XIRL: Cross-embodiment Inverse Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: We investigate the visual *cross-embodiment* imitation setting, in which agents learn policies from videos of other agents (such as humans) demonstrating the same task, but with stark differences in their *embodiments* – shape, actions, end-effector dynamics, etc. In this work, we demonstrate that it is possible to automatically discover and learn vision-based reward functions from cross-embodiment demonstration videos that are robust to these differences. Specifically, we present a self-supervised method for Cross-embodiment Inverse Reinforcement Learning (XIRL) that leverages temporal cycle-consistency constraints to learn deep visual embeddings that capture task progression from offline videos of demonstrations across multiple expert agents, each performing the same task differently due to embodiment differences. Prior to our work, producing rewards from self-supervised embeddings typically required alignment with a reference trajectory, which may be difficult to acquire under stark embodiment differences. We show empirically that if the embeddings are aware of task-progress, simply taking the negative distance between the current state and goal state in the learned embedding space is useful as a reward for training policies with reinforcement learning. We find our learned reward function not only works for embodiments seen during training, but also generalizes to entirely new embodiments. Additionally, when transferring real-world human demonstrations to a simulated robot, we find that XIRL is more sample efficient than current best methods.

Keywords: inverse reinforcement learning, imitation learning, self-supervised learning

1 Introduction

The ability to learn new tasks from third-person demonstrations holds the potential to enable robots to leverage the vast quantities of tutorial videos that can be gleaned from the world-wide web (e.g., YouTube videos). However, distilling diverse and unstructured videos into motor skills with vision-based policies can be daunting – as the videos themselves are often not only captured from different camera viewpoints in different environments, but also with different experts that may use different tools, objects, or strategies to perform the same task. Perhaps most critically, there often exists a clear *embodiment gap* between the human expert demonstrator, and the robot hardware that executes the learned policies. One approach to close this gap is to learn a mapping between the human and robot embodiment [1], which is a non-trivial intermediate problem in itself.

Despite considerable progress in learning policies with paired observations and actions (e.g., collected via teleoperation) [2, 3], much less work has been done in getting robots to learn policies from tasks defined only by third-person observations of demonstrations [4, 5, 6]. This is a surprisingly challenging problem, as different embodiments are likely to use unique strategies that suit them and allow them to make progress on a given task. For example, if asked to "place five pens into a cup", a human hand is likely to scoop up all pens before slipping them into the cup, whereas a two-fingered gripper might instead need to pick and place each pen individually. Both strategies complete the task, but generate different state-action trajectories. In this setting, it may be difficult to acquire labeled frame-to-frame correspondences between expert demonstration videos and learned embodiments [6], particularly across a multitude of embodiments or experts. The approach we investigate instead is *whether we can successfully learn tasks through a learned notion of task progress that is invariant to the embodiment performing the task.*

In this work, we propose to enable agents to imitate video demonstrations of experts – including ones with different embodiments – by using a task-specific, embodiment-invariant reward formulation trained via

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

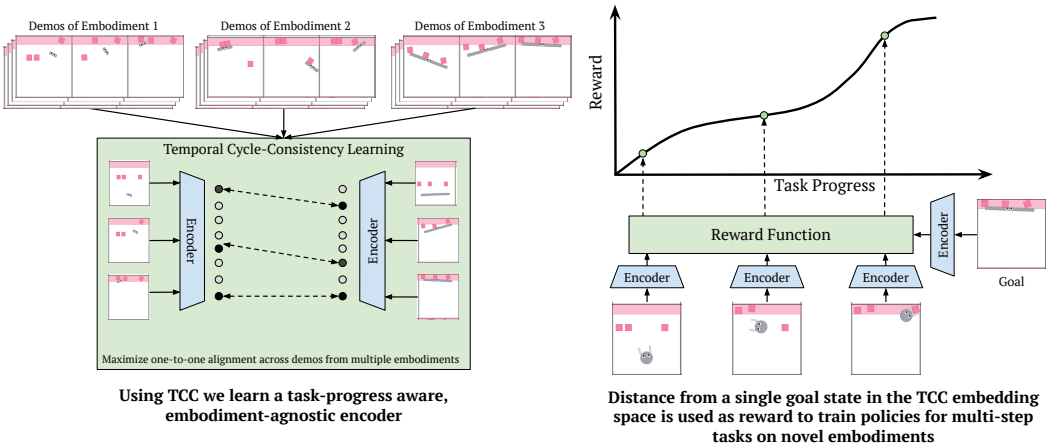


Figure 1. **Cross-embodiment Inverse Reinforcement Learning.** We learn embodiment-invariant visual representations from offline video demonstrations (stick agents on the left) using TCC [7], then use the trained encoder to generate embodiment-invariant visual reward functions that can be used to learn policies on new embodiments (gripper on the right) with reinforcement learning.

temporal cycle-consistency (TCC) [7]. We demonstrate how we can leverage an encoder trained with TCC to define dense rewards for downstream reinforcement learning (RL) policies via simple distances in the learned embedding space. Simulated experiments across four different embodiments show that these learned rewards are capable of generalizing to new embodiments, enabling unseen agents to learn the task via reinforcement, and surprisingly in some cases, exceeding the sample efficiency of the same agent learned with ground truth sparse rewards. We also demonstrate the effectiveness of our approach for learning robot policies using human demonstration on the puck-pushing environment from [6], where our reward is first learned on real-world human demonstrations, then used to teach a Sawyer arm how to perform the task in simulation.

Our contributions are as follows: **(i)** We introduce Cross-embodiment Inverse Reinforcement Learning (XIRL), an effective, label-free framework for tackling cross-embodiment visual imitation learning. Our core contribution is to use self-supervised learning on third-person demonstration videos to define dense reward functions amenable for downstream reinforcement learning policies, **(ii)** Along with XIRL, we release a cross-embodiment imitation learning benchmark, X-MAGICAL, which features multiple simulated agents with different embodiments performing the same manipulation task, including one thousand expert demonstrations for each agent, **(iii)** We show that XIRL significantly outperforms alternative methods on both the X-MAGICAL benchmark and the human-to-robot transfer benchmark from [6], and discuss our observations, which point to interesting areas for future research, **(iv)** Finally, we introduce a real-world dataset, X-REAL (Cross-embodiment Real demonstrations), of a manipulation task performed with nine different embodiments, which can be used to evaluate cross-embodiment reward learning methods[1]. Code and datasets will be made available.

2 Related Work

Traditional formulations of imitation learning [8, 1, 9] assume access to a corpus of expert demonstration data which includes both state and action trajectories of the expert policy. In the context of third-person imitation learning, including when learning from expert agents with different embodiments, obtaining access to ground-truth actions is difficult or impossible[2].

Inferring expert actions. To address this issue, several approaches either try to infer expert actions [10, 11, 12] – for example by training an inverse dynamics model on agent interaction data [10] – or employ forward prediction on the next state to imitate the expert without direct action supervision [13]. While these methods successfully address learning from observation-only demonstrations, they either do not support skill transfer to different policy embodiments at all, or they cannot take advantage of multiple embodiments in order to improve generalization to unseen policy configurations. We explicitly address these problems in this work.

Imitation via learned reward functions. In contrast to imitation via supervised methods, such as BCO [10], a recent body of work [4, 5, 14, 6, 15] has focused on learning reward functions from expert video data and then training RL policies to maximize this reward. In [4], the authors combine ImageNet pre-trained visual features with an L2-norm distance reward to match policy and expert observations in a latent feature space. In their follow-up work [5], the reward is computed in a viewpoint-invariant

[1]For more details regarding X-REAL, please see Appendix C.

[2]For a more comprehensive related work, please see Appendix A.

representation that is self-supervised on video data. While both these methods are compelling in their use of cheap unlabeled data to learn invariant rewards, the use of a time index as a heuristic for defining weak correspondence is a constraining limitation for tasks that need to be executed at different speeds, or are not strictly monotonic (e.g., have ambiguous sub-task ordering). In [14] a dense reward is learned via unsupervised learning on YouTube data and the authors make no assumption about time alignment. However, in their work, the expert and learned policy are executed in the same domain and embodiment, an assumption we relax in our work. Framed in a multi-task learning setting, [16] propose training policies with morphologically different embodiments first on a similar set of proxy tasks, in order to learn a latent space to map between domains, and then sharing skills on a held-out task from one policy to another. A time-index heuristic is used to define a metric reward when performing RL training of the new task. In our work, the learned embedding finds correspondences in a fully-unsupervised fashion, without the need for such strict time alignment. In [17], a small sub-set of states is human labeled for goal success and a convolutional network is then trained to detect successful task completion from image observations, where on-policy samples are used as negatives for the classifier. By contrast, our learned embedding encodes task progress in its latent representation without the use of expensive human labels.

Imitation via domain adaptation. An additional category of approaches to third-person imitation learning are those that perform domain adaptation of expert observations [18, 19, 20, 21]. For instance, in [18] a CycleGAN [22] architecture is used to perform pixel-level image-to-image translation between policy domains, which is then used to construct a reward function for a model-based RL algorithm. A similar model-free approach is proposed in [19]. In [20], a generative model is used to predict robot high-level sub-goals conditioned on a third-person demonstration video, and a lower-level control policy is trained in a task-agnostic manner. Similarly, [21] uses high level task conditioning from zero-shot transfer of expert demonstrations, but they use KL matching to perform both high and low-level imitation. In contrast to these methods, the unsupervised TCC alignment in this work avoids performing explicit domain adaptation or pixel-level image translation by instead learning a robust and invariant feature space in a fully offline fashion.

Reinforcement learning with demonstrations. Recent work in offline-reinforcement learning [6] explicitly tackles the problem of policy embodiment and domain shift. Their method, Reinforcement Learning from Videos (RLV), uses a labelled collection of expert-policy state pairs in conjunction with adversarial training to learn an inverse dynamics model jointly optimized with the policy. In contrast, we avoid the limitation of collecting human-labeled dense state correspondences by using a self-supervised algorithm (i.e., TCC [7]) which uses cycle-consistency to automatically learn the correspondence between states of two domains. We also show that this formulation improves generalization to unseen embodiments. Since the problem setup is similar to ours, we also compare to their method as a baseline.

3 Approach

Our overall XIRL framework (Figure 1) addresses the *cross-embodiment visual imitation* problem (Section 3.1). The framework consists of first using TCC to self-supervise embodiment-invariant visual representations (Section 3.2), then using a novel embodiment-invariant visual reward function (Section 3.3) to perform cross-embodiment visual imitation via reinforcement learning (Section 3.4).

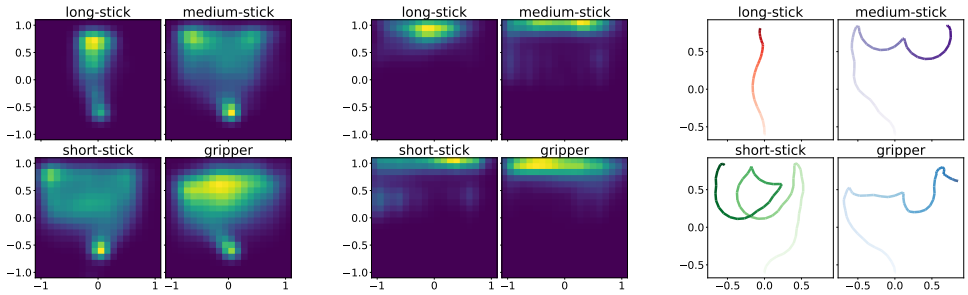
3.1 Problem Formulation

Our objective is to extract an agent-invariant definition of a task, via a learned reward function, from a dataset of videos of separate agents performing the same task. In particular, we are interested in agents that may solve the task in entirely different ways due to differences in their end-effector, shapes, dynamics, etc., which we refer to as *embodiment* differences. For example, consider how differently a vacuum gripper and a parallel-jaw gripper will grasp an object as a result of their respective end-effectors. Such a setup is quite common in robotic imitation learning, where we might have access to observation data of humans demonstrating a task, but want to teach a robot to perform it. It is very likely that the way the human executes the task will diverge from how the robot would execute it.

We define a dataset of multiple agents performing the same task T as $D = \bigcup_{i=1}^{n} D_i$, where D_i is an agent-specific dataset containing observations of *only* agent i performing task T. Each agent's dataset D_i is a collection of videos defined as $D_i = \{v_i^1, v_i^2 ... v_i^K\}$, where v_i^j represents the video of the jth demonstration of agent i successfully performing the task T. We would like to highlight that D only contains observation data, i.e., it does not store the actions taken by the respective agents. We use self-supervised representation learning techniques to learn task-specific representations from this dataset.

3.2 Representation Learning

In this work, we use TCC [7] to learn task-specific representations in a self-supervised way. The method has been shown to learn useful representations on videos of the same action for temporally fine-grained



(a) Agent state distribution. *(b)* Debris state distribution. *(c)* Trajectory taken by each agent in a randomly sampled video demonstration.

Figure 2. **Difference in state distributions** of the agent and debris across different embodiments performing the same task in X-MAGICAL (Section 4.1).

downstream tasks. In their paper, the authors show that TCC representations can predict frame-level task progress, such as predicting how much water is in a cup during pouring, without requiring any human annotations. Task progress can provide dense signals for learning a new task and we would like to bake this property into our learned reward. Another advantage of TCC is that it does not require supervision for frame-level alignment (i.e., which frames in two videos correspond to each other). Such frame-to-frame correspondences are required by a prior method, RLV [6], to achieve successful reinforcement learning on the considered tasks, but we would like to avoid this type of manual supervision.

We train an image encoder ϕ, that ingests an image I and returns an embedding vector $\phi(I)$, using TCC. TCC assumes that there exist semantic temporal correspondences between two video sequences, even though they are not labeled and the actions may be executed at different speeds. Note that in our case, the assumption holds since all sequences in our dataset originate from the same task. Furthermore, even if two different agents accomplish the task very differently, there will be a common set of states or frames that will temporally correspond. When we apply the TCC loss, we compare frames of one agent performing the task with frames from another, and search for temporal similarities in how both execute the task. By performing multiple such comparisons and *cycling-back* (described in the next paragraph), TCC encodes task progress in its latent representation, a property that is useful for learning task-specific, yet embodiment-invariant reward functions. For completeness, we describe the training technique below.

We define a dataset $D_{\text{all}} = \{v_1, v_2 ... v_N\}$ that contains the videos of all agents executing the same task T. We are able to merge the datasets of different agents since TCC does not require embodiment labels (i.e., IDs) during training. We first sample a random batch of videos and embed all their frames using the aforementioned image encoder ϕ. For each video v_i, this results in a sequence of embeddings $V_i = \{\phi(v_i^1), \phi(v_i^2) ..., \phi(v_i^{L_i})\}$, where L_i is the length of the ith video. From this mini-batch of sequences of video frame embeddings, we choose a pair of sequences V_i and V_j and compute their TCC loss. In particular, we randomly sample a frame embedding from sequence V_i – say V_i^t corresponding to the tth frame of video v_i – and compute the soft nearest-neighbor of V_i^t in sequence V_j in the embedding space as follows:

$$\widetilde{V_{ij}^t} = \sum_k^{L_j} \alpha_k V_j^k, \quad \text{where} \quad \alpha_k = \frac{e^{-\|V_i^t - V_j^k\|^2}}{\sum_k^{L_j} e^{-\|V_i^t - V_j^k\|^2}}$$

We then *cycle-back* [7] to the first sequence V_i by computing the soft-nearest neighbor of $\widetilde{V_{ij}^t}$ with all the frames in V_i. The probability of cycling-back to the kth frame in V_i can be computed as:

$$\beta_{ijt}^k = \frac{e^{-\|\widetilde{V_{ij}^t} - V_i^k\|^2}}{\sum_k^{L_i} e^{-\|\widetilde{V_{ij}^t} - V_i^k\|^2}}$$

The expected frame index we cycle-back to is then $\mu_{ij}^t = \sum_k^{L_i} \beta_{ijt}^k k$. Since we know the index of the frame that started the cycle, in this case t, we can minimize the mean-squared (MSE) error loss between t and the closest index retrieved via soft-nearest neighbor, i.e., $\widetilde{V_{ij}^t}$. The loss for a single frame is thus: $L_{ij}^t = (\mu_{ij}^t - t)^2$. Finally, we minimize the average loss L over all frames in video v_i with all other videos in the dataset v_j, defined as $L = \sum_{ijt} L_{ij}^t$.

3.3 Reward Function

Once the encoder ϕ has been trained on demonstrations of different agents performing the same task T, we want to use it to transfer information about the task from one agent to another. We do this by leveraging

Table 1. Statistics of Demos of the X-MAGICAL Embodiments

Embodiment	**Mean ± Std. Dev. Demo Length (no. of frames)**
Long-stick	48.0 ±28.5
Medium-stick	59.9 ±27.9
Short-stick	81.7 ±35.0
Gripper	110.8 ±39.8

ϕ to generate rewards via distances to goal observations in the learned embedding space. Specifically, we define the goal embedding g as the mean embedding of the last frame of all the demonstration videos in our offline dataset D_{all}. Concretely, $g = \sum_{i=1}^{N} \phi(v_i^{L_i})/N$, where L_i is the length of video v_i. Our reward r then is the scaled negative distance of the current state embedding to the goal embedding g i.e., $r(s) = -1/\kappa \cdot \|\phi(s) - g\|_2^2$, where $\phi(s)$ is the state embedding at the current timestep and κ is a scale parameter that ensures the distances are in a range amenable for training reinforcement learning algorithms [23]. We found it effective to set κ to be the average distance of the first frame's embedding to the goal embedding for all the demonstrations in the dataset. Defining r in this manner gives us several advantages: (a) it is dense, encoding both task completion and task progress, (b) it does not require any correspondence with a reference trajectory [5, 14] and (c) it sidesteps the need for a finite library of reference trajectories, unlike prior work [5, 16] that define time-indexed rewards relative to some reference trajectory. Thus, agents with trajectories of varying lengths (due to embodiment-specific constraints) can efficiently leverage this reward because the learned encoder can map different strategies to a common notion of task progress.

3.4 Reinforcement Learning

Using the pre-trained frozen encoder ϕ, we define a Markov Decision Process (MDP) for any agent as the tuple $\langle \mathcal{S}, \mathcal{A}, P, r \rangle$ where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the set of possible actions, P is the state transition probability matrix encoding the dynamics of the environment (including the agent) and r is the *learned* reward function (defined in Section 3.3). Notice how we are able to use the same reward function r for any agent – even ones that the encoder may not have seen during training. Furthermore, this reward function solely depends on the learned encoder. Hence, the task represented by the MDP now depends solely on how well the encoder has learned task-specific representations since we do not use the environment reward to either define the task or learn the policy. This is an important distinction because we are expecting the encoder to generalize to new states that an agent might encounter during training, as the expert demonstrations in our dataset only contain successful trajectories. It is also possible augment sparse rewards (which only define task success or failure) with our learned dense reward while training policies.

4 Experimental Setup

4.1 X-MAGICAL Benchmark

We introduce a cross-embodiment imitation learning benchmark, X-MAGICAL, which is based on the imitation learning benchmark MAGICAL [24], implemented on top of the physics engine PyMunk [25]. In this work, we consider a simplified 2D equivalent of a common household robotic sweeping task, wherein an agent has to push three objects into a predefined zone in the environment (colored in pink). We choose this task specifically because its long-horizon nature highlights how different agent embodiments can generate entirely different trajectories. The reward in this environment is defined as the fraction of debris swept into the zone at the end of the episode.

Multiple Embodiments in X-MAGICAL. We create multiple *embodiments* by designing agents with different shapes and end-effectors that induce variations in how each agent solves a task. In Figure 1, we show three of these embodiments and some sample trajectories that solve the sweeping task. Please see Appendix B for a detailed description of the benchmark. Three agents are shaped like a *stick* and they differ only in length. We call them *short-stick*, *medium-stick* and *long-stick* based on the length of their body. The agent in the last row is called *gripper*: it is circular in shape and has two arms that can actuate. All agents are capable of two actions - a rotation around their axis and a translation in a forward/backward direction along this axis (similar to the agent in the MAGICAL benchmark). All agents have a two-dimensional action space and use force and torque control to change their position and orientation respectively. The gripper agent has an additional degree of freedom for opening or closing its fingers. The default state of the gripper's fingers is open. For all agents, the state representation is a 16-dimensional vector with the following information: (x, y) position of the agent, $(\cos\theta, \sin\theta)$ where θ is the agent's 2D orientation, and for each of the three debris: its (x, y) position, its distance to the agent and its distance to the goal

zone. We frame-stack [26] three consecutive state vectors to encode temporal and velocity information, resulting in a final state dimension of 48.

Demonstrations and Different Embodiment Strategies in X-MAGICAL. To learn task-specific representations for this task, we collect 1000 demonstrations per agent, where each demonstration consists in sweeping all three debris, initialized with random positions, into the target zone. This is the dataset D_{all} (described in Section 3.1) containing observation-only agent-specific demonstrations. In Figure 2, we highlight the differences that exist between the trajectories taken by these agents. Figure 2a shows a heat map of the frequency of visits (*state visitation count*) of each agent at every 2D position in the grid, across all demonstrations in the dataset. We plot the 2D projection of the state visitation count onto the XY plane with a bin width of 0.1. Yellow encodes higher state visitation whereas blue encodes lower state visitation. We observe that agent *long-stick* has less coverage of the environment as opposed to agent *gripper*, which has significantly more coverage. Similarly, we show the distribution of debris locations for all agents across all demonstrations in Figure 2b. In Figure 2c, we plot a randomly sampled trajectory from each agent's demonstration pool. We use transparency to encode the start (lighter) and end (darker) of the trajectory. It is clear from this figure that each agent solves the task in a different manner. Additionally, there is a significant difference in the time taken by each agent to execute this task, as shown in Table 1. Agent *long-stick* is able to finish the task the quickest because of its long shape that can sweep all the debris at once, while agents *short-stick* and *gripper* take longer because they have to frequently push or grasp one debris at a time. These differences are the types of challenges that a representation must overcome to successfully generalize across embodiments. As such, X-MAGICAL serves to create a highly simplified version of a real-world scenario where we might want to learn new tasks from an observation dataset of humans performing these tasks in highly diverse ways.

4.2 Baselines

Here, we describe the alternative reward functions we baseline our method against, color coded to match their appearance in Figs. 3, 4, and 5. *1) ImageNet:* We use an ImageNet pre-trained ResNet-18 encoder with no additional self-supervised training, i.e., we load the pre-trained weights, discard the classification head, and use the 512-dimensional embedding space from the previous layer. *2) Goal classifier:* We follow [27] and train a goal frame classifier on a binary classification task where the last frame of all the demonstrations is considered positive and all the others are considered negatives. We use the output probabilities of the classifier as the reward function. *3) LIFS:* We implement the method from [16] which learns a feature space that is invariant to different embodiments using a contrastive loss function paired with an autoencoding loss. *4) TCN:* single-view Time-Contrastive Network (TCN) [5] with positive and negative frame windows of 1 and 4 respectively. For more details regarding baseline implementations, see Appendix D.1.1.

4.3 Implementation Details

Our encoder for all experiments and methods is a ResNet-18 [28] initialized with ImageNet pre-trained weights. We replace the classification head with an embedding layer outputting a 32-dimensional vector. The encoder is trained on images of resolution 224×224 with ADAM [29] and a learning rate of 10^{-5} which is decreased by a factor of 2 when the validation loss plateaus. Note that our learned reward is agnostic to the RL algorithm used – in this work, we opt for Soft-Actor Critic (SAC) [30], which is a reinforcement learning algorithm that has been successfully used to train policies for continuous control tasks [31]. Once the TCC encoder is trained, we use it to embed the observation frames as the agent interacts with the environment.

5 Experiments

We execute a series of experiments to evaluate whether the learned reward functions are effective at visual imitation. Specifically, our experiments seek to answer the following questions: first (Section 5.1), in the ***same***-embodiment case, where the demonstration dataset D contains the embodiment of the learning agent, does our method enable successful reinforcement learning for that agent? Next (Section 5.2), we investigate our primary interest, the ***cross***-embodiment case, where the demonstration dataset D does **not** contain the embodiment of the learning agent. To additionally test our approach using real-world data (Section 5.3), we use the dataset from [6] to leverage real-world human demonstrations to learn policies in simulation. Note that each embodiment's performance is evaluated over 50 episodes and all figures plot the mean performance over 5 random seeds, with a standard deviation shading of ± 0.5. Videos of our results are in the supplementary video.

5.1 Results on Learning from Same-Embodiment Demonstrations

In this experiment, we want to validate whether our approach of using a reward function trained with TCC is good enough to train agents to perform the task defined in a dataset of expert demonstrations. Note that the learned reward function has to be robust enough such that it can provide a useful signal for

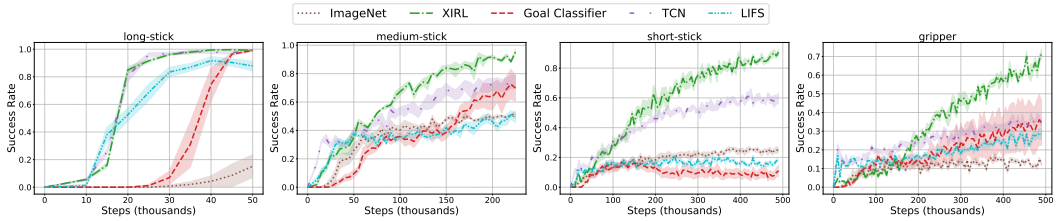


Figure 3. ***Same-embodiment setting:*** Comparison of XIRL with other baseline reward functions, using SAC [30] for RL policy learning on the X-MAGICAL *sweeping* task.

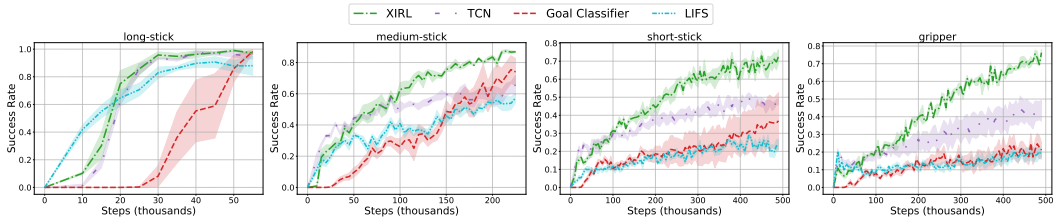


Figure 4. ***Cross-embodiment setting:*** XIRL performs favorably when compared with other baseline reward functions, trained on observation-only demonstrations from different embodiments. Each agent{*long-stick, medium-stick, short-stick, gripper*} is shown using demonstrations from the other 3 embodiments, with SAC [30] for RL policy learning on the X-MAGICAL *sweeping* task.

new states an agent might encounter while learning a policy and interacting with the environment. In Figure 3, we compare our method XIRL with baselines described in Section 4.2. We find XIRL is more sample-efficient than the other learned reward baselines. We attribute this sample efficiency to the fact that the TCC embeddings encode task progress which helps the agent learn to reach for objects and goal zones while interacting with the environment, rather than exploring in a purely random manner. This experiment provides evidence that XIRL's reward function is suitable for downstream reinforcement learning.

5.2 Results on Learning from Cross-Embodiment Demonstrations

After verifying that XIRL works on the embodiments it was trained with, we move to the experiments that answer the **core question** in our work: *can XIRL generalize to unseen embodiments*? In this section we conduct experiments where the reward function is learned using embodiments that are different from the ones on which the policy is trained. As noted in Table 1, the timescales with which the agents execute the task can vary significantly. We conduct four experiments, each one corresponding to holding-out one agent from the expert demonstration set. In each such experiment, we train an encoder on demonstrations from the remaining three agents. We compare with reward functions learned using TCN, LIFS, and goal frame classifiers. While both TCC and TCN are contrastive losses, the former makes explicit comparisons across different embodiments, whereas the latter implicitly relies on an encoder shared across embodiments to learn the cross-embodiment representation. In Figure 4, we show that XIRL generalizes to new agents significantly better than the other learned reward baselines.

5.3 Results on Learning from Real-World Cross-Embodiment Demonstrations

In this experiment, we test how well we can learn rewards from more challenging real-world human demonstration videos. To do so, we use the dataset introduced in [6]. We train only on real-world human demonstrations and use the learned reward to train a Sawyer arm in sim to push a puck to a designated target zone. We compare our results to training the policy on the sparse reward from the environment and RLV method presented in [6]. Unlike RLV, we (i) do not use human labeling of paired frames, and (ii) do not use any data from the robot domain in learning the reward function. Like RLV we also use the sparse reward from the environment. As demonstrated in Figure 5, we find our approach can improve the sample-efficiency of learning in the environment, compared to using RLV or solely the environment reward.

5.4 Qualitative comparison between learned reward functions vs. handcrafted rewards

In Figures 6 and 7, we visualize and compare the XIRL reward function with the environment's reward (ground truth) for the *Sweeping* task from X-MAGICAL (see Sec. 4.1 for details) and the *Puck Pushing* and *Drawer Opening* tasks from [6][3]. We find that the learned reward is highly correlated with the ground truth reward from the environment for both successful demos (first column in Figures 6 and 7) and unsuccessful demos (second column in Figures 6 and 7). It is especially encouraging to see that in a sparser reward environment like the *Drawer Opening* task, XIRL provides a dense signal that should allow the agent to learn the task more efficiently. Additionally, we can see that in the example of the failed

[3]For additional qualitative visualizations on a real-world dataset with multiple embodiments, see Appendix C.

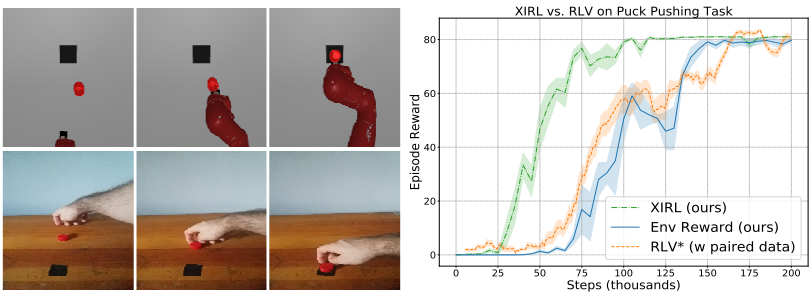


Figure 5. ***Real-world-demo cross-embodiment setting:*** Comparison of XIRL with baselines using the simulated *Puck Pushing* environment from [6], which leveraged real-world demonstration videos of humans (left, row 2) to teach a robot arm in sim (left, row 1). Like [6], we use the sparse environment reward together with using the demonstrations. Unlike [6], we do not use human-labeled data of paired frame correspondences. RLV* denotes results taken verbatim from [6] which uses a different implementation of SAC for RL policy learning.

collision trajectory (i.e., second row third column in Fig. 7), where the arm collides with the drawer rather than opening it, XIRL is able to provide it with a partial reward (i.e., for correctly moving towards the drawer) as opposed to the environment reward which remains zero.

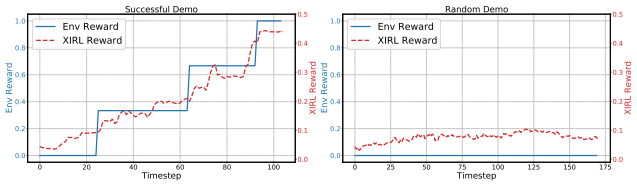


Figure 6. **X-MAGICAL demo cross-embodiment setting:** Visualizing our learned reward function XIRL vs. the environment's sparse reward on a successful and unsuccessful demonstration, for the *short-stick* agent on the X-MAGICAL *sweeping* task. The learned reward was trained on the three other agents.

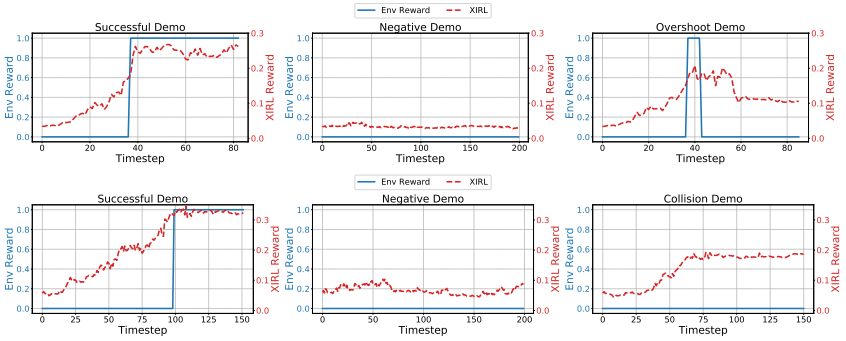


Figure 7. ***Real-world-demo cross-embodiment setting:*** Visualizing our learned XIRL reward function vs. the environment's sparse reward on the *Puck Pushing* (top) and *Drawer Opening* (bottom) tasks from [6].

6 Conclusion

This paper presents XIRL, a framework for learning vision-based reward functions from videos of expert demonstrators exhibiting different embodiments. XIRL uses TCC to self-supervise a deep visual encoder from videos, and uses this encoder to generate rewards via simple distances to goal observations in the embedding space. XIRL enables unseen agents with new embodiments to learn the demonstrated tasks via IRL. Reward functions from XIRL are fully self-supervised from videos, and we can successfully learn tasks without requiring manually paired video frames [6] between the demonstrator and learner. In this sense, our method presents favorable *scalability* to an arbitrary number of embodiments or experts with varying skill levels. Experiments show that policies learned via XIRL are more sample efficient than multiple baseline alternatives, including TCN [5], LIFS [16], and RLV [6]. While our experiments demonstrate promising results for learning policies in simulated environments using rewards learned from both simulated and real-world videos, we have yet to show policy learning on a real robot, which we look forward to trying post-COVID.

Appendix A Expanded Related Work

Traditional formulations of imitation learning [8, 1, 9] assume access to a corpus of expert demonstration data which includes both state and action trajectories of the expert policy. In the context of third-person imitation learning, including when learning from expert agents with different embodiments, obtaining access to ground-truth actions is difficult or impossible.

Inferring expert actions. To address this issue, several approaches either try to infer expert actions [10, 11, 12, 32] – for example by training an inverse dynamics model on agent interaction data [10] – or employ forward prediction on the next state to imitate the expert without direct action supervision [13]. In the case of [33], a video-based action classifier trained on a large-scale human activity dataset is leveraged to provide rewards for single-task RL policies, which are then used to provide expert state-action pairs for multi-task behavior cloning. While these methods successfully address learning from observation-only demonstrations, they either do not support skill transfer to different policy embodiments at all, or they cannot take advantage of multiple embodiments in order to improve generalization to unseen policy configurations. We explicitly address these problems in this work.

Imitation via learned reward functions. In contrast to imitation via supervised methods, such as BCO [10], a recent body of work [4, 5, 14, 6, 15, 34] has focused on learning reward functions from expert video data and then training RL policies to maximize this reward. In [4], the authors combine ImageNet pre-trained visual features with an L2-norm distance reward to match policy and expert observations in a latent feature space. In their follow-up work [5], the reward is computed in a viewpoint-invariant representation that is self-supervised on video data. While both these methods are compelling in their use of cheap unlabeled data to learn invariant rewards, the use of a time index as a heuristic for defining weak correspondence is a constraining limitation for tasks that need to be executed at different speeds, or are not strictly monotonic (e.g., have ambiguous sub-task ordering). In [14] a dense reward is learned via unsupervised learning on YouTube data and the authors make no assumption about time alignment. However, in their work, the expert and learned policy are executed in the same domain and embodiment, an assumption we relax in our work. Framed in a multi-task learning setting, [16] propose training policies with morphologically different embodiments first on a similar set of proxy tasks, in order to learn a latent space to map between domains, and then sharing skills on a held-out task from one policy to another. A time-index heuristic is used to define a metric reward when performing RL training of the new task. In our work, the learned embedding finds correspondences in a fully-unsupervised fashion, without the need for such strict time alignment. In [17], a small sub-set of states is human labeled for goal success and a convolutional network is then trained to detect successful task completion from image observations, where on-policy samples are used as negatives for the classifier. By contrast, our learned embedding encodes task progress in its latent representation without the use of expensive human labels.

Imitation via domain adaptation. An additional category of approaches to third-person imitation learning are those that perform domain adaptation of expert observations [18, 19, 20, 21]. For instance, in [18] a CycleGAN [22] architecture is used to perform pixel-level image-to-image translation between policy domains, which is then used to construct a reward function for a model-based RL algorithm. A similar model-free approach is proposed in [19]. In [20], a generative model is used to predict robot high-level sub-goals conditioned on a third-person demonstration video, and a lower-level control policy is trained in a task-agnostic manner. Similarly, [21] uses high level task conditioning from zero-shot transfer of expert demonstrations, but they use KL matching to perform both high and low-level imitation. In contrast to these methods, the unsupervised TCC alignment in this work avoids performing explicit domain adaptation or pixel-level image translation by instead learning a robust and invariant feature space in a fully offline fashion.

Inspired by maximum entropy inverse RL [35, 36] and generative adversarial networks [37], the seminal GAIL algorithm [38] performs distribution matching between the expert and policy's state-action occupancy via an adversarial formulation; a discriminator is trained with on-policy samples, which is then used as a reward in an RL framework. Many recent works [39, 40, 41, 42, 43] build upon GAIL in order to perform observation-only imitation learning using state-only occupancy matching [39], domain adaptation via domain confusion [40, 41], and state-alignment using a variational autoencoder next-state predictor [42]. Likewise, the algorithm proposed in [15] combines a metric learning loss that uses temporal video coherence to learn a robust skill representation with an entropy-regularized adversarial skill transfer loss. Finally, the authors of [44] propose an adversarial formulation for learning across domains with dynamics, embodiment and viewpoint mismatch. In contrast to these methods, our unsupervised reward is robust to domain shift without requiring online fine-tuning or the additional complexity of dynamic reward learning.

Reinforcement learning with demonstrations. Recent work in offline-reinforcement learning [6] explicitly tackles the problem of policy embodiment and domain shift. Their method, Reinforcement Learning from Videos (RLV), uses a labelled collection of expert-policy state pairs in conjunction with

adversarial training to learn an inverse dynamics model jointly optimized with the policy. In contrast, we avoid the limitation of collecting human-labeled dense state correspondences by using a self-supervised algorithm (i.e., TCC [7]) which uses cycle-consistency to automatically learn the correspondence between states of two domains. We also show that this formulation improves generalization to unseen embodiments. Since the problem setup is similar to ours, we also compare to their method as a baseline.

Appendix B X-MAGICAL Benchmark Details

In this section, we provide more information about our X-MAGICAL benchmark, including a description of the task, an overview of the different embodiments and details regarding horizons, the success metric and the environment reward. We encourage the reader to read [24] for an in-depth description of the base MAGICAL benchmark.

B.1 Action and Observation Space

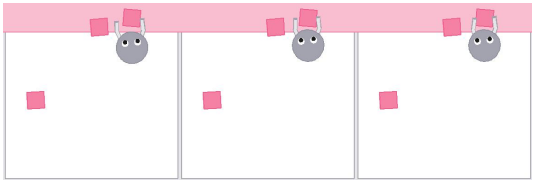


Figure 8. A stack of three image observations for the *gripper* embodiment.

We use a continuous action space for our *Sweeping* task. All embodiments have a 2D action space with the exception of the *gripper* agent which has an additional degree of freedom to open and close its arms. The first degree of freedom is for longitudinal movement (forward/backward), the second degree of freedom is for angular movement (left and right rotation) and the third degree of freedom, if applicable, is a gripping action (push fingers closed/allow fingers to open).

x-MAGICAL provides both state and image-based observations. The state observations are used as input to the RL policies whereas the pixel observations are used by the pretrained encoder to generate rewards. The state vector contains the (x,y) position of the agent, $(\cos\theta,\sin\theta)$ where θ is the agent's 2D orientation, and for each of the three debris: its (x,y) position, its distance to the agent and its distance to the goal zone. For the RGB image, we employ an allocentric, top-down perspective (Figure 8) with full view of the workspace. Similar to MAGICAL, we use an 8Hz control rate, thus with a frame stacking value of 3, this corresponds to roughly 0.3 seconds of interaction.

B.2 Detailed Task and Embodiment Description

In the *Sweeping* task, the robotic agent must push 3 debris blocks to the pink zone at the top of the workspace. The agent's position is constrained to always spawn below the debris. Both the agent's position and the debris positions are randomized at every environment reset. Specifically, we sample the same y-coordinate for all three debris, then randomly space them out from each other (different x-coordinate). The horizon for the *longstick* agent is $H=50$ time steps since it can solve the task much faster than the other embodiments thanks to its morphology. The horizon for all other embodiments is $H=100$. The ground-truth environment reward is defined as $\frac{1}{3}\cdot\sum_{i=1}^{3}\mathbb{1}\{d_i\in G\}$, i.e., the fraction of total debris present inside the goal zone G.

In Figure 9, we provide a film strip demonstration of each embodiment solving the *Sweeping* task with a plot of the environment reward as a function of time. For this visualization specifically, we manually teleoperate each agent and disable the environment horizon limit.

B.3 Demonstration Data

To collect demonstration data for each embodiment, we train an oracle policy with SAC using the ground-truth environment reward. We then rollout the policy in the environment, discarding any potentially unsuccessful demonstrations, until we are left with 1000 demonstrations per embodiment. A comprehensive overview of the hyperparameters used for reinforcement learning are detailed in Table 7 in Appendix G.2.

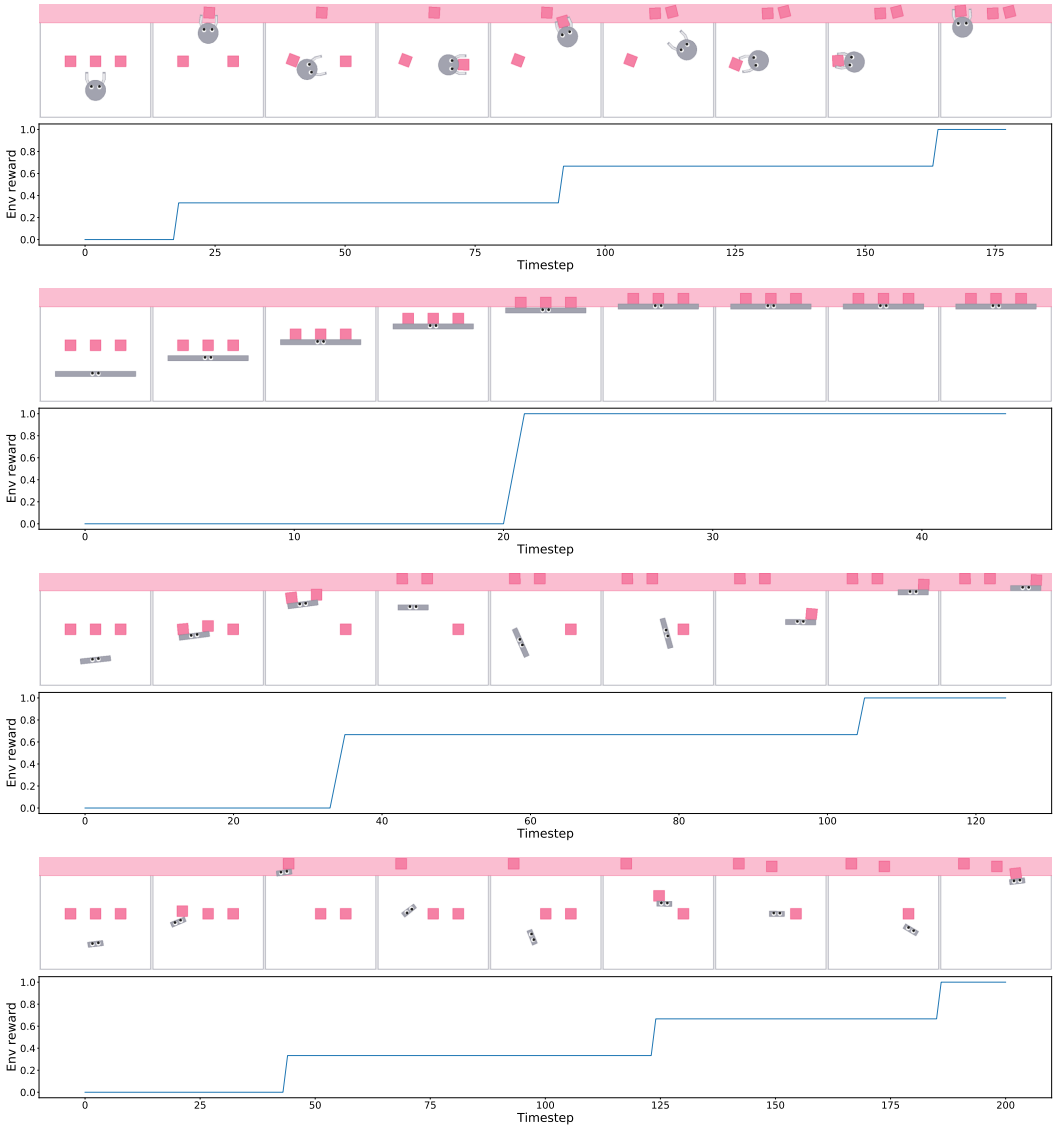


Figure 9. A film strip of the *gripper*, *longstick*, *mediumstick* and *shortstick* embodiments (vertical order) solving the Sweeping task with a corresponding visualization of the ground-truth environment reward.

Appendix C X-REAL: A Real-World Cross-Embodiment Dataset

To test reward learning in the real world on more challenging manipulation tasks, we collect a real-world dataset named X-REAL (Cross-embodiment Real-world demonstrations), which contains 93 demonstration videos of different embodiments (manifested as different manipulator end-effectors) solving the same manipulation task in the real-world: *transferring five pens to two cups consecutively*. This is a multi-step manipulation task where the pens on the table need to be lifted to one cup and then moved again to a separate cup. The different end-effectors consist in a human hand as well as six tools purchased from Amazon and displayed in Figure 10. In contrast to the dataset used in Section 5.3, there is visual diversity among the different end-effectors in X-REAL, and there is also significant variation in how the task is solved and how long it takes to solve the task. Some end-effectors (e.g., tweezers) can only move one pen at a time, while others (e.g., human hand with five fingers) can move all pens at once. Additionally, the demonstrations are not collected in a constrained fashion that tries to mimic the robot. We report the mean and standard deviation of demonstration lengths for each embodiment in Table 2. Note the variation in demo lengths across different embodiments.

Embodiment	**Demo Length Stats (seconds)**
1 Hand 5 Fingers	10.8 ± 1.8
2 Hands 2 Fingers	11.8 ± 1.2
1 Hand 2 Fingers	14.9 ± 1.4
Ski Gloves	20.0 ± 3.9
Kitchen Tongs	21.1 ± 3.1
Lobster Hands (costume)	21.3 ± 3.8
Tweezers	27.8 ± 2.3
RMS Grabber Reacher	29.9 ± 3.2
Irwin Quick-Grip Clamps	38.1 ± 9.2

Table 2. Mean $\pm$ Std. Dev. of Demonstrations Lengths for Different X-REAL Embodiments

Figure 10. Embodiments in the X-REAL dataset, ordered by their appearance in Table 2.

C.1 Data Collection & Hardware Setup

The hardware and data collection setup is shown in Figure 11. We use a GoPro Hero8 mounted on a tripod to record the demonstrations and use voice commands to efficiently start and stop the video recordings. The Hero8 records RGB images with a resolution of 1920×1080 at 30 frames per second.

C.2 Reward Learning from Real-world Multi-Cross-Embodiment Demonstrations

In this section, we learn reward functions from videos in X-REAL, and demonstrate that our method is capable of handling the visual complexities of the real world without requiring annotations of end-effectors, objects, or their states. We train the encoder on all embodiments in the training set and present examples of the learned XIRL rewards on video demonstrations from the validation set in Figure 12. Specifically, we visualize two embodiments: the *RMS Grabber Reacher* (top row) and the human *1 Hand 5 Fingers* (bottom row) and for each embodiment, we show both a successful and unsuccessful trajectory.

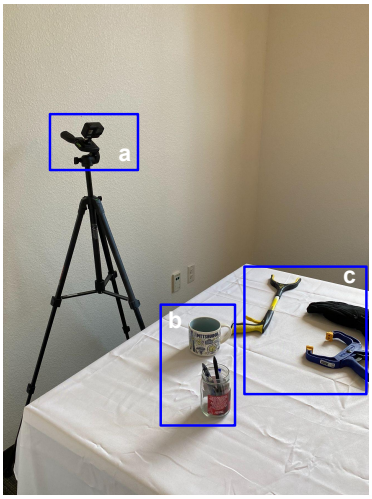


Figure 11. **Hardware setup.** a) Tripod-mounted GoPro Hero8, b) Cup, Mug and Pens from Task c) Example end-effectors used.

In the top row, both the successful and unsuccessful demonstrations follow a similar trajectory at the start of the task execution. The successful one nets a high reward for placing the pens consecutively into the mug then into the glass cup, while the unsuccessful one obtains a low reward because it drops the pens outside the glass cup towards the end of the execution. In the bottom row, for the *1 Hand 5 Fingers* embodiment, we observe that not completing the task and more specifically, leaving the pens in the first cup, generates a reward that is roughly half (image row 2, plot orange curve) the one achieved by a successful execution (image row 1, plot blue curve). These results are encouraging – they show that our learned encoder can represent fine-grained visual differences relevant to the task. Additionally, the training process for this visual reward did not require any additional environment instrumentation (apart from a camera), a desirable property for scaling to more complex, multi-step manipulation tasks.

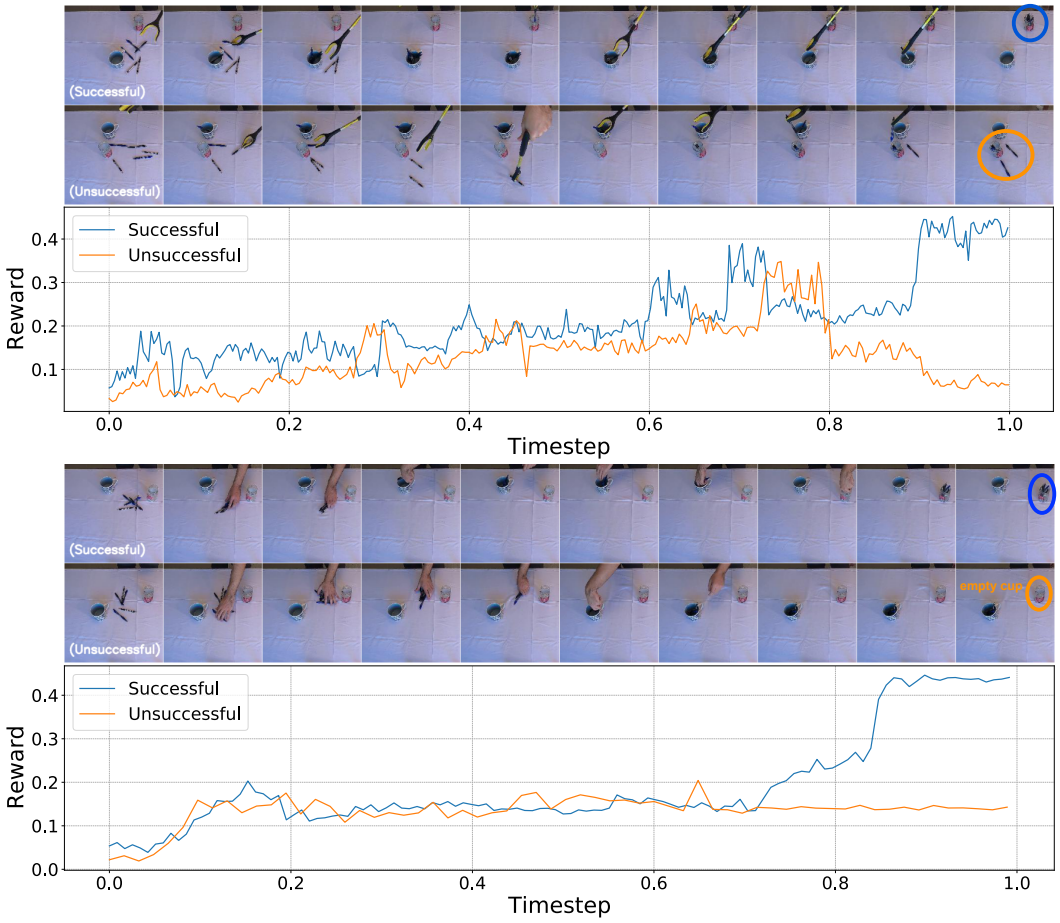


Figure 12. ***X-REAL "move pens to mug then cup" task:*** Visualizing our learned XIRL reward function on successful and unsuccessful demonstrations for the *RMS Grabber Reacher* (*top*) and *1 Hand 5 Fingers* (*bottom*) embodiments.

Appendix D Additional Experimental Details

Our codebase is implemented in PyTorch [45]. Experiments were performed on a desktop machine with an AMD Ryzen 7 2700X CPU (8 Cores/16 Threads, 3.7GHz base clock), 32GB RAM, and a single NVIDIA GeForce RTX 2080 Ti GPU.

D.1 Representation Learning

Each representation learning run – specifically training and evaluating a representation and computing the final goal embedding vector – took an average of 25-30 minutes of wall clock time.

D.1.1 Additional Baseline Details

In this section, we provide a comprehensive overview of the baselines briefly described in Section 4.2.

ImageNet: We use an ImageNet pre-trained ResNet-18 with no additional training, i.e., we load the pre-trained weights, discard the classification head, and use the 512-dimensional embedding space from the penultimate layer.

Goal Classifier: A common strategy for using a learned visual reward function is to train a goal frame classifier [27] on a binary classification task where the last frame of all the demonstrations is considered positive and all the others are considered negatives. Since demonstration sequences do not necessarily end exactly when the task is solved, we randomly sample the negatives from frames that are at least 5 indices away from the final frame. This was chosen heuristically by examining the dataset. Note that unlike [27], we do not have access to any unsuccessful trajectories in the dataset.

Parameter	Setting
Total train iterations	$6k$
Number of frames	15
Batch size	4
Embedding size	32
Unit normalize embeddings	False
Negative window size	5

Table 3. Hyperparameters used for the **Goal Classifier** baseline.

LIFS: We re-implement the approach from [16], which learns a feature space that is invariant to different embodiments using a loss function that encourages *corresponding* pairs of embodiment states across demonstrations to be close in the embedding space. We use the time-based alignment method described in their paper to find these corresponding pairs which assumes each embodiment performs the task at the same rate. To prevent the embeddings from collapsing to a constant value, they use an additional reconstruction loss to encourage the network decoders to preserve as much domain-invariant information as possible. We found early stopping to be crucial in preventing LIFS from collapsing to trivial embeddings. For this baseline, we also randomly sample N evenly-spaced frames from a video to construct each mini-batch.

Parameter	Setting
Total train iterations	$8k$
Number of frames	15
Batch size	4
Embedding size	32
Unit normalize embeddings	True
Embedding temperature	0.1

Table 4. Hyperparameters used for the **LIFS** baseline.

TCN: We re-implement the single-view variant of Time-Contrastive Network (TCN) [5] with positive and negative frame windows of 1 and 4 respectively. We use the n-pair loss [46] to implement TCN as done in [47]. Different from [5], we do not use a time-indexed reward which is not applicable to agents with different embodiments. Like XIRL, we use the negative distance to the average goal embedding as reward. For this baseline, we randomly sample N evenly-spaced frames from a video to construct each mini-batch.

Parameter	Setting
Total train iterations	$4k$
Number of frames	20
Batch size	4
Embedding size	32
Unit normalize embeddings	True
Embedding temperature	learned
Positive window	1
Negative Window	4

Table 5. Hyperparameters used for the **TCN** baseline.

D.1.2 Data Augmentation & Preprocessing

We apply data augmentation during training using the **Albumentations** library [48]. Concretely, this involves the following transformations:

- `RandomResizedCrop`: This transformation takes a random crop from the original image, then resizes it to a final output height and width. The lower and upper bounds for the random crop area are set to $[0.6, 1.0]$, the lower and upper bounds for the random aspect ratio of the crop are set to $[0.75, 1.33]$ and the final output size is set to 224×224. We apply this transformation with a probability of 1.0 and denote it by C.

- `ColorJitter`: This transformation varies the brightness, contrast, saturation and hue of an input image. Our parameters for these respective changes are 0.4, 0.4, 0.1 and 0.1. We apply it with a probability of 0.8 and denote it by J.

- `Normalize`: Lastly, we divide the pixel values by 255 to scale their range to $[0,1]$. We apply it with a probability of 1.0 and denote it by N.

We apply the same randomly sampled transformations to all the sampled frames from the same video. However, independently sampled transformations are applied for each such frame stack in the mini-batch. Note that the order of transformation matters, i.e., we apply the following composed transform: $N \circ J \circ C$.

D.1.3 Training and Evaluation

All our representations are trained using an ADAM optimizer with $\beta_1 = 0.99$, $\beta_2 = 0.999$ and weight decay of 10^{-5}. We use a plateau learning rate scheduler: an initial learning rate of 10^{-5} is decayed by a factor of 2 if the validation loss fails to decrease for 10 consecutive epochs.

While the representations are evaluated on the downstream policy learning performance, we also compute the following quantitative metrics and qualitative results on the train and validation sets to diagnose our representations:

- **Kendall's Tau**: A metric ranging from $[-1,1]$ that measures how well-aligned two sequences are in time. We refer the reader to [7] for a more in-depth explanation.

- **Nearest-Neighbor Alignment Video**: We randomly select one demonstration as a reference video. We use nearest-neighbor in the embedding space to align a test video with the reference video. See Appendix F and the supplemental video for example visualizations for both *same* and *cross*-embodiment settings. These videos highlight how well the embedding space encodes the task progress across different embodiments.

D.1.4 Other Hyperparameters

For a comprehensive list of hyperparameters used for representation learning on the X-MAGICAL environment, the Puck Pushing environment, and X-REAL (Appendix C), see Appendix G.1.

D.2 Policy Learning

The SAC [30] implementation we use is based off of [49]. Each run on the X-MAGICAL benchmark, i.e., the training and evaluation of a specific reward learning method on a specific embodiment with a single seed, took an average of 00h27m, 01h42m, 03h56m and 03h56m wall clock times for *long-stick*, *medium-stick*, *short-stick* and *gripper*, respectively. Note the difference in run times is due to the fact that each embodiment is trained for a different number of total training steps since each converges at different rates: 75k, 225k, 500k and 500k for *long-stick*, *medium-stick*, *short-stick* and *gripper* respectively. For the Puck Pushing experiments in Section 5.3, each run took an average wall clock time of 01h25m for 200k timesteps. Note that run times for both the above environments are recorded while performing up to 5 seed runs in parallel.

D.2.1 Soft-Actor Critic Architecture

We use clipped double Q-learning [50, 51] for the critic, where each Q-function is parameterized by a 3-layer multi-layer perceptron (MLP) with `ReLU` activations. The actor is implemented as a `tanh`-diagonal-Gaussian, and is also parameterized by a 3-layer MLP which outputs mean and covariance. Both actor and critic MLPs have a hidden size of 1024 – the weights are initialized with orthogonal [52] initialization, while the biases are initialized to zero.

D.3 Policy Input

As mentioned in Section 4.1 and Appendix B.1, we construct our observational inputs by stacking 3 consecutive state vectors. The input to the policy is thus a flattened vector in $\mathbb{R}^{48}$. For the *Puck Pushing* environment described in Section 5.3, we construct the observational input by stacking 3 consecutive state vectors containing the 3D Cartesian coordinates of the robot end-effector and the planar 2D coordinates of the puck. The input to the policy is thus a flattened vector in $\mathbb{R}^{15}$.

D.3.1 Training and Evaluation Setup

We first collect 5000 seed observations with a uniform random policy, after which we sample actions using the SAC policy. We then perform one gradient update every time we receive a new environment observation. When evaluating our agent every 5000 steps, we take the mean policy output (i.e., no sampling) and average the final success rate over 50 evaluation episodes.

D.4 Other Hyperparameters

For a comprehensive list of hyperparameters used for policy learning on the X-MAGICAL and Puck Pushing environments, see Appendix G.2.

Appendix E Additional Experiments

E.1 XIRL vs. Ground-truth Environment Reward

In this section, we compare the performance of our method XIRL against the ground-truth environment reward on the same-embodiment experiment from Section 5.1 and the cross-embodiment experiment from Section 5.2. We observe that in both same-embodiment (Figure 13, **top**) and cross-embodiment (Figure 13, **bottom**) settings across multiple embodiments, XIRL is either more or just as sample-efficient as the environment reward. This highlights XIRL's ability to provide denser reward information via encoding task progress, as opposed to the sparser ground-truth environment reward, which was shown in Figure 6(a).

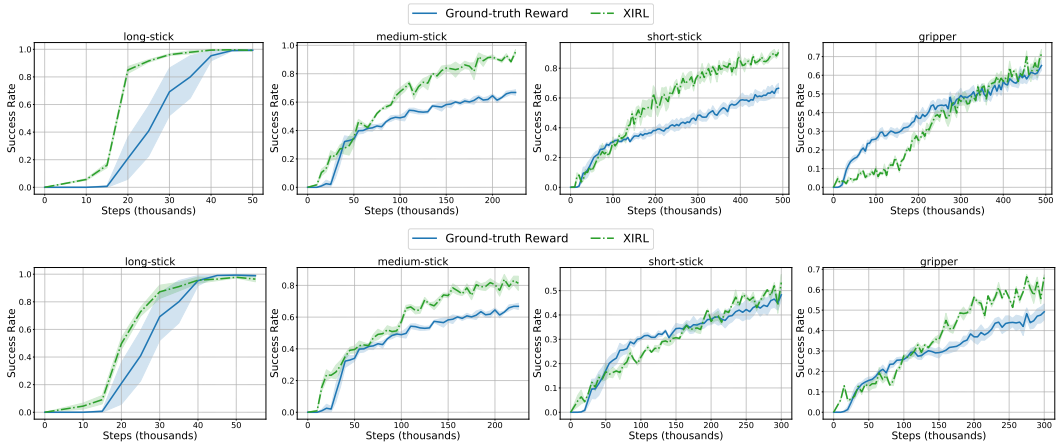


Figure 13. Comparison of XIRL with the ground-truth environment reward in the ***same-embodiment setting*** (top) and the ***cross-embodiment setting*** (bottom

E.2 XIRL vs. SimCLR

In this section, we compare the performance of XIRL against SimCLR [53], a constrative pretraining technique that has exhibited SOTA self-supervised performance on ImageNet. We implement two SimCLR baselines: (a) *SimCLR* is a ResNet18 trained on the x-MAGICAL demonstration dataset with the constrastive pretraining pipeline described in [53], and (b) *SimCLR ImageNet* is a ResNet18 pretrained on ImageNet with SimCLR, with no further pretraining on x-MAGICAL. Below, we present results for the *longstick* and *mediumstick* embodiments of the x-MAGICAL benchmark:

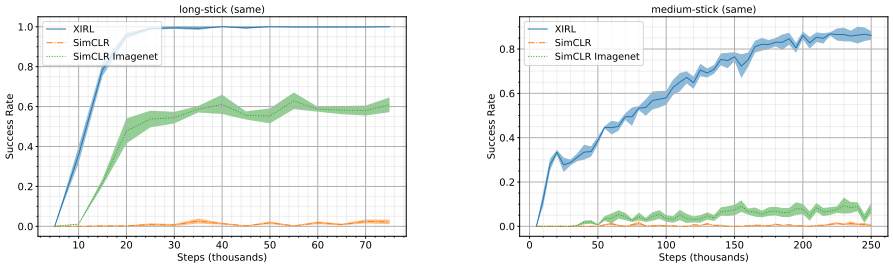


Figure 14. Comparison of XIRL & SimCLR in the ***same-embodiment setting***.

We find that the SimCLR objective performs poorly when trained solely on the x-MAGICAL dataset, whereas the SimCLR baseline pertained on ImageNet (without finetuning on x-MAGICAL) does much better on the longstick embodiment (which is easier to solve). For the mediumstick embodiment, both perform poorly. XIRL, shown in blue, performs significantly better. This highlights that overall, both visual pretraining on cross-embodiment demonstrations and the inductive biases offered by the TCC loss are required to obtain good performance on downstream RL tasks.

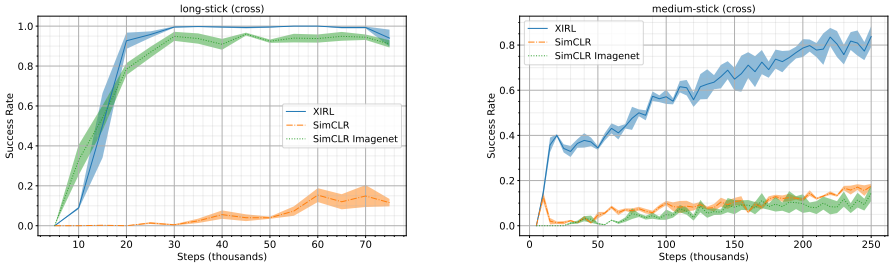


Figure 15. Comparison of XIRL & SimCLR in the ***cross-embodiment setting***.

Appendix F Qualitative Results

F.1 t-SNE Visualizations

To better understand and compare our learned representations, we visualize the t-SNE projection of the learned XIRL and Goal Classifier embedding spaces for 4 video demonstrations of the *shortstick* agent in Figure 16. We observe that:

- Trajectories for different demonstrations overlap and are well-aligned in the XIRL embedding space. In contrast, there is significantly less structure in the Goal Classifier space.
- Distances to the goal (top left corner in the top figure) correlate well to task progress.

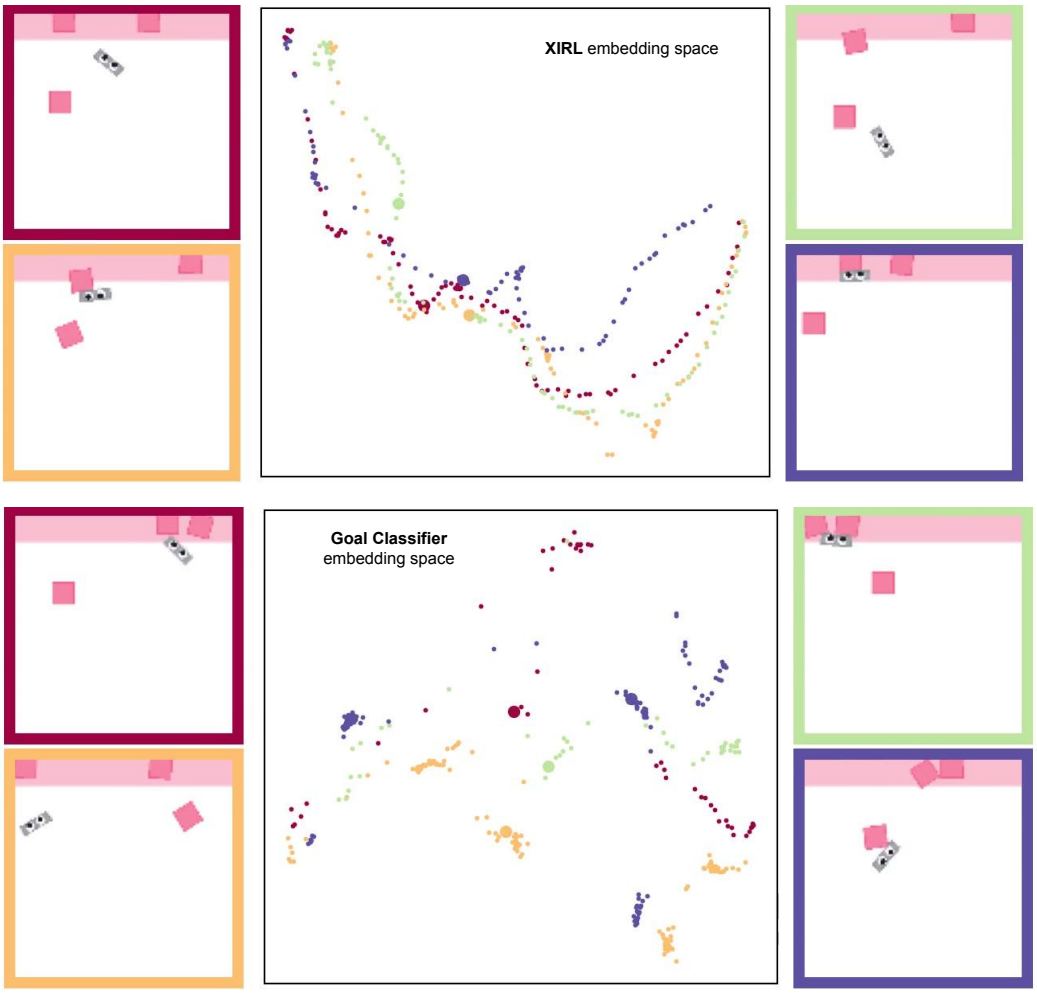


Figure 16. **t-SNE 2D projection** of the learned embedding space for XIRL (**top**) and Goal Classifier (**bottom**), where 4 video demonstrations are embedded and displayed for the *shortstick* agent from the X-MAGICAL benchmark.

We also provide video visualizations of the t-SNE embeddings in the supplementary video, which highlight some more properties.

F.2 Nearest-Neighbor Retrieval

We provide nearest-neighbor alignment videos in the supplementary video.

F.3 Other Visualizations

Please see the supplemental video for videos showing trained policy rollouts on the *Sweeping* and *Puck Pushing* tasks for the cross-embodiment setting, as well as interactive visualizations of the learned reward for the above environments and X-REAL.

Appendix G Hyperparameters

In this section, we give a comprehensive overview of the hyperparameters used for representation learning and policy learning on the *Sweeping* task from X-MAGICAL, the *Puck Pushing* task from [6], and X-REAL.

G.1 Representation Learning

We used mostly the same hyperparameters to train the XIRL encoder across all environments. The main parameters that vary are the embedding dimension and the number of sampled frames.

Hyperparameter	**Value**	**Range Considered**
Loss		
Type	regression (mse)	{mse, huber, x-ent}
Stochastic matching	False	-
Normalize time indices	True	-
Variance-aware	False	-
Distance metric	L2	{cosine, L2}
Optimizer		
Type	ADAM	-
Initial learning rate	10^{-6}	10^{-5} to 10^{-3}
Final learning rate	0	-
β_1	0.99	-
β_2	0.999	-
Weight decay	10^{-5}	10^{-6} to $5{\cdot}10^{-4}$
Learning Rate Scheduler		
Type	`ReduceLROnPlateau`	-
Factor	2	2-10
Patience	10	5-10
Misc.		
Total train iters.	$8{\cdot}10^3$	4-8
Batch size	4	4-8
Normalize embeddings	False	-
Temperature	0.1	-
Learnable temperature	False	-
Augmentations	color jit, rand. crop	-
Frame Sampler	uniform	
Number of frames		
Sweeping	15	-
Puck Pushing	15	-
X-REAL	30	-
Embedding size		
Sweeping	32	32-128
Puck Pushing	64	32-128
X-REAL	64	32-128

Table 6. Hyperparameters for all XIRL representation learning experiments.

G.2 Policy Learning

Most hyperparameters used for downstream reinforcement learning are identical across *X-MAGICAL* and *Puck Pushing*. What changes is the total number of training steps for each embodiment, since some converge much faster than others.

Hyperparameter	**Value**
Total train steps	
Gripper	500K
Shortstick	500K
Mediumstick	200K
Longstick	75K
Sawyer arm (RLV env)	200K
Optimizer	
Type	ADAM
Learning rate	10^{-4}
β_1	0.9
β_2	0.999
Q-network	
Hidden units	1024
Hidden layers	2
Non-linearity	`ReLU`
Actor	
Hidden units	1024
Hidden layers	2
Non-linearity	`ReLU`
Misc.	
Frames stacked	3
Action repetitions	1
Discount factor	0.99
Minibatch size	1024
Replay period every	1 step
Eval period every	5000 step
Number of eval episodes	50
Replay buffer capacity	10^6
Seed steps	5000
Critic target update frequency	2
Actor update frequency	2
Critic target EMA momentum (τ_Q)	0.005
Actor log std dev. bounds	$[-5,2]$
Entropy temperature	0.1
Learnable temperature	True

Table 7. Hyperparameters for all RL experiments.

References

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5), 2009. 1, 2, 9

[2] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *ICRA*. IEEE, 2018. 1

[3] P. Florence, L. Manuelli, and R. Tedrake. Self-supervised correspondence in visuomotor policy learning. *IEEE RAL*, 5(2), 2019. 1

[4] P. Sermanet, K. Xu, and S. Levine. Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*, 2016. 1, 2, 9

[5] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, and S. Levine. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018. 1, 2, 5, 6, 8, 9, 15

[6] K. Schmeckpeper, O. Rybkin, K. Daniilidis, S. Levine, and C. Finn. Reinforcement learning with videos: Combining offline observations with interaction, 2020. 1, 2, 3, 4, 6, 7, 8, 9, 21

[7] D. Dwibedi, Y. Aytar, J. Tompson, P. Sermanet, and A. Zisserman. Temporal cycle-consistency learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2, 3, 4, 10, 16

[8] C. Atkeson and S. Schaal. Robot learning from demonstration. In *ICML*, 1997. 2, 9

[9] T. Osa, J. Pajarinen, G. Neumann, J. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *Found. Trends Robotics*, 7:1–179, 2018. 2, 9

[10] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *IJCAI*, 2018. 2, 9

[11] A. D. Edwards, H. Sahni, Y. Schroecker, and C. L. Isbell. Imitating latent policies from observation. *arXiv preprint arXiv:1805.07914*, 2018. 2, 9

[12] I. Radosavovic, X. Wang, L. Pinto, and J. Malik. State-only imitation learning for dexterous manipulation. *arXiv preprint arXiv:2004.04650*, 2020. 2, 9

[13] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 2050–2053, 2018. 2, 9

[14] Y. Aytar, T. Pfaff, D. Budden, T. Paine, Z. Wang, and N. de Freitas. Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems*, pages 2930–2941, 2018. 2, 3, 5, 9

[15] O. Mees, M. Merklinger, G. Kalweit, and W. Burgard. Adversarial skill networks: Unsupervised robot skill learning from video. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4188–4194. IEEE, 2020. 2, 9

[16] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine. Learning invariant feature spaces to transfer skills with reinforcement learning, 2017. 3, 5, 6, 8, 9, 15

[17] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019. 3, 9

[18] L. Smith, N. Dhawan, M. Zhang, P. Abbeel, and S. Levine. Avid: Learning multi-stage tasks via pixel-level translation of human videos. *arXiv preprint arXiv:1912.04443*, 2019. 3, 9

[19] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *ICRA*. IEEE, 2018. 3, 9

[20] P. Sharma, D. Pathak, and A. Gupta. Third-person visual imitation learning via decoupled hierarchical controller. In *Advances in Neural Information Processing Systems*, pages 2597–2607, 2019. 3, 9

[21] D. Hejna, L. Pinto, and P. Abbeel. Hierarchically decoupled imitation for morphological transfer. In *International Conference on Machine Learning*, pages 4159–4171. PMLR, 2020. 3, 9

[22] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017. 3, 9

[23] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 5

[24] S. Toyer, R. Shah, A. Critch, and S. Russell. The magical benchmark for robust imitation. *arXiv preprint arXiv:2011.00401*, 2020. 5, 11

[25] V. Blomqvist. *Pymunk: A easy-to-use pythonic rigid body 2d physics library (version 6.0.0)*, 2007. URL https://www.pymunk.org. 5

[26] 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*, 518(7540): 529–533, 2015. 6

[27] M. Vecerik, O. Sushkov, D. Barker, T. Rothörl, T. Hester, and J. Scholz. A practical approach to insertion with variable socket position using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 754–760. IEEE, 2019. 6, 15

[28] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 6

[29] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint:1412.6980*, 2014. 6

[30] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018. 6, 7, 16

[31] 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*, 2018. 6

[32] C. Yang, X. Ma, W. Huang, F. Sun, H. Liu, J. Huang, and C. Gan. Imitation learning from observations by minimizing inverse dynamics disagreement. *ArXiv*, abs/1910.04417, 2019. 9

[33] L. Shao, T. Migimatsu, Q. Zhang, K. Yang, and J. Bohg. Concept2robot: Learning manipulation concepts from instructions and human demonstrations. *Robotics: Science and Systems*, 2020. 9

[34] G. Berseth and C. Pal. Visual imitation learning with recurrent siamese networks. *ArXiv*, abs/1901.07186, 2019. 9

[35] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. 9

[36] B. D. Ziebart, J. A. Bagnell, and A. K. Dey. Modeling interaction via the principle of maximum causal entropy. In *ICML*, 2010. 9

[37] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014. 9

[38] J. Ho and S. Ermon. Generative adversarial imitation learning. In *NeurIPS*, 2016. 9

[39] F. Torabi, G. Warnell, and P. Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018. 9

[40] B. C. Stadie, P. Abbeel, and I. Sutskever. Third-person imitation learning. *arXiv:1703.01703*, 2017. 9

[41] Y. Lu and J. Tompson. Adail: Adaptive adversarial imitation learning. *NeurIPS Workshop on Learning Transferable Skills,*, 2019. 9

[42] F. Liu, Z. Ling, T. Mu, and H. Su. State alignment-based imitation learning. *arXiv preprint arXiv:1911.10947*, 2019. 9

[43] W. Sun, A. Vemula, B. Boots, and J. Bagnell. Provably efficient imitation learning from observation alone. *ArXiv*, abs/1905.10948, 2019. 9

[44] K. Kim, Y. Gu, J. Song, S. Zhao, and S. Ermon. Domain adaptive imitation learning. *arXiv*, 2020. 9

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019. 15

[46] K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 1857–1865, 2016. 15

[47] D. Dwibedi, J. Tompson, C. Lynch, and P. Sermanet. Learning actionable representations from visual observations. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1577–1584. IEEE, 2018. 15

[48] A. Buslaev, A. Parinov, E. Khvedchenya, V. Iglovikov, and A. A. Kalinin. Albumentations: fast and flexible image augmentations. *arXiv e-prints*, 2018. 15

[49] D. Yarats and I. Kostrikov. Soft actor-critic (sac) implementation in pytorch. https://github.com/denisyarats/pytorch_sac, 2020. 16

[50] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *arXiv e-prints*, 2015. 16

[51] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmassan, Stockholm, Sweden, July 10-15, 2018*, 2018. 16

[52] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv e-prints*, 2013. 16

[53] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020. 18