Learning from Pixels with Expert Observations

Minh-Huy Hoang[1*], Long Dinh[2*], Hai Nguyen[3†]

Abstract— In reinforcement learning (RL), sparse rewards can present a significant challenge. Fortunately, expert actions can be utilized to overcome this issue. However, acquiring explicit expert actions can be costly, and expert observations are often more readily available. This paper presents a new approach that uses expert observations for learning in robot manipulation tasks with sparse rewards from pixel observations. Specifically, our technique involves using expert observations as intermediate visual goals for a goal-conditioned RL agent, enabling it to complete a task by successively reaching a series of goals. We demonstrate the efficacy of our method in five challenging block construction tasks in simulation and show that when combined with two state-of-the-art agents, our approach can significantly improve their performance while requiring 4-20 times fewer expert actions during training. Moreover, our method is also superior to a hierarchical baseline.

I. INTRODUCTION

Learning from observations without explicit guidance from an expert is an essential learning mechanism. Often, humans lack precise knowledge of the actions taken by a human expert, but they can deduce these actions by attempting to reproduce the expert's observations. For example, in Figure 1, the task is to construct a desired structure consisting of a triangle on top of a stack of three blocks. At the current state s_0 when all blocks are on the ground, the optimal move is to form a stack of two blocks to reach the next state s_1, which closely resembles the scene an expert would observe at the subsequent step (g). Forming the stack should be much simpler than building the final structure, given only a pick and a place actions are needed. This example shows that an intermediate visual goal provided by expert observations can break down a complex task into more manageable subtasks and facilitate step-by-step task achievement.

This paper presents a hierarchical agent consisting of a *fixed* top level and a *learned* bottom level. At the top level, we use the indices of expert observations within an expert episode as the *abstract goals* $\bar{g}$ (see Figure 1) for the bottom level to achieve. The bottom level implements a goal-conditioned policy $\pi(s, \bar{g})$ that gradually realizes these abstract goals until the task is completed. The goal achievement is determined by comparing the *abstract state* $\bar{s}$ produced by a *state abstractor* $\mathcal{C}(s)$ with the corresponding abstract goal $\bar{g}$. The state abstractor is a multi-class classifier pre-trained using supervised learning with expert transitions.

This paper proposes a novel approach to improve the performance of current state-of-the-art methods in the setting

[*]Equal contribution,[1] University of Science, Ho Chi Minh City, Vietnam [2]Hanoi University of Science & Technology, Hanoi, Vietnam. [3]Northeastern University, Boston, MA 02115, USA. [†]Corresponding author nguyen.hai1@northeastern.edu.

Fig. 1. We utilize expert observations as intermediate visual goals to train a goal-conditioned policy. Using our approach, achieving a hard task (*e.g.*, building a structure) equals achieving a series of easier goals sequentially.

of block construction tasks with sparse rewards and pixel-based observations. Specifically, we apply our approach to a version of Deep Q-Network (DQN) [1] and Strict DQN from Demonstrations (SDQfD) [2] and demonstrate substantial performance improvements in five challenging tasks. Our approach relies on fewer expert actions and utilizes expert observations as intermediate visual goals to create a simple yet effective learning strategy. Moreover, we also encode the domain symmetries into the state abstractor and the RL agent to increase the performance further. Our results indicate that our method outperforms the baselines, including a strengthened version of h-DQN [3], with 4-20 times fewer expert actions. Additional information including videos and code be found at `https://sites.google.com/view/leo-rb`.

II. RELATED WORK

Goal-conditioned Reinforcement Learning is a popular paradigm in which an RL agent seeks to achieve a goal, which may be a specific state (*e.g.*, moving a block to a specific position) or an abstract concept (*e.g.*, building a desired structure as in Figure 1). For more efficiency, goals are often diversified. Most notably, Hindsight Experience Replay [4] (HER) *relabels* previously visited states as goals. Instead of choosing goals randomly from past states like HER, [5] selects goals from previously visited states based on how close they are to the original goal. However, in many tasks such as the one in Figure 1, useful goals may be difficult to visit by random actions, making relabelling unhelpful. In this work, we proposed to use expert observations as goals. Recent work in this area includes [6], which uses visual goals to navigate the open world, [7], which chooses goals from successful demonstration trajectories, and [8], which employs a hand-coded abstraction function and goal

graph for selecting goals to learn a multi-task policy in block construction domains. Our approach differs because we automatically propose abstract goals (using a learned state abstractor, not a hand-coded one like [8]) instead of explicit ones selected from previous states like [6], [7] and operate in a single-task setting unlike [8]. Moreover, we encode domain symmetries in our agent to enhance efficiency.

Hierarchical Reinforcement Learning (HRL) can break down a long-horizon RL task into smaller subtasks, which are more manageable. HRL has demonstrated superior performance over non-hierarchical approaches in various tasks [3], [9], [10]. Furthermore, [11], [12] utilize imitation learning to improve the hierarchical subgoal selection process to achieve better performance. Our approach is similar to h-DQN [3], which involves concurrently training two DQN [1] agents. However, instead of learning two levels in parallel, which often causes instability, *we only learn the bottom level*. At the top level, we use expert observations as goals for the bottom level to achieve, which we found to be more effective.

Equivariant Neural Networks can encode chosen symmetries within their structures, greatly enhancing generalization and sample efficiency. The idea was introduced by G-Convolution [13] and Steerable Convolutional Neural Networks (CNN) [14] and has since been utilized in computer vision [15], [16] and reinforcement learning [17]. In particular, earlier works [2], [18], [19], [20] have successfully applied equivariant networks to robot manipulation. We also found that using equivariant networks for our RL agent and state abstractor can increase task performance.

Block Construction Tasks. In the context of these challenging tasks, unlike other methods, such as object-factored models used in [21] and [22], our approach learns directly from image observations and sparse rewards. Additionally, [23], [24] perform model-based planning in a learned latent space to tackle block construction tasks from pixels. Recently, [25] proposed a new manipulation benchmark involving picking and placing objects of complex geometries, adopted by several [26], [27]. However, in this work, we consider block construction tasks of regular shapes, taken from the BulletArm benchmark [28].

III. BACKGROUND

A. Goal-conditioned DQN

A goal-conditioned Markov decision process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{G}, T, R)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the *discrete* action space, $\mathcal{G}$ is the goal space, $T(s, a, s') = p(s' \mid s, a)$ is the transition function that gives the probability of reaching state s' given an action $a \in \mathcal{A}$ is taken in state s, $R(s, a, s', g)$ is the reward function. The objective is to find a goal-conditioned policy $\pi(s, g)$ that maximizes the expected discounted return $\mathbb{E}_\pi[\sum_{t=0}^\infty \gamma^t r_t \mid s_0, g]$ with a discount factor $\gamma \in [0, 1)$, starting with initial state s_0 and goal g. In contrast to DQN [1], which learns the optimal Q-function using transitions of the form (s, a, s', r), the optimal goal-conditioned Q-function $Q^*(s, a, g)$ is learned using transitions of the

form (s, a, s', r, g) [29]. From $Q^*(s, a, g)$, we can obtain an optimal goal-conditioned policy $\pi^*(s, g)$:

$$\pi^*(s, g) = \operatorname{argmax}_a Q^*(s, a, g). \tag{1}$$

B. Group Equivariance and Invariance

Given a symmetry group G and an element $\alpha \in G$, a function $f : X \to Y$ is *equivariant* if $f(\alpha x) = \alpha f(x)$, and *invariant* if $f(\alpha x) = f(x)$. More precisely, how α acts on X or Y depends on the *representation* ρ of the symmetry group G. For instance, if G is a group of planar rotations and x is a single-channel image, then $\alpha x = \rho(\alpha)x$ will be a multiplication between a rotation matrix $\rho(\alpha)$ and an image x, resulting in a rotated image. However, for clarity, in the remainder of the paper, we use $\alpha x = \rho(\alpha)x$ to denote the action of α on x without referring to any specific representation ρ. For more details about group representations, please refer to [30].

C. Group-invariant MDP in SE(2)

Many manipulation tasks of rigid blocks from top-down images [28] have the symmetry about the rotation and translation in plane $\mathbb{R}^2$ (*i.e.*, spanning the special Euclidean group SE(2)). Such tasks can be defined as a group-invariant MDP [31], which is invariant in SE(2) by satisfying the following conditions for all group element $\alpha \in$ SE(2):

- Reward Invariance:

$$R(s, a, s') = R(\alpha s, \alpha a, \alpha s'). \tag{2}$$

- Transition Invariance:

$$T(s, a, s') = T(\alpha s, \alpha a, \alpha s'). \tag{3}$$

The key feature of a group-invariant MDP is that its optimal Q-function is invariant, *i.e.*, $Q^*(s, a) = Q^*(\alpha s, \alpha a)$ for all $\alpha \in$ SE(2). This invariance property is the foundation to build very sample-efficient RL agents [18], [2], [31] by directly encoding the property into the Q-network structure.

D. Strict DQfD (SDQfD)

Besides fitting the Q-function $Q(s, a)$ similar to DQN [1], DQfD [32] leverages expert demonstrations by *permanently* storing the expert transitions inside the replay buffer. Specifically, DQfD additionally minimizes a margin loss to force the value of non-expert actions to be below the value of expert actions given the same state:

$$\mathcal{L}_M = \mathbb{E}_{s, a^e}[\max_a(Q(s, a) + l(a, a^e)) - Q(s, a^e)], \tag{4}$$

where a^e is an expert action in state s and $l(a, a^e)$ is penalty that is positive when a is non-expert and is 0 otherwise.

In *large* action spaces, SDQfD [2] utilizes a similar penalty but applies it to *all* non-expert actions that have Q-values higher than that of an expert action (given the same state) minus the penalty. The replacement of $\mathcal{L}_M$ is a new *strict* margin loss defined as:

$$\mathcal{L}_{SM} = \frac{1}{|A^{s, a^e}|} \sum_{a \in A^{s, a^e}} [Q(s, a) + l(a, a^e) - Q(s, a^e)], \tag{5}$$

where A^{s,a^e} is set of actions satisfied:

$$A^{s,a^e} = \{a \in \mathcal{A} \mid Q(s,a) + l(a,a^e) > Q(s,a^e)\}. \quad (6)$$

SDQfD considerably outperformed both DQfD and DQN with demonstrations in many block construction tasks with sparse rewards [2].

IV. STRUCTURE CONSTRUCTION FROM PIXELS

Fig. 2. A visual description of our block construction tasks and a list of five tasks considered for experiments.

TABLE I
MORE DETAILS ON TASKS.

Task	BS	HB1	HB2	HB3	HB4
Number of Blocks	4	4	3	4	6
Number of Optimal Steps	6	6	4	6	10
Max Number of Steps	10	10	10	10	20

Figure 2 shows five structure construction tasks in the BulletArm benchmark [28], namely Block-Stacking (`BS`) and House-Building-i (`HBi`) with $i = \{1,2,3,4\}$. In these tasks, a robot arm must construct a desired structure from unstructured blocks using top-down depth images of the scene taken by a camera on top (see an example task in Figure 2). At the beginning of each episode, the agent is presented with necessary blocks with random poses to build the desired structure (see Table I for more task details).

State Space. We consider a state $s = (I, H, k)$, where $I \in \mathbb{R}^{128 \times 128}$ is a top-down depth image of the scene, $H \in \mathbb{R}^{24 \times 24}$ is an in-hand image denoting the current object in the gripper, and $k \in \{\texttt{holding}, \texttt{empty}\}$ refers to the current grasping status of the gripper. The in-hand image H is an orthographic projection of the partial point cloud where the last pick occurred (see Figure 2).

Action Space. An action $a = (p, x, y, \theta)$ where $p \in \{\texttt{pick}, \texttt{place}\}$, (x,y) are the pixel coordinate in I that the agent wants to perform a pick or a place, and $\theta \in \{0, \frac{\pi}{16}, \frac{2\pi}{16}, \ldots, \frac{31\pi}{16}\}$ is the discretized rotation of the gripper around the z axis. The value of p decides whether to close the fingers (pick) or open the fingers (place). In this setting, the action space is *large*, *e.g.*, there are $2 \times 128 \times 128 \times 32 = 1,048,576$ possible discrete actions.

Reward. The agent is given a sparse reward $r = 1.0$ if accomplishing the desired structure, else $r = 0.0$.

Translation and Rotation in SE(2). Given an arbitrary rotation and translation in the plane $\alpha \in$ SE(2), here we define how it would operate on a state s and an action a. First, α operates on $s = (I, H, k)$ by translating and rotating I but leaving H and k unchanged:

$$\alpha s = (\alpha I, H, k). \quad (7)$$

For an action $a = (p, x, y, \theta)$, it rotates and translates the spatial components $(x, y, \theta) \in$ SE(2) but leaving p unchanged:

$$\alpha a = (p, \alpha x, \alpha y, \alpha \theta). \quad (8)$$

Fig. 3. $0 \rightarrow 6$: A deconstruction planner decomposes a fully built structure by randomly picking the highest block and placing it on the ground until the structure is fully decomposed. Reverting $6 \rightarrow 0$ results in a demonstration episode. Besides indexing, numbers are also used for abstract states.

Generating Expert Demonstrations. To collect expert demonstrations, we utilize a simple deconstruction planner similar to [28], [2], illustrated in Figure 3. The process starts with the desired structure fully built; then, the highest object is picked and randomly placed on the ground until the structure is fully decomposed. We then reverse the trajectory to obtain an expert episode. This process allows us to efficiently generate expert episodes without requiring task-specific expert policies, which would be difficult to obtain.

V. LEARNING WITH EXPERT OBSERVATIONS

A. Formulating as a Goal-Conditioned MDP

We formulate the problem of building a structure with N blocks as a goal-conditioned MDP $M_G = (\mathcal{S}, \mathcal{A}, \bar{\mathcal{G}}, T, R_g)$, where $\mathcal{S}$, $\mathcal{A}$, and T are the same as in the original problem. The differences are the goal space and the reward function:

$$\bar{\mathcal{G}} = \{0, 1, \ldots, 2N - 2\} \quad (9)$$

$$R_g(s, a, s', \bar{g}) = \begin{cases} r, & \text{if } \mathbb{1}(\bar{s}', \bar{g}) = 1 \\ r - 1, & \text{otherwise}, \end{cases} \quad (10)$$

where $\bar{\mathcal{G}}$ is an abstract goal space, $\bar{g} \in \bar{\mathcal{G}}$ is an abstract goal, $\mathbb{1}(.)$ is the indicator function, r is the environment reward (see Section IV), and $\bar{s}'$ is the abstract state of s'. Notice that besides rewarding 0 when the abstract goal is achieved and -1 otherwise, our reward function gives the agent more reward when the final structure is achieved successfully, *i.e.*, when $r = 1.0$. This additional reward is to emphasize the

importance of achieving the desired structure. Moreover, the indices in $\bar{\mathcal{G}}$ are exactly the indices of expert observations for the task (see Figure 3 for the task HB1).

Fig. 4. The agent starts at a state s where all the blocks are on the ground. In this state, the abstract state is $\bar{s} = 6$, as illustrated in Figure 3. To construct the desired structure in the middle, the agent must transition to the next state, s', which should resemble the scene numbered 5 in Figure 3, where the agent picks up a red block. Therefore, the abstract goal is $\bar{g} = \bar{s}' = \bar{s} - 1$. If the agent succeeds in achieving this goal in the following step, it will be supplied with another goal. Otherwise, the environment will be reset.

To make the idea more concrete, Figure 4 shows how our agent tackles the HB1 task. Initially, the agent is in state s, where all the blocks are on the ground. This state corresponds to an abstract state $\bar{s} = 6$, as depicted in Figure 3. To build the desired structure, the agent must transition to the next state s' where it picks up a red block. This next state should resemble the scene labeled as number 5 in Figure 3, making the abstract goal $\bar{g} = \bar{s}' = \bar{s} - 1 = 5$. Since s and the desired s' are only one pick action away in Figure 3, the agent should only need a single timestep to reach s' from s. Therefore, if the agent achieves the abstract goal in the next timestep, it is presented with another goal; otherwise, we reset the episode. In the ideal case, the process continues until the desired structure is fully built, *i.e.*, when the abstract goal $\bar{g} = 0$ is achieved.

To facilitate such a learning process, our agent comprises the following components:

- A state abstractor $\mathcal{C}(s)$:

$$\mathcal{C}(s) : \mathcal{S} \to \bar{\mathcal{S}}, \qquad (11)$$

  where $\bar{\mathcal{S}} \equiv \bar{\mathcal{G}}$.
- An *implicit* goal-conditioned policy extracted from a Q-function:

$$\pi(s, \bar{g}) = \operatorname{argmax}_a Q(s, a, \bar{g}). \qquad (12)$$

We also note that our formulation can be easily extended to accommodate a more general goal space $\mathcal{G}$. In such cases, the goals can be actual expert observations, such as images, rather than abstract states representing those observations.

B. State Abstractor

$\mathcal{C}(s)$ can be considered as a multi-class classifier that maps each state s to its corresponding abstract state $\bar{s}$. It is responsible for determining goal achievement for our agent. To train it, we use a dataset $\mathcal{D}_{train}$ of pairs $(s, \bar{s})$ collected

from expert demonstrations. As shown in Figure 3, for each state s in the expert episode, we use the index of s inside the episode to be $\bar{s}$. Given $\mathcal{D}_{train}$, $\mathcal{C}(s)$ is trained by minimizing the following cross-entropy loss:

$$\mathcal{L}_{CE} = -\frac{1}{|\mathcal{D}_{train}|} \sum_{i=1}^{|\mathcal{D}_{train}|} \sum_{j=0}^{2N-2} \mathbb{1}[\bar{s}_i = j] \log \mathcal{C}(s_i)_j, \quad (13)$$

where $|\mathcal{D}_{train}|$ denotes the size of $\mathcal{D}_{train}$. For each given task, we pre-train $\mathcal{C}(s)$ and *freeze* it to use for checking goal achievement. In addition, given a group element $\alpha \in \mathrm{SE}(2)$, then $\mathcal{C}(s) = \mathcal{C}(\alpha s)$, *i.e.*, the abstract state should be the same if we rotate and translate the state (*e.g.*, top-down image). Therefore, we construct it as an invariant function.

C. Learning $Q(s, a, \bar{g})$

Augmented state representation (ASR) [2] is a powerful method for learning the Q-function in domains with large action spaces, as in our tasks. The idea is to transform an MDP with a large action space into a new one with more states but fewer actions. We also adopt ASR to learn $Q(s, a, \bar{g})$ but modify it for our goal-conditioned MDP.

1) Transition Division: We first divide the action into three components: the *pick/place* component: p, the *position* component: $a_{xy} = (x, y)$, and the *rotation* component: θ. Now considering that the agent wants to achieve an abstract goal $\bar{g}$ at state s by deciding the next action $a_m = (p_m, x_m, y_m, \theta_m)$. Depending on the grasping status k, whether to pick or place p_m can be chosen using simple logic: we pick when the gripper is currently empty and place when it is holding something, *i.e.*,

$$p_m = \begin{cases} \texttt{pick}, & \text{if } k = \texttt{empty} \\ \texttt{place}, & \text{if } k = \texttt{holding} \end{cases} \qquad (14)$$

For selecting (x_m, y_m, θ_m), we need to learn $Q(s, a, \bar{g})$. ASR does that by dividing the *original* transition, which is

$$s, \bar{g} \xrightarrow{a} s', \qquad (15)$$

into two transitions. The position transition only involves a_{xy}:

$$s, \bar{g} \xrightarrow{a_{xy}} \tilde{s}, \qquad (16)$$

where $\tilde{s} \in \mathcal{S}$ can be considered as the resulting next state of the original transition after taking action $a = (p, a_{xy}, 0)$. The rotation transition only involves θ:

$$\tilde{s}, \bar{g} \xrightarrow{\theta} s'. \qquad (17)$$

2) Learning Factorial Q-functions: By the transition division, learning $Q(s, a, \bar{g})$ can be effectively performed by learning a position Q-function $Q_{xy}(s, a_{xy}, \bar{g})$ and a rotation Q-function $Q_\theta(\tilde{s}, \theta, \bar{g})$.

- Q_{xy} decides where we should perform pick/place in the image. As shown in Figure 5, it outputs *two* position maps with the same size as I's, corresponding for $p_m =$ pick and $p_m =$ place. These two maps represent the Q-values of picking or placing at a certain pixel position concerning the current abstract goal $\bar{g}$. Now,

Fig. 5. Learning $Q(s, a, \bar{g})$ in HB4 using goal-conditioned ASR [2]. Learning $Q(s, a, \bar{g})$ is divided into learning a position Q-function Q_{xy} and a rotation Q-function Q_θ. Picking or placing (p_m) is chosen based on the grasping status k. The red dots denote the selected pixel location to pick/place (x_m, y_m) using Q_{xy} and the subsequently selected rotation θ_m using Q_θ. Q_θ decides θ_m using a H and a small patch P around (x_m, y_m) of I.

during exploitation, the position to perform a pick/place can be chosen greedily:

$$(x_m, y_m) = \text{argmax}_{a_{xy}} Q_{xy}(s, a_{xy}, \bar{g}). \quad (18)$$

- Q_θ decides the rotation θ_m of the gripper when we have decided where to pick/place. To do that, ASR only considers a small patch P of I, centered around (x_m, y_m). Instead of learning the original Q_θ, we learn $Q_\theta(H, P, \theta, \bar{g})$, which should be sufficient to decide the best θ_m given the information contained in H and P. During exploitation, we simply greedily select the orientation using a rotation map outputted by Q_θ:

$$\theta_m = \text{argmax}_\theta Q_\theta(H, P, \theta, \bar{g}). \quad (19)$$

3) Invariant Factorial Q-functions: We show that Q_{xy} and Q_θ are invariant so that they can be constructed as invariant models. Following Section III-C, we will show that the two Q-functions correspond to group-invariant MDPs with invariant reward and transition functions.

From Equation (16), we can observe that the Q-function $Q_{xy}(s, a_{xy}, \bar{g})$ corresponds to the original goal-conditioned MDP M_G except that the action space is constrained by only allowing $\theta = 0$. We convert M_G to be a traditional MDP by using an augmented state $(s, \bar{g})$, which changes the reward function of M_G into $R'_g((s, \bar{g}), a_{xy}, (s', \bar{g}))$. The new reward function is invariant because achieving $\bar{g}$ should be independent of the position and orientation of the scene. Moreover, the transition is invariant because, in the task setting, the outcome of an action should be invariant when the scene and the action rotate and translate with the same amount. Therefore, Q_{xy} is invariant.

From Equation (17), because $\tilde{s} \in \mathcal{S}$, the orientation transition $\tilde{s}, \bar{g} \xrightarrow{\theta} s'$ correspond to a transition in M_G except that now with actions being the rotations θ only. Therefore, the same above logic can apply here with an augmented state $(\tilde{s}, \bar{g})$ to show that Q_θ is also invariant.

VI. EXPERIMENTS AND RESULTS

In the five structures considered, we want to investigate whether our approach of learning from expert observations (LEO) can help improve the performance of existing best RL agents [28] for our tasks, namely, DQN and SDQfD.

A. Agents

To investigate the possible benefits of our approach on DQN and SDQfD, we consider the following agents:

- DQN/SDQfD-x [28]: These are baselines that use x demonstration episodes (*with expert actions*) by permanently storing them in the replay buffer. Note that SDQfD better uses the demonstration using the margin loss (see Equation (5)). Both employ equivariant ASR [18] and are currently state-of-the-art agents for our tasks (see [28]).
- LEO-DQN/SDQfD-x (Ours): The versions of DQN/SDQfD-x that use our approach (LEO-). We also benchmark LEO-based agents' performance using non-invariant state abstractors (Non-Inv).
- h-DQN-100, h-SDQfD-100 are two-level hierarchical agents based on h-DQN [3] agent that learn both levels concurrently. The top learns to produce abstract goals, and the bottom attempts to achieve them. Like our agent, the bottom uses the same pre-trained $\mathcal{C}(s)$ to determine goal achievement. We consider two versions of these agents. The original version (Original) follows the original reward scheme by only using the environment rewards for the top level. We improve the original agents (Modified) by additionally rewarding the top with 0 and -1 if the bottom can or cannot achieve its abstract goals. Furthermore, to stabilize training, we follow the practice in [3] to pre-train the bottom level with random abstract goals before training the two levels in parallel. It is also worth noting that these agents also use equivariant ASR and 100 demonstration episodes.

B. Evaluation Metrics

We compare methods using the *success rates of an evaluation agent*, which is evaluated for every 500 training (environment) steps. We train all agents in 25,000 environment steps in HB4 (the hardest task) and in 10,000 environment steps in the four remaining structures. The reported results are averages of 4 seeds with shaded one standard error.

C. Learning Curves

We visualize all learning curves in Figure 6, where the top row shows DQN-based agents and the bottom shows SDQfD-based agents. Given the same demonstration episodes, LEO-

Fig. 6. Evaluation success rate averaged over four seeds with shaded one standard error. The top row (grey background) contains DQN-based agents, and the bottom row (white background) contains SDQfD-based agents. Note that all agents utilize equivariant ASR [18].

based agents outperform the original agents regarding final performance and learning speed. The superiority is consistent over all tasks and all algorithms (DQN/SDQfD) tested. Moreover, given more (100) expert episodes, the original and hierarchical agents are still outperformed by our agents with 4-20 times fewer expert actions used, *e.g.*, our agents only use 5, 15, or 25 expert episodes. The figure also indicates that using an invariant state abstractor (solid green lines) benefits our agents more than not (dotted green lines) in all cases (except for DQN-based agents in BS), regardless of the algorithms used. Finally, the original versions of h-DQN and h-SDQfD perform much worse than our modified version. They fail to learn in all tasks, probably due to a lack of positive environment rewards for the top level to learn. In contrast, with the modified reward function by rewarding/penalizing the top when it produces achieved/unachieved abstract goals, the modified version performs much better in all tasks.

D. Learning with No Expert Actions

We hypothesize that our approach can benefit learning in specific tasks even when no expert actions are given. To verify this hypothesis, we compare DQN and LEO-DQN in HB1 and HB2 with *no* expert actions (we exclude SDQfD because expert actions are required to implement it, see Equation (5)). Figure 7 shows that LEO-DQN-0 agents (solid orange), even though they perform worse than LEO-DQN-5 (as expected with fewer expert episodes), can still greatly outperform DQN-0 agents (dotted orange lines). Specifically, with no expert actions provided, DQN-0 agents fail to learn both tasks, while LEO-DQN-5 can nearly master the two tasks at the end of training.

Fig. 7. Performance in HB1 and HB2 when no expert actions are used.

E. Implementation Details

Our code is implemented in PyTorch [33], and all optimization uses the Adam [34] optimizer. Below we only highlight key implementation for Q-functions and the state abstractor (see our website for their detailed diagrams).

Discrete Approximation of SE(2). In practice, it is common to approximate the infinite group SE(2) with finite groups [30], [35]. Follow the previous work [2], we approximate SE(2) with $\hat{SE}(2)$, which is the cross product between $\{1, \ldots, 128\} \times \{1, \ldots, 128\} \subset \mathbb{Z}^2$ and the cyclic group $C_{32} = \{k\pi/16 : 0 \leq k < 32, k \in \mathbb{Z}\}$.

Q-functions of all agents are constructed by equivariant ASR networks [2], implemented using the e2cnn [30] library. Non-equivariant inputs, such as in-hand images and abstract goals, are integrated into our agents using Dynamic Filter [18]. We use the dihedral group D_4 for Q_{xy} and C_{32} group for Q_θ, where D_4 is $C_4 = \{k\pi/2 : 0 \leq k < 4, k \in \mathbb{Z}\}$ combined with the reflection symmetry. All agents are trained with a learning rate of 1e-4. The batch size is set to 32, and the discount is 0.95. We run five simulated environments in parallel to collect transitions for training. The size of the cropped patch P is 24×24 pixels.

State Abstractor uses two branches to extract features from the top-down and the in-hand images. The branch for top-down images is a series of five [Conv+Max-Pool] modules. The branch for in-hand images only consists of three such modules. The extracted features are then concatenated before going through three fully-connected layers and a softmax layer at the end. The invariant state abstractor replaces the traditional CNNs with equivariant CNNs using the group D_4. For each task, we train our state abstractors in 12,000 training steps with a batch size of 32 using a learning rate of 1e-3. The number of samples per class for training $\mathcal{C}(s)$ in BS and HB1 is 250, in HB2 and HB3 is 500, and in HB4 is 1000.

VII. CONCLUSIONS

This paper showed that expert observations could be beneficial learning signals when expert actions are only sparsely provided. Even though we only demonstrate the improved performance on DQN and SDQfD, we project that our

approach should benefit other off-policy learning algorithms. The limitation of our approach is that the state abstractor, which acts similarly to human eyes, must be sufficiently good, ideally trained from *optimal* experts. Moreover, while we have used equivariant neural networks to improve the efficiency and generalization of our state abstractor, it cannot currently handle outliers [36], which can be addressed using approaches [37], [38] that equip classifiers with such an ability. Recognizing outliers might lead to more sample efficiency as the state abstractor can signal a reset to an environment when the agent is in bad and/or unrecoverable states. Nevertheless, although our state abstractor is trained using the in-distribution data from expert demonstrations, it still helps LEO-based agents perform well.

REFERENCES

[1] 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.

[2] D. Wang, C. Kohler, and R. Platt, "Policy learning in se (3) action spaces," *arXiv preprint arXiv:2010.02798*, 2020.

[3] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, 2016.

[4] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.

[5] H. Nguyen, H. M. La, and M. Deans, "Hindsight experience replay with experience ranking," in *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. IEEE, 2019, pp. 1–6.

[6] D. Shah, B. Eysenbach, G. Kahn, N. Rhinehart, and S. Levine, "Ving: Learning open-world navigation with visual goals," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 13 215–13 222.

[7] T. Davchev, O. Sushkov, J.-B. Regli, S. Schaal, Y. Aytar, M. Wulfmeier, and J. Scholz, "Wish you were here: Hindsight goal selection for long-horizon dexterous manipulation," *arXiv preprint arXiv:2112.00597*, 2021.

[8] D. Klee, O. Biza, and R. Platt, "Graph-structured policy learning for multi-goal manipulation tasks," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4765–4772.

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

[10] H. Nguyen, Z. Yang, A. Baisero, X. Ma, R. Platt, and C. Amato, "Hierarchical reinforcement learning under mixed observability," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2022, pp. 188–204.

[11] F. Xie, A. Chowdhury, M. De Paolis Kaluza, L. Zhao, L. Wong, and R. Yu, "Deep imitation learning for bimanual robotic manipulation," *Advances in neural information processing systems*, vol. 33, pp. 2327–2337, 2020.

[12] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," *arXiv preprint arXiv:1910.11956*, 2019.

[13] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *International conference on machine learning*. PMLR, 2016, pp. 2990–2999.

[14] T. S. Cohen and M. Welling, "Steerable cnns," *arXiv preprint arXiv:1612.08498*, 2016.

[15] O. Puny, M. Atzmon, H. Ben-Hamu, E. J. Smith, I. Misra, A. Grover, and Y. Lipman, "Frame averaging for invariant and equivariant network design," *arXiv preprint arXiv:2110.03336*, 2021.

[16] C. Esteves, A. Makadia, and K. Daniilidis, "Spin-weighted spherical cnns," *Advances in Neural Information Processing Systems*, vol. 33, pp. 8614–8625, 2020.

[17] E. van der Pol, D. Worrall, H. van Hoof, F. Oliehoek, and M. Welling, "Mdp homomorphic networks: Group symmetries in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4199–4210, 2020.

[18] D. Wang, R. Walters, X. Zhu, and R. Platt, "Equivariant q learning in spatial action spaces," in *Conference on Robot Learning*. PMLR, 2022, pp. 1713–1723.

[19] X. Zhu, D. Wang, O. Biza, G. Su, R. Walters, and R. Platt, "Sample efficient grasp learning using equivariant models," *arXiv preprint arXiv:2202.09468*, 2022.

[20] A. Simeonov, Y. Du, A. Tagliasacchi, J. B. Tenenbaum, A. Rodriguez, P. Agrawal, and V. Sitzmann, "Neural descriptor fields: Se (3)-equivariant object representations for manipulation," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 6394–6400.

[21] R. Li, A. Jabri, T. Darrell, and P. Agrawal, "Towards practical multi-object manipulation using relational reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4051–4058.

[22] N. Funk, G. Chalvatzaki, B. Belousov, and J. Peters, "Learn2assemble with structured representations and search for robotic architectural construction," in *Conference on Robot Learning*. PMLR, 2022, pp. 1401–1411.

[23] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine, "Visual reinforcement learning with imagined goals," *Advances in neural information processing systems*, vol. 31, 2018.

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

[25] A. X. Lee, C. M. Devin, Y. Zhou, T. Lampe, K. Bousmalis, J. T. Springenberg, A. Byravan, A. Abdolmaleki, N. Gileadi, D. Khosid, *et al.*, "Beyond pick-and-place: Tackling robotic stacking of diverse shapes," in *5th Annual Conference on Robot Learning*, 2021.

[26] A. X. Lee, C. Devin, J. T. Springenberg, Y. Zhou, T. Lampe, A. Abdolmaleki, and K. Bousmalis, "How to spend your robot time: Bridging kickstarting and offline reinforcement learning for vision-based robotic manipulation," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 2468–2475.

[27] Y. Zhou, Y. Aytar, and K. Bousmalis, "Manipulator-independent representations for visual imitation," *arXiv preprint arXiv:2103.09016*, 2021.

[28] D. Wang, C. Kohler, X. Zhu, M. Jia, and R. Platt, "Bulletarm: An open-source robotic manipulation benchmark and learning framework," in *The International Symposium of Robotics Research*. Springer, 2022, pp. 335–350.

[29] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International conference on machine learning*. PMLR, 2015, pp. 1312–1320.

[30] M. Weiler and G. Cesa, "General e (2)-equivariant steerable cnns," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[31] D. Wang and R. Walters, "So (2) equivariant reinforcement learning," in *International Conference on Learning Representations*, 2022.

[32] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, *et al.*, "Deep q-learning from demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[35] T. S. Cohen and M. Welling, "Steerable cnns," *arXiv preprint arXiv:1612.08498*, 2016.

[36] J. Yang, K. Zhou, Y. Li, and Z. Liu, "Generalized out-of-distribution detection: A survey," *arXiv preprint arXiv:2110.11334*, 2021.

[37] G. Chen, L. Qiao, Y. Shi, P. Peng, J. Li, T. Huang, S. Pu, and Y. Tian, "Learning open set network with discriminative reciprocal points," in *ECCV*, 2020.

[38] B. Liu, H. Kang, H. Li, G. Hua, and N. Vasconcelos, "Few-shot open-set recognition using meta-learning," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8795–8804, 2020.