks in the real world. (a), (b) Experimental setup with real robots. (c) Successful trajectory of the pick-and-place operation on goals at different heights. (d), (e) Sampled trajectory of the peg-in-hole and block stacking tasks.

Table 5. Quantitative evaluation of H-HSAC in the pick-and-place task.

Task		Simulated environment	Real environment with DR
	#	Successful placing over 20 trials	
Pick-and-place with Sawyer (H-HSAC)	0	-	2
	1	-	5
	2	5	7
	3	11	6
	4	2	-
	5	2	-
Pick-and-place with Panda (H-HSAC)	0	-	-
	1	-	
	2	4	
	3	14	
	4	1	
	5	1	
Pick-and-place at different heights with Panda (H-HSAC)	0	-	2
	1	-	4
	2	5	8
	3	14	6
	4	1	0
	5	0	0

Table 6. Quantitative test performance of H-SHAC, HIRO, and HAC.

	H-HSAC (proposed)		HIRO		HAC	
	#	Suc.	#	Suc.	#	Suc.
Pick-and-place with Panda in simulated environment	0	-		1		3
	1	-		1		4
	2	5		5		6
	3	11		12		7
	4	2		1		-
	5	2		-		-

Table 7. Performance of H-HSAC on peg-in-hole and block stacking tasks.

		H-HSAC	
		# of success / blocks over 10 trials	
		Sim	Real
Peg-in-hole		2/10	- /10
Block stacking	0	1	3
	1	7	7
	2	2	-

good training performance reported in Fig. 13(e). It shows the limitation of the proposed method in that the peg-in-hole and block stacking operations require the inference of the physical dynamics properties of the environment. In other words, the hierarchical decomposition of H-HSAC has limitations in figuring out the unknown physical laws of the environment.

7. CONCLUSION AND FUTURE WORK

In this study, we presented a H-HSAC algorithm, which is a goal-conditioned hierarchical control policy for controlling a multi-DOF robot manipulator using the camera and the joint encoder information. The proposed algorithm overcome three difficulties associated with RL on robotics by adopting hierarchical, human demonstration, and simulated learning.

The experimental results on the training performance indicate that H-HSAC was able to learn the end-to-end control policy that was robust to a change in the random seed. Utilizing human demonstrations improved the performance over the entire course of learning. Finally, H-HSAC outperformed the pure TD3 and TD3+HER algorithm on both training and test performances. During the training, H-HSAC scored higher mean and lower variance return, while the other two received substantially lower value and unstable return.

The evaluation results on the test performance suggest that domain randomization played a critical role in sim-to-real transfer learning. Additionally, H-HSAC could control a real-world robot in an end-to-end manner with success rates of 64% and 28% success rates for the reach and the reach-and-pick tasks, respectively, while the pure TD3 and TD3+HER algorithms almost failed to achieve both tasks.

The test results indicate that the generalization of the end-to-end control policy for various tasks and the improvement on the qualitative performance (e.g., increasing the number of valid grasps) both remain issues.

Furthermore, in more complex tasks such as pick-and-place operations of multiple objects with different kinematic conditions, the performance of H-HSAC was generalized to be applied to two different types of robots. In addition, H-HSAC outperformed other state-of-the-art model-free, off-policy hierarchical RL algorithms in such tasks. However, its limitation was demonstrated by the environments where implicit physical laws affect the agent-environment interaction.

In future research, we will train the policy networks through variational inference to achieve a better understanding of the visual information and hidden physics model.

REFERENCES

[1] S. Xiao and J. Dong, "Adaptive fault-tolerant control for a class of uncertain TS fuzzy systems with guaranteed time-varying performance," *Fuzzy Sets and Systems*, vol. 385, pp. 1-19, April 2020.

[2] R. Raul-Cristian, R.-E. Precup, and E. M. Petri, "Hybrid data-driven fuzzy active disturbance rejection control for tower crane systems," *European Journal of Control*, vol. 58, pp. 373-387, March 2021.

[3] A. Turnip and J. H. Panggabean, "Hybrid controller design based magneto-rheological damper lookup table for quarter car suspension," *International Journal of Artificial Intelligence*, vol. 18, no. 1, pp. 193-206, March 2020.

[4] R. Bellman, "Dynamic programming and Lagrange multipliers," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 42, no. 10, p. 767, October 1956.

[5] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238-1274, September 2013.

[6] F. Zhang, J. Leitner, M. Milford, and P. Corke, "Modular deep Q networks for sim-to-real transfer of visuo-motor policies," *arXiv preprint*, arXiv:1610.06781, October 2016.

[7] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 763-768, May 2009.

[8] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469-483, May 2009.

[9] C. Daniel, G. Neumann, and J. Peters, "Hierarchical relative entropy policy search," *Artificial Intelligence and Statistics*, PMLR, pp. 273-281, March 2012.

[10] S. Levine and V. Koltun, "Guided policy search," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1-9, May 2013.

[11] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, "Neural task programming: Learning to generalize across hierarchical tasks," *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3795-3802, May 2018.

[12] R. Julian, E. Heiden, Z. He, H. Zhang, S. Schaal, J. Lim. G. Sukhatme, and K. Hausman, "Scaling simulation-to-real transfer by learning composable robot skills," *Proc. of International Symposium on Experimental Robotics*, pp. 267-279, Springer, Cham, November 2018.

[13] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2169-2176, May 2017.

[14] D. A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, and J. C. Niebles, "Neural task graphs: Generalizing to unseen tasks from a single video demonstration," *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8565-8574, 2019.

[15] S. Sohn, J. Oh, and H. Lee, "Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies," *Advances in Neural Information Processing Systems*, pp. 7156-7166, December 2018.

[16] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," *Advances in Neural Information Processing Systems*, vol. 10, pp. 1043-1049, 1997.

[17] Z. J. Pang, R. Z. Liu, Z. Y. Meng, Y. Zhang, Y. Yu, and T. Lu, "On reinforcement learning for full-length game of starcraft," *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 1, pp. 4691-4698, July 2019.

[18] S. Nair and C. Finn, "Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation," *arXiv preprint*, arXiv:1909.05829, September 2019.

[19] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181-211, August 1999.

[20] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227-303, 2000.

[21] D. Precup, *Temporal Abstraction in Reinforcement Learning*, University of Massachusetts Amherst, 2000.

[22] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 5, pp. 271-278, 1992.

[23] M. Klimek, H. Michalewski, and P. Mi, "Hierarchical reinforcement learning with parameters," *Proc. of Conference on Robot Learning*, PMLR, pp. 301-313, October 2017.

[24] P. L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, pp. 1726-1734, February 2017.

[25] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "FeUdal networks for hierarchical reinforcement learning," *Proc. of International Conference on Machine Learning*, PMLR, pp. 3540-3549, July 2017.

[26] J. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine, "Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1009-1018, July 2018.

[27] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," *arXiv preprint*, arXiv:1712.00948, December 2017.

[28] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, R. P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, W. Zaremba, "Hindsight experience replay," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[29] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[30] Y. Jiang, S. Gu, K. Murphy, and C. Finn, "Language as an abstraction for hierarchical deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[31] C. Li, F. Xia, R. Martin-Martin, and S. Savarese, "HRL4IN: Hierarchical reinforcement learning for interactive navigation with mobile manipulators," *Conference on Robot Learning*, PMLR, pp. 603-616, May 2020.

[32] D. Jain, A. Iscen, and K. Caluwaerts, "Hierarchical reinforcement learning for quadruped locomotion," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7551-7557, November 2019.

[33] O. Nachum, M. Ahn, H. Ponte, S. Gu, and V. Kumar, "Multi-agent manipulation via locomotion using hierarchical sim2real," *arXiv preprint*, arXiv:1908.05224, August 2019.

[34] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. T. Springenberg, M. Neunert, T. Hertweck, T. Lampe, N. Siegel, N. Heess, and M. Riedmiller, "Compositional transfer in hierarchical reinforcement learning," *arXiv preprint*, arXiv:1906.11228, June 2019.

[35] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," *arXiv preprint*, arXiv:1712.00948, December 2017.

[36] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1587-1596, July 2018.

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

[38] S. Nasiriany, V. Pong, S. Lin, S. Levine, "Planning with goal-conditioned policies," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[39] 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, September 2015.

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

[41] N. Chentanez, A. Barto, and S. Singh, "Intrinsically motivated reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 17, 2004.

[42] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," *arXiv preprint*, arXiv:1710.06542, October 2017.

[43] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, November 2018.

[44] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23-30, September 2017.

Cheol-Hui Min received his B.S. degree in mechanical engineering from Korea University, in 2017, and an M.S. degree in mechanical engineering from Korea University, in 2019. He is currently pursuing a Ph.D. degree in electrical and computer engineering at Seoul National University. His research interests include 3D vision for robotics, model-based deep reinforcement learning, and optimal control.

Jae-Bok Song received his B.S. and M.S. degrees in mechanical engineering from Seoul National University, in 1983 and 1985, respectively. He was awarded his Ph.D. degree from M.I.T., in 1992. He is currently a Professor at the School of Mechanical Engineering at Korea University. He has served as a director of Intelligent Robotics Laboratory since 1993. His research interests include robot safety and robotic system design and control.