A Derivations of Equation (10)

By representing $J_{\pi_s}(s_t) = \widetilde{v}_{\pi_s}(s_t)$, Equation (9) can be derived by successor states as follows:

$$\begin{aligned} J_{\pi_s}(s_t) = \widetilde{v}_{\pi_s}(s_t) &= \mathbb{E}_{\pi_s}[G_t + w_s H_{\pi_s}(a_t|s_t)] \\ &= \mathbb{E}_{\pi_s}[r(s_t, a_t) + w_s h_{\pi_s}(a_t|s_t) + \gamma\{G_{t+1} + w_s H_{\pi_s}(a_{t+1}|s_{t+1})\}] \\ &= \mathbb{E}_{\pi_s}[r(s_t, a_t) + w_s h_{\pi_s}(a_t|s_t) + \gamma\widetilde{v}_{\pi_s}(s_{t+1})], \end{aligned} \tag{21}$$

where Equation (21) is the Bellman equation for $\widetilde{v}_{\pi_s}(s_t)$. By applying the modified Bellman backup operation in [23] to Equation (21), then $\widetilde{v}_{\pi_s}(s_t)$ is identical to the soft state value function in [23] as

$$J_{\pi_s}(s_t) = \widetilde{v}_{\pi_s}(s_t) = \mathbb{E}_{\pi_s}[\widetilde{q}_{\pi_s}(s_t, a_t) + w_s h_{\pi_s}(a_t|s_t)] \tag{22}$$

where $\widetilde{q}_{\pi_s}(s_t, a_t) = r(s_t, a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}[\widetilde{v}_{\pi_s}(s_{t+1})]$ is the soft Q-function for π_s.

B Experimental Details

In this section, we describe experimental details. For the implementation, we use Pytorch as a deep learning tool and Coppeliasim as a robotic simulator, and we conduct experiments on a workstation with Intel i9-7920X CPU and Nvidia TITAN Xp GPU.

B.1 Network architectures

B.1.1 Actor networks

We use actor networks for all the methods we used as a three-layered MLP of size 64 with ReLU activations except the last layer. The last layer outputs two heads, mean and standard deviation, for an action distribution. Here, the action distribution is designed by a normal distribution.

B.1.2 Critic networks

We design critic networks for all the methods we used as a two-layered MLP of size 64 with ReLU activations except the last layer.

B.1.3 Discriminator network for DIAYN

When utilizing DIAYN, we design a discriminator network as a two-layered MLP of size 64 with ReLU activations except the last layer. The last layer outputs two heads, mean and standard deviation, for a skill distribution, where each output has L-dimensionality size. The skill distribution is designed by a normal distribution.

B.2 Hyperparameters

Hypermaraters are summarized in Table 1.

Table 1: Hyperparameters

Parameters	Value
learning rate	$3 \cdot 10^{-4}$
gradient steps	50
batch size	256
discount factor	0.99
target smoothing coefficient	0.005
replay buffer size	10^3
L-dimensionality for DIAYN	5
optimizer	Adam

B.3 Training

The total number of steps for training is 10^6 and the maximum steps per episode is 10^2. An episode ends when the task fails or succeeds. Here, the task fails when one of the following cases occurs: (1) the task does not succeed until the episode reaches the maximum step and (2) a state corresponds to one of the failure conditions of the task. In success case that the robot completes a given task, a huge reward is given for the success. Whenever an episode ends, each method updates its own neural network models.

C Environment Details

In this section, we describe state-action spaces, a reward design, and a heuristic information design for more detailed explanation.

C.1 State-action spaces

As we mentioned in the manuscript, the environment for a task contains three types of objects, such as a robot, a object, and a detection sensor in a target position. A state and an action are described in more detail as following sections.

C.1.1 State

The state consists of three types of information, as robot information, object information, and target position information. A size of the robot information is 29, which includes information of the arm and the gripper. The arm information consists of 6 joint angular positions and 6 joint angular velocities. The gripper information includes 3-dimensional positions, 4 quaternions, 3 position velocities, and 3 rotation velocities of an end-effector. In addition, the gripper information includes 2 joint positions and 2 joint velocities of fingers attached at the end-effector.

A size of the object information is 13, which includes 3-dimensional positions, 4 quaternions, 3 position velocities, and 3 rotation velocities of the object. A size of the target position information is 13, which includes 3-dimensional positions, 4 quaternions, 3 position velocities, and 3 rotation velocities of the detection sensor placed on the position.

C.1.2 Action

The action is based on a velocity control for the arm joints and a binary signal for the gripper's fingers. A size of the action is 7, which includes 6 joint velocities for the arm and 1 binary signal for the gripper indicating open or close. To bound the action in the range $(-1, 1)$, we apply hyperbolic tangent function as the same way in SAC.

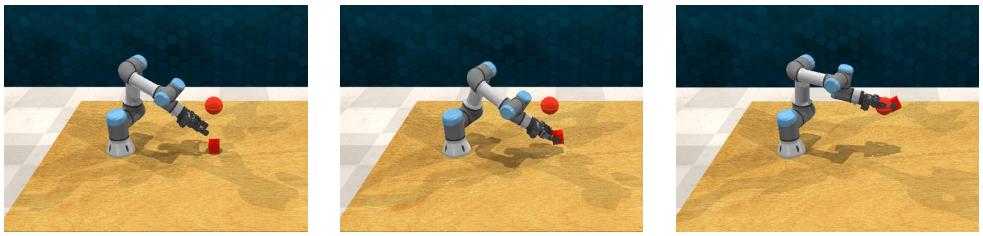
D Details of *Pick-and-Lift*

D.1 Task design

As shown in Figure 4, the environment of *Pick-and-Lift* consists of a single UR3 arm with 6 DoF including the two-finger gripper, a cube object on a table, and a target position. To succeed the task, successive hierarchical sub-tasks are required, such as reaching out the gripper for the cube, grasping stably the cube, and lifting the cube without pitching, and moving and exactly placing the cube to the target position. A failure condition is defined as "the cube falls under the table" or "the robot moves the cube by pushing not grasping".

D.2 Reward design

We design the reward of *Pick-and-Lift* as the sum of dense and sparse rewards. As we mentioned in the manuscript, the rewards for reaching and moving are designed by dense rewards and the reward for grasping is designed by sparse reward. In detail the rewards for reaching and moving are



(a) reaching the cube (b) grasping the cube (c) moving to the target position

Figure 4: *Pick-and-Lift*: it requires reaching out the gripper for the cube, grasping stably the cube, and moving and exactly placing the cube to the target position.

designed as

$$r_{reaching}(O_g, O_c) = -\sqrt{\left(pos(O_g) - pos(O_c)\right)^2},$$
$$r_{moving}(O_c, O_t) = -\sqrt{\left(pos(O_c) - pos(O_t)\right)^2},$$

where $pos(\cdot)$ is 3-dimensional positions, O_g is the gripper, O_c is the cube, and O_t is the target position. The reward for grasping is designed as

$$r_{grasping}(O_c) = \begin{cases} 1, & \text{for the case of the gripper grasping } O_c \\ 0, & \text{for otherwise} \end{cases},$$

where a touch sensor is used in the gripper to recognize whether the gripper grasps the cube.

To encourage a policy to complete the task as soon as possible, a success reward is designed as

$$r_{success}(O_c, O_t) = \begin{cases} T_{max} - step(t), & \text{for the case of } O_c \text{ located in } O_t \\ 0, & \text{for otherwise} \end{cases},$$

where T_{max} is the maximum number of steps per episode, $step(t)$ is the number of steps taken up to now, and a detection sensor is used to recognize whether the cube is located in the target position. If the task is succeed quickly, the higher success reward is given.

To support the cube to be placed in the gripper, an auxiliary reward for the gripper's fingers is designed as

$$r_{fingers}(O_c) = -\cos\psi(O_f, O_c),$$

where, O_f is the gripper's fingers, and $\psi(O_f, O_c)$ is an angle between two vectors from the cube to two fingers of the gripper. If the cube is located in the center of two fingers, the angle is 180 degrees and the auxiliary reward for fingers is 1.

Thus, the reward of *Pick-and-Lift* is given as

$$r = r_{reaching}(O_g, O_c) + r_{fingers}(O_c) + r_{grasping}(O_c) + r_{moving}(O_c, O_t) + r_{success}(O_c, O_t).$$

D.3 Heuristic information design

Heuristic information is a function of a state, $z_t = f(s_t)$. The function can be designed to include physical relationship between a robot and objects, and binary signals representing specific states. Using the reward components in the reward design, physical relationships are easily represented as

$$f_{distance}(O_i, O_j) = \sqrt{\left(pos(O_i) - pos(O_j)\right)^2},$$
$$f_{grasping}(O_i) = r_{grasping}(O_i),$$
$$f_{fingers}(O_i) = r_{fingers}(O_i),$$

where O_i and O_j can be the gripper, the cube or the target position.

In addition, to give additional physical information for the gripper, a function for a horizontal pose of the gripper is designed as

$$f_{gripper} = \cos\psi(g_v, u_v),$$

where g_v is a unit vector vertical to the gripper, u_v is a unit vector $[0\ 0\ 1]$, and $\psi(g_v, u_v)$ is an angle between the two vectors. If the gripper's pose is horizontal, g_v is equal to u_v and $f_{gripper}$ is equal to 1.

Based on the above physical information, we compose the heuristic information as

$$z_t = f(s_t) = \begin{bmatrix} f_{distance}(O_g, O_c) \\ f_{distance}(O_c, O_t) \\ f_{grasping}(O_c) \\ f_{fingers}(O_c) \\ f_{gripper} \end{bmatrix}. \tag{23}$$

Although we use the heuristic information with size 5 in the experiments, the heuristic information can be extended without limitation by defining and appending new physical information, such as angular positions and state indicators for the components in the environment.

D.4 Effect of a simple combination of heuristic information with SAC and DIAYN

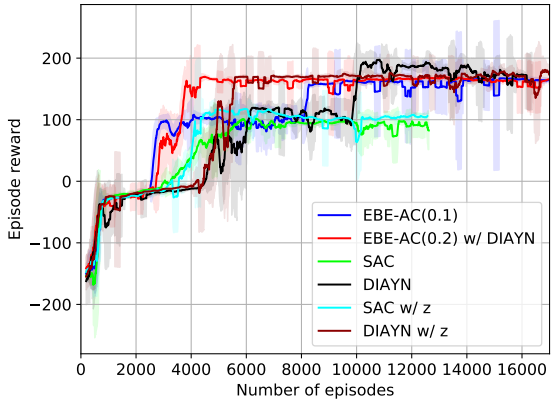


Figure 5: Episode rewards comparisons including SAC with z and DIAYN with z.

In addition to the experiments on our method, we conduct additional experiments on SAC with z and DIAYN with z to verify the performance of our method, where z is heuristic information. The methods with z use a concatenated input as $s' = (s,\ z)$, which means a simple combination of s with z.

Figure 5 represents episode rewards including SAC with z and DIAYN with z. As observed in the comparisons, the methods with z show faster convergences than the methods without z for SAC and DIAYN. Although the methods with z have the enhanced convergence speeds, our methods for EBE-AC(0.1) and EBE-AC(0.2) with DIAYN show faster convergences than the methods with z. Therefore, we can conclude that our methods are more sample-efficient than other methods, such as the simple combinations of SAC and DIAYN with heuristic information, because our methods use two types of policies and temperature optimization. In addition, we can say that our methods use heuristic information more efficiently.

D.5 Effect of temperature and probability for selecting policy with bounded exploration

We show additional results for EBE-AC methods according to the probability for selecting policy with bounded exploration $p_m(z) = \epsilon$ in Equation (5) and the temperature w_z for the entropy in Equation (1). In Figure 6, multiple ϵ in EBE-AC(multiple ϵ) means that ϵ is changed in evenly divided intervals for the total number of training steps. In addition, $w_z = 0$ means that the information theoretic reward does not consider the entropy of a_t for given s_t and z_t, but considers only the mutual information between a_t and z_t for given s_t.

As observed in Figure 6, EBE-AC(0.2) for $w_z = 0$ does not have good episode rewards. The temperature $w_z = 0$ affects to Equations (12) and (16). For the case of $w_z = 0$, the soft value

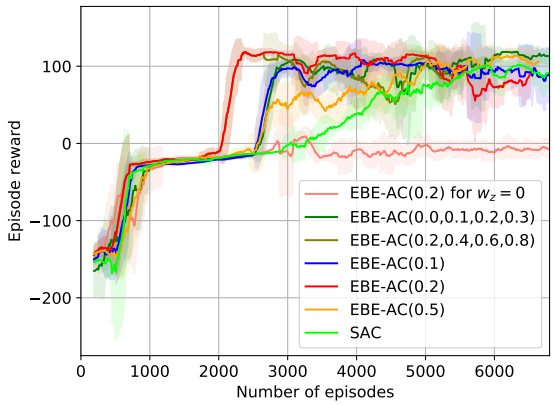


Figure 6: Episode rewards comparisons for EBE-AC methods according to ϵ and w_z.

function in Equation (12) has only a negative weighted entropy, and the constraint in Equation (16) is removed. Due to these modifications, the entropy of a_t for given s_t and z_t does not have lower bound. It means that the policy with z_t is not encouraged to explore action spaces, even though a_t has a dependency on z_t. Thus, only the consideration of the mutual information between a_t and z_t for given s_t can not guarantee good performances.

We have conducted additional experiments on our proposed method with different values of ϵ, as shown in Fig. 6, in order to obtain better performance for our proposed method. Empirically, we could gain the best performance for EBE-AC methods with the values $\epsilon = 0.1$ and 0.2 for EBE-AC and EBE-AC with DIAYN, respectively. We expect further improvements of the performance for our proposed method via an optimization for ϵ. Here, we note that our main contributions of this manuscript are the utilization of two types of policies with and without heuristic information and the derivation of dependency between the policies based on temperature optimization.

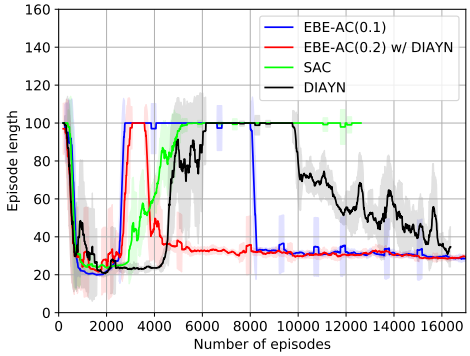
D.6 Additional results

Figure 7a represents a training curve for episode lengths. As shown in Figure 7a, each method has a different episode length over the number of episodes due to the terminate conditions[8]. The case of an episode length equal to 100 means the task failure. In other words, many training steps are spent in the episode and it leads to a decrease in opportunities to experience new episodes. Due to the decrease in opportunities to experience new episodes, SAC having the small number of episodes obtains low episode rewards as shown in Figure 2b.

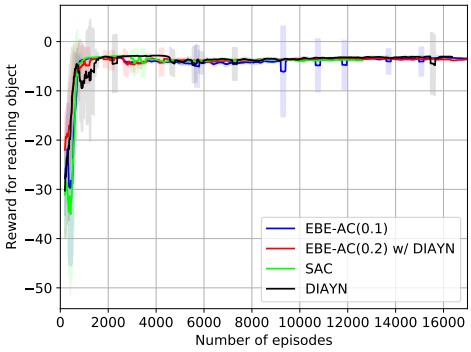
Figures 7b-7d represent training curves for three types of rewards, which are the components of the episode rewards[9]. As shown in Figure 7b, all the methods have good rewards for reaching the cube. However, the performances of the methods are represented differently in Figures 7b and 7c. Some methods including SAC maintain high rewards for grasping the cube, but these methods can not obtain the rewards for moving the cube to the target position. The high rewards for grasping means that the robot holds the cube in the gripper. This case is shown that these methods are stuck in local optimums. EBE-AC(0.2) with DIAYN, EBE-AC(0.1) and DIAYN do not maintain high rewards for grasping, but obtain the rewards for moving. It means that these methods find better optimums than those of the other methods. EBE-AC(0.2) with DIAYN and EBE-AC(0.1) pass the local optimums before DIAYN in order to pursue the success reward at the end of the episode. However, DIAYN spends more time to obtain the rewards for grasping as shown in Figure 7c. For this reason, EBE-AC(0.2) with DIAYN and EBE-AC(0.1) obtain slightly lower episode rewards than DIAYN in Figure 2a, but our methods reach much faster the success condition than DIAYN.

[8]The episode length means the number of steps per episode. For all methods, the sum of the episode lengths over the number of episodes are equal to the total number of steps for training

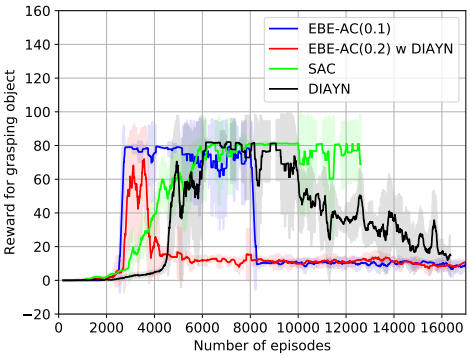
[9]Because the success condition is the state that the robot moves the cube to the target position, the reward for moving the cube to the target position include the huge success reward at the end of the episode.



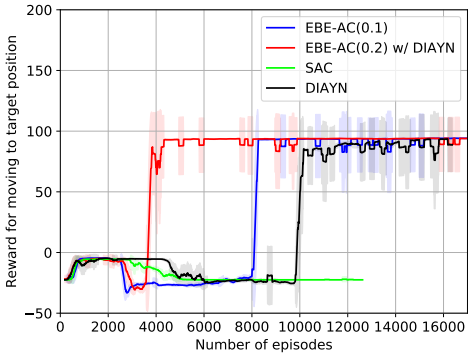
(a) Episode lengths



(b) Reward for reaching the cube



(c) Reward for grasping the cube



(d) Reward for moving the cube to the target position

Figure 7: Training curves for *Pick-and-Lift* according to the number of episodes.

For all Figures, EBE-AC(0.2) with DIAYN has lower variances than the other methods. Especially, EBE-AC(0.2) with DIAYN has very low variances in the rewards for grasping as shown in Figure 7c, which is designed by the sparse reward. It means that EBE-AC combined with DIAYN improve not only the sample efficiency but also the robustness.

E Additional Experiments: *Pick-and-Drag*

E.1 Task design

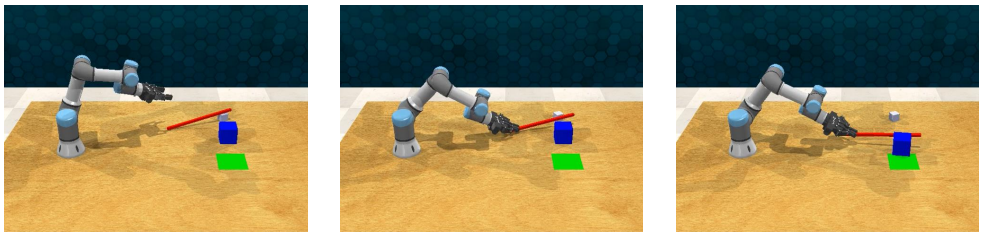
In order to validate our method, we present another example task "*Pick-and-Drag*", in which there are a single UR3 arm, a cube object, a stick, and a target position, as shown in Figure 8. A success condition of the task is "the cube is placed at the target position". To succeed the task, the robot arm should pick up the stick and drag the cube by using the stick, because the length of the arm is not sufficient to move the cube to the target position. A failure condition is defined as "the cube falls under the table" or "the robot drops the sticks on the table".

E.2 Reward design

We design the reward of *Pick-and-Drag* in the same manner as that of *Pick-and-Lift*. Thus, the reward of *Pick-and-Drag* is easily defined as

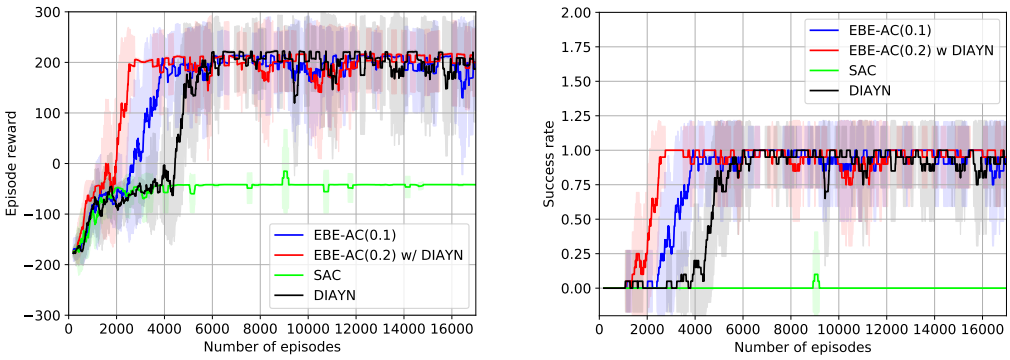
$$r = r_{reaching}(O_g, O_s) + r_{fingers}(O_s) + r_{grasping}(O_s) + r_{moving}(O_s, O_c) + r_{moving}(O_c, O_t) + r_{success}(O_c, O_t),$$

where O_s is the stick.



(a) reaching the stick (b) grasping the stick (c) dragging to the target position

Figure 8: *Pick-and-Drag*: it requires reaching out the gripper for the stick, grasping stably the stick, and dragging and exactly placing the cube to the target position by using the stick.



(a) Episode rewards (b) Success rates

Figure 9: Training curves for *Pick-and-Drag* according to the number of episodes.

E.3 Heuristic information design

We design the heuristic information of *Pick-and-Drag* in the same manner as that of *Pick-and-Lift*. Thus, the heuristic information of *Pick-and-Drag* is easily defined as

$$z_t = f(s_t) = \begin{bmatrix} f_{distance}(O_g, O_s) \\ f_{distance}(O_s, O_c) \\ f_{distance}(O_c, O_t) \\ f_{distance}(O_s, O_t) \\ f_{grasping}(O_s) \\ f_{fingers}(O_s) \\ f_{gripper} \end{bmatrix}. \tag{24}$$

E.4 Experimental results

Figure 9 represents training curves for episode rewards and success rates. In Figure 9a, a range of the episode reward in y-axis is roughly divided into three cases as follows:

- *Range 1*. (episode reward $<$ -100): The robot tries to reach the stick.
- *Range 2*. (-100 $<$ episode reward $<$ 0): The robot tries to grasp the stick.
- *Range 3*. (0 $<$ episode reward): The robot tries to move the cube to the target position by using the stick.

Compared with SAC and DIAYN, EBE-AC methods reach *Range 2* faster than the other methods. SAC can not reach *Range 3* and it means that SAC is stuck in local optimums *Range 2* during training. EBE-AC(0.1) and EBE-AC(0.2) with DIAYN methods reach *Range 3* faster than DIAYN. In *Range 3*, EBE-AC methods have lower variances than DIAYN so that we can say EBE-AC is

more robust than DIAYN. The fluctuations in *Range 3* are caused by task failures, in which the robot drops the stick while dragging the cube.

In Figure 9b which represents the success rates, it is also observed that the task failures in *Range 3* affect success rates and EBE-AC methods have better success rates than DIAYN. Compared to Figure 2b, the success rates in Figure 9b are slightly lower than those in Figure 2b, because *Pick-and-Drag* is more difficult than *Pick-and-Lift* due to the interaction between the stick and cube.

Through the additional experimental results, we confirmed that the EBE-AC methods outperform the baselines in terms of the sample efficiency for not only *Pick-and-Lift* but also *Pick-and-Drag*.