Towards Broader and More Efficient Object Manipulation via Deep Reinforcement Learning

Yufei Wang

CMU-CS-20-138

Dec 2020

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
David Held, Chair
Katerina Fragkiadaki
Deepak Pathak

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

Keywords: Reinforcement Learning, Robotic Manipulation, Image-based Reinforcement Learning, Inverse Reinforcement Learning, Self-supervised Learning, Reward Learning, State Marginal Matching, Deformable Object Manipulation, Benchmarking

For my family, friends, and teachers.

Abstract

Reinforcement learning ("RL") has achieved great success in many robotic object manipulation tasks, such as pushing, grasping, tossing, and more. However, there remain some challenges in applying RL to a broader range of object manipulation tasks in the real world. First, it is challenging to design the correct reward function, as well as to obtain it directly from high-dimensional images in the real world. Second, although great progress has been made in the regime of rigid object manipulation, manipulating deformable objects remains challenging due to its high dimensional state representation, and complex dynamics. In this thesis, we aim to push forward the application of deep RL to object manipulation, by proposing the following solutions to address these two challenges.

Specifically, for obtaining a reward function directly from images, current image-based RL algorithms typically operate on the whole image without performing object-level reasoning. This leads to ineffective reward functions. In this thesis, we improve upon previous visual self-supervised RL by incorporating object-level reasoning and occlusion reasoning. We use unknown object segmentation to ignore distractors in the scene for better reward computation and goal generation; we further enable occlusion reasoning by employing a novel auxiliary loss and training scheme. We demonstrate that our proposed algorithm, ROLL (Reinforcement learning with Object Level Learning), learns dramatically faster and achieves better final performance compared with previous methods in several simulated visual control tasks.

We further propose a new inverse reinforcement learning method for learning the reward function to match the given expert state density. Our main result is the analytic gradient of any f-divergence between the agent and expert state distribution w.r.t. reward parameters. Based on the derived gradient, we present an algorithm, f-IRL, that recovers a stationary reward function from the expert density by gradient descent. We show that f-IRL can learn behaviors from a hand-designed target state density or implicitly through expert observations. Our method outperforms adversarial imitation learning methods in terms of sample efficiency and the required number of expert trajectories on IRL benchmarks. Moreover, we show that the recovered reward can be used to quickly solve downstream tasks, and empirically demonstrate its utility on hard-to-explore tasks and for behavior transfer across changes in dynamics.

To facilitate the research of using deep RL to explore the challenges of deformable object manipulation, in this thesis, we present SoftGym, a set of open-source simulated benchmarks for manipulating deformable objects, with a standard OpenAI Gym API and a Python interface for creating new environments. Our benchmark will enable reproducible research in this important area. Further, we evaluate a variety of algorithms on these tasks and highlight challenges for reinforcement learning algorithms, including dealing with a state representation that has a high intrinsic dimensionality and is partially observable. The experiments and analysis indicate the strengths and limitations of existing methods in the context of deformable object manipulation that can help point the way forward for future methods development.

Acknowledgments

Among the list of people that deserve my sincere gratitude, I would love to put my advisor, Professor David Held, at the top of it, for his exceptional guidance, patience and support during my master's study. His insightful research advice made this thesis possible. More importantly, his passion and professional vision in the field of robot learning has greatly influenced me.

I would like to thank my mentor, Xingyu Lin. He sets for me an excellent example of being a professional researcher and a good person. I've learned a lot from his professional skills, valuable insights, and persistence in addressing research challenges. I thank Professor Katerina Fragkiadaki and Professor Deepak Pathak for being my committee members and providing useful feedbacks.

It is also my great fortune to meet and work with a group of nice people at CMU and RPAD - Gautham Narsimhan, Tianwei Ni, Harshit Sikchi, Tejus Gupta, Brian Okorn, Wenxuan Zhou, Thomas Weng, Sujay Bajracharya, Qiao Gu, Lisa Lee, and Benjamin Eysenbach. I give my sincere appreciation for all your kind help during my study here.

At last, I would like to thank my parents for their continuous love and support along every step in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Recently, reinforcement learning ("RL") has achieved great success in many robotic object manipulation tasks, such as pushing [2, 28], grasping [88, 89], tossing [131], dexterous in-hand manipulation [3], and more. However, there remain some challenges in applying RL to a broader range of object manipulation tasks in the real world. One challenge is designing the correct reward function for RL learning, which is known to be tedious and requires lots of manual engineering [42]. Even with a correctly designed reward function, obtaining such a reward function in the real world directly from images is still challenging, as it can be very difficult to perform state estimation in cluttered real-world environments with severe occlusions to give the reward. Meanwhile, although great progress has been achieved in the regime of rigid object manipulation, manipulating deformable objects remains challenging due to its high dimensional state representation, and complex dynamics and visual observation. In this thesis, we aim to push forward the application of deep RL for object manipulation by proposing solutions to the aforementioned two challenges: 1. We propose a novel algorithm that learns a self-supervised reward function directly from images that achieves better sample-efficiency; 2. We propose a new inverse reinforcement learning algorithm that can extract both the reward and the optimal policy from given expert state densities; 3. We create a suite of benchmarking tasks for evaluating deep RL for deformable object manipulation.

1.2 Learning a Self-Supervised Reward Function from Images

To address the problem of obtaining a reward in the real world from high-dimensional images, recent methods proposed to use a learned reward function [81, 91] using distance in a learned latent space. These methods also sample goals from this learned latent space for self-training on a diverse set of goals. However, when learning the latent representation, most of these previous image-based self-supervised RL algorithms operate on the whole image without performing object-level reasoning. This leads to inefficient goal sampling and an ineffective reward function.

For example, in previous methods, a robot arm can be a distractor for policy learning, because it is appears in the robot observations and is thereby encoded into the latent space; thus, the reward derived from that latent space will account for the position of the robot arm information. Ideally, for object manipulation tasks, the sampled goals and reward should relate to only the target objects, rather than the robot arm or static parts of the scene.

In this thesis, we improve upon previous image-based self-supervised RL by incorporating object-level reasoning. Specifically, to help the robot reason more about the target objects, we propose to use unknown object segmentation to segment out the target object and remove all other distractors in the scene; we show that this can allow for better reward computation and goal sampling.

However, with a naive segmentation, the latent representation might suffer from object occlusions; if the object is occluded, then the segmented image might be empty. To handle this case, we augment our method with occlusion reasoning by using a novel occlusion-aware loss function and training scheme. Our proposed algorithm can generally be used in any setting that has a static background, which is common for many robotics tasks. Furthermore, it works with any type of objects and does not require any prior information about the objects; thus our method can be applied for learning various object manipulation tasks.

To summarize, the main contributions of this part of work are:

- We propose a new algorithm, ROLL (Reinforcement learning with Object Level Learning), that incorporates object-level reasoning to improve reward computation and goal sampling for visual self-supervised RL.
- We make ROLL robust to object occlusions by using a novel auxiliary loss and training scheme.
- We demonstrate that ROLL learns dramatically faster and achieves better final performance compared with previous methods in several simulated visual control tasks.

1.3 Inverse Reinforcement Learning via State Marginal Matching

Imitation learning (IL) is a powerful tool to design autonomous behaviors in robotic systems. Although reinforcement learning methods promise to learn such behaviors automatically, they have been most successful in tasks with a clear definition of the reward function. Reward design remains difficult in many robotic tasks such as driving a car [90], tying a knot [85], and human-robot cooperation [39]. Imitation learning is a popular approach to such tasks, since it is easier for an expert teacher to demonstrate the desired behavior rather than specify the reward [8, 43, 47].

Methods in IL frameworks are generally split into behavior cloning (BC) [9] and inverse reinforcement learning (IRL) [1, 83, 98]. BC is typically based on supervised learning to regress expert actions from expert observations without the need for further interaction with the environment, but suffers from the covariate shift problem [96]. On the other hand, IRL methods aim to learn the reward function from expert demonstrations, and use it to train the agent policy. Within IRL, adversarial imitation learning (AIL) methods (GAIL [45], AIRL [32], f-MAX [34], SMM [61]) train a discriminator to guide the policy to match the expert's state-action distribution.

AIL methods learn a *non-stationary* reward by iteratively training a discriminator and taking a single policy update step using the reward derived from the discriminator. After convergence, the learned AIL reward cannot be used for training a new policy from scratch, and is thus discarded. In contrast, IRL methods such as ours learn a **stationary reward** such that, if the policy is trained from scratch using the reward function until convergence, then the policy will match the expert behavior. We argue that learning a stationary reward function can be useful for solving downstream tasks and transferring behavior across different dynamics.

Traditionally, IL methods assume access to expert demonstrations and minimize some divergence between policy and expert's trajectory distribution. However, in many cases, it may be easier to directly specify the state distribution of the desired behavior rather than to provide fully-specified demonstrations of the desired behavior [61]. For example, in a safety-critical application, it may be easier to specify that the expert never visits some unsafe states, instead of tweaking reward to penalize safety violations [20].

Similarly, we can specify a uniform density over the whole state space for exploration tasks, or a Gaussian centered at the goal for goal-reaching tasks. Reverse KL instantiation for f-divergence in f-IRL allows for unnormalized density specification, which further allows for easier preference encoding.

In this paper, we propose a new method, f-IRL, that learns a stationary reward function from the expert density via gradient descent. To do so, we derive an analytic gradient of any arbitrary f-divergence between the agent and the expert state distribution w.r.t. reward parameters. We demonstrate that f-IRL is especially useful in the limited data regime, exhibiting better sample efficiency than prior work in terms of the number of environment interactions and expert trajectories required to learn the MuJoCo benchmark tasks. We also demonstrate that the reward functions recovered by f-IRL can accelerate the learning of hard-to-explore tasks with sparse rewards, and these same reward functions can be used to transfer behaviors across changes in dynamics.

1.4 Benchmarking Deep Reinforcement Learning for Deformable Object Manipulation

Robotic manipulation of deformable objects has wide application both in our daily lives, such as folding laundry and making food, and in industrial applications, such as packaging or handling cables. However, programming a robot to perform these tasks has long been a challenge in robotics due to the high dimensional state representation and complex dynamics [73, 75, 76].

One potential approach to enable a robot to perform these manipulation tasks is with deep reinforcement learning (RL), which has achieved many successes in recent years [6, 47, 77, 102, 116]. Some recent works have used learning-based methods to explore the challenges of deformable object manipulation [2, 65, 74, 106]; however, these works often each evaluate on a different task variant with different simulators or robot setups, making it challenging to directly compare these approaches. There is currently no benchmark for evaluating and comparing different approaches for deformable object manipulation.

In contrast, there are a number of popular benchmarks for reinforcement learning with rigid or articulated objects [15, 27, 110]. Many of these benchmarks assume that the agent directly observes a low dimension state representation that fully describes the underlying dynamics of the environment, such as the joint angles and velocities of the robot [15, 22] or object state [6, 49]. However, such low-dimensional sufficient state representations are difficult to perceive (or sometimes even define) for many deformable object tasks, such as laundry folding or dough manipulation. For deformable object manipulation, the robot must operate directly on its observations, which can include camera images and other sensors.

In this thesis, we present SoftGym, a set of open-source simulated benchmarks for manipulating deformable objects, with a standard OpenAI Gym API and Python interface for creating new environments. Currently, SoftGym includes 10 challenging environments involving manipulation of rope, cloth and fluid of variable properties, with different options for the state and action spaces. These environments highlight the difficulty in performing robot manipulation tasks in environments that have complex visual observations with partial observability and an inherently high dimensional underlying state representation for the dynamics. SoftGym provides a standardized set of environments that can be used to develop and compare new algorithms for deformable object manipulation, thus enabling fair comparisons and thereby faster progress in this domain.

We benchmark a range of algorithms on these environments assuming different observation spaces for the policy, including full knowledge of the ground-truth state of the deformable object, a low-dimension state representation, and only visual observations. Our results show that learning with visual observations leads to much worse performance compared to learning with ground-truth state observations in many deformable object manipulation tasks. The poor performance of image-based methods on these environments motivates the need for future algorithmic development in this area. We also provide an analysis to give some insight into why current methods that use visual observations might have suboptimal performance; this analysis can hopefully point the way towards better methods for deformable object manipulation.

Chapter 2

Related Work

Image-based RL: Image-based deep reinforcement learning has been applied to robotics tasks to learn a variety of behaviors including navigation [58], grasping [11, 89], pushing [2], and more. Goal-conditioned RL has the potential of learning general-purpose policies for performing different tasks specified as different goals [5, 101, 109]. Recent work in goal-conditioned image-based RL uses goals defined as images [68, 80, 81, 91]. These approaches learn a latent encoding of the image goals and then condition the policy on the low-dimensional encoded goal vector. However, such methods typically also encode information that are not part of the intended goal, such as the position of the robot arm. Our work also uses image goals, but we perform object-level reasoning, leading to more robust goal representations and faster learning.

Object Reasoning in Reinforcement Learning: Object-oriented RL and OO-MDP [21] have been proposed as alternatives to classic MDPs to improve data efficiency and generalization by leveraging representations of objects and their relations. Most of the existing approaches require hand-crafted object representations and their relations [19, 21, 33], with some recent work aiming to automatically discover these from data [52, 113, 118, 130]. In a recent work [52], object-oriented RL is used to help exploration. Relational RL [130] uses an attention mechanism to extract the relations between objects in the scene to help policy learning. In our method, we use unknown object segmentation to extract the object representation directly from images, which does not need any prior knowledge of the object. Similar to previous work, we also use object reasoning to improve the data efficiency for RL learning, but we do so to obtain a better reward function in the self-supervised RL setting.

Occlusion Reasoning in Robotics: Object occlusions pose a huge challenge for robot learning: even simple distractors that occasionally occlude the object can cause state-of-the-art RL algorithms to fail [17]. Theoretically, occlusions can be modeled using the framework of POMDP [50], but usually such a formulation is intractable to solve, especially when the states are image observations. Other than POMDP, there has been lots of work aiming to solve the occlusion problem. Cheng, [17] use active vision which learns a policy to move the camera to avoid occlusions. To track the possibly occluded pixels, Ebert, [23] use a Conv-LSTM with temporal skip connections to copy pixels from prior images in the history. We also use an LSTM to handle occlusions, but we further use an occlusion-aware loss and training procedure to make it more robust.

Inverse Reinforcement Learning: IRL methods [1, 83, 98] obtain a policy by learning a

IL Method	Space	f-Divergence	Recover Reward?
MaxEntIRL, GCL	τ	Forward Kullback-Leibler	✓
GAN-GCL	τ	Forward Kullback-Leibler*	✓
AIRL, EAIRL	τ	Forward Kullback-Leibler*	✓
EBIL	τ	Reverse Kullback-Leibler	✓
GAIL	s, a	Jensen-Shannon	×
f-MAX	s, a	f-divergence	×
SMM	s	Reverse Kullback-Leibler	×
f-IRL (Our method)	s	f-divergence	✓

Table 2.1: IL methods vary in the domain of the expert distribution that they model ("space"), the choice of f-divergence, and whether they recover a stationary reward function. *GAN-GCL and AIRL use biased IS weights to approximate FKL (see Appendix B.2).

reward function from sampled trajectories of an expert policy. MaxEntIRL [133] learns a stationary reward by maximizing the likelihood of expert trajectories, i.e., it minimizes forward KL divergence in trajectory space under the maximum entropy RL framework. Similar to MaxEntIRL, Deep MaxEntIRL [122] and GCL [30] optimize the forward KL divergence in trajectory space. A recent work, EBIL [70], optimizes the reverse KL divergence in the trajectory space by treating the expert state-action marginal as an energy-based model. Another recent method, RED [117], uses support estimation on the expert data to extract a fixed reward, instead of trying to minimize a f-divergence between the agent and expert distribution.

Adversarial Inverse Reinforcement Learning: One branch of IRL methods train a GAN [35] with a special structure in the discriminator to learn the reward. This is first justified by Finn et al. [29] to connect GCL [30] with GAN, and several methods [29, 32, 92] follow this direction. Our analysis in Appendix B.2 suggests that the importance-sampling weights used in these prior methods may be biased. We show that AIRL does not minimize the reverse RL in state-marginal space (as argued by [34]). Moreover, AIRL [32] uses expert state-action-next state transitions, while our method can work in a setting where only expert states are provided.

Adversarial Imitation Learning: A set of IL methods [34, 45] use a discriminator to address the issue of running RL in the *inner* loop as classical IRL methods. Instead, these methods directly optimize the policy in the *outer* loop using adversarial training. These methods can be shown to optimize the Jensen-Shannon, and a general f-divergence respectively, but do not learn a reward function. SMM [61] optimizes the reverse KL divergence between the expert and policy state marginals but also does not recover a reward function due to its fictitious play approach. SQIL [93] and DRIL [14] utilize regularized behavior cloning for imitation without recovering a reward function. Unlike these prior methods, f-IRL can optimize any f-divergence between the state-marginal of the expert and the agent, while also recovering a stationary reward function. Table 2.1 summarizes the comparison among imitation learning methods.

Robotic Manipulation of Deformable Objects: Robotic manipulation of deformable objects has a rich history across various fields, such as folding laundry [73], preparing food [12], or assistive dressing and feeding [16, 25]. Early works used traditional vision algorithms to detect key features such as edges and corners [73, 112, 120]. Motion planning is then adopted along with analytical models of the deformable objects [94, 99]. However, these planning approaches often suffer from the large configuration space induced by the high degree of freedom of the deformable objects [26]. We refer to [53, 100] for a more detailed survey on prior methods for robot manipulation of deformable objects.

Manipulating Deformable Objects via Deep Reinforcement Learning: Recently, the success of deep learning has garnered increased interest in learning to solve perception and manipulation tasks in an end-to-end framework [2, 24, 87, 106, 128]. In cloth manipulation, recent work uses demonstrations to learn an image-based policy for cloth folding and draping [74]. Other works learn a pick-and-place policy to spread a towel [105, 121]. Due to the large number of samples required by reinforcement learning, as well as the difficulty in specifying a reward function, all these works start by training the policy in simulation and then transfer the policy to a real robot through domain randomization and dynamics randomization. However, these papers do not systematically compare different methods on a range of tasks.

Benchmarking Environments in Reinforcement Learning: Standard environments with benchmarks have played an important role in the RL community, such as the Arcade Learning environments [10] and the MuJoCo environments [22]. A variety of new environments have been created recently to benchmark reinforcement learning algorithms [27, 48, 62, 86, 129]. However, none of these benchmark environments incorporate deformable objects, and usually the full state of the system can be represented by a low-dimensional vector. Other recent environments built on top of the Nvidia PhysX simulator also have the ability to simulate of deformable objects [104, 123] but do not include any tasks or assets for manipulating deformable objects. As such, we believe that SoftGym would be a unique and valuable contribution to the reinforcement learning and robotics communities, by enabling new methods to be quickly evaluated and compared to previous approaches in a standardized and reproducible set of simulated environments.

Chapter 3

ROLL: Visual Self-Supervised Reinforcement Learning with Object Reasoning

3.1 Problem Formulation

We consider the setting in which a robot needs to learn to achieve a parameterized set of goals, e.g., pushing a puck to various target locations, with the goals specified using images. The robot only has RGB sensor inputs and has no access to the ground-truth states, e.g., puck positions. Formally, this can be formulated as image-based goal-conditioned RL, with a state space S of images, a goal space G which we assume to be the same as S, an action space A, a transition dynamics $T : S \times A \times S \rightarrow [0, 1]$, and a reward function $r : S \times A \times G \rightarrow \mathcal{R}$. Given the current observation I_t and the goal image I_g, the robot needs to learn a goal-conditioned policy $\pi(\cdot|I_t, I_g) : S \times G \rightarrow [0, 1]$ that maps the image $I_t \in S$ and goal $I_g \in G$ to a distribution over actions $a \in A$ to reach the goal.

As it is hard to specify a reward function and learn a policy directly based on high-dimensional images [68, 81], previous work [81, 91] has employed a β-VAE [44]. The β-VAE consists of an encoder $f_\phi(\cdot)$ to encode the image I_t to a latent vector $z_t = f_\phi(I_t)$. This previous work then uses z_t as the input to the policy and for computing the reward. Specifically, a latent embedding is computed for both the observation image $z_t = f_\phi(I_t)$ and the goal image $z_g = f_\phi(I_g)$. The policy $\pi(\cdot|z_t, z_g)$ is conditioned on both of these latent embeddings. Further, because it is assumed that the robot does not have access to the ground-truth state of the environment, it must use its observation images to compute the reward. In this past work, the reward is calculated as the negative L2 distance between the observation and goal latent embeddings, $r_t = -||z_t - z_g||_2^2$.

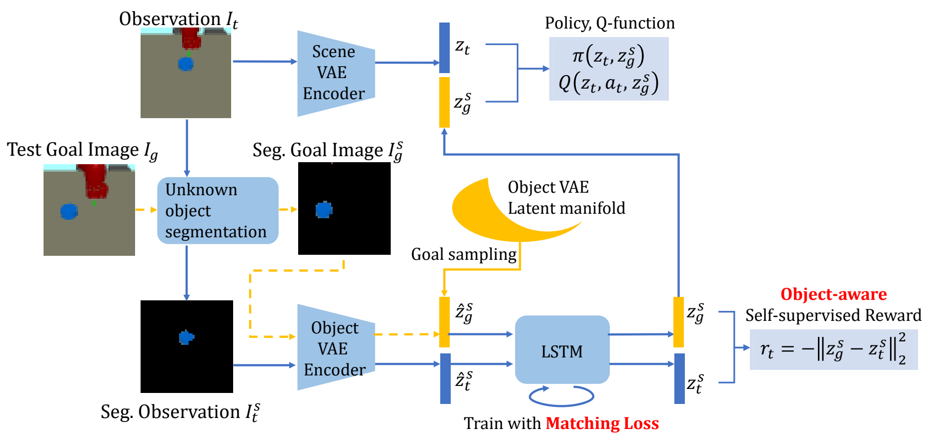


Figure 3.1: Overview of our proposed method. Top part: a scene-VAE encodes the whole image observation I_t to the latent vector z_t, which is used as input to the policy and Q-function. Bottom part: an object-VAE encodes the segmented image I_t^s to the latent vector $\hat{z}_t^s$, which is further encoded using an LSTM and used for object-aware reward computation and goal-conditioning. We train the LSTM with a novel matching loss and training scheme to make it robust to occlusions. At training time, the latent goal z_g^s is obtained by sampling from the object-VAE prior and encoding by the LSTM. During test time, we segment the goal image and then encode it to the latent goal vector.

3.2 Method

3.2.1 Key Intuition and Overview

The key intuition behind our method is that, for object manipulation tasks, goals are usually intended to refer to movable objects in the environment, rather than the robot arm or static parts of the scene. Our experiments show that, for previous methods, a robot arm can be a distractor for policy learning. Specifically, it distorts the reward function to account for the position of the robot arm information, whereas, in most cases, the intended reward function should ignore the robot arm and only consider the target object. This error occurs because the VAE encodes the entire observation image into a latent vector, including the robot arm, and this VAE encoding is used for the reward computation.

Therefore, to help the robot reason more about the target objects, we propose unknown object segmentation to segment out the target object and remove all other distractors in the scene for better reward computation and goal sampling. However, we found that, if we naively segment out the target object from the scene, then our method performs poorly when the object is occluded; hence, we use an LSTM and a novel occlusion-aware loss function and training procedure to train our method to be robust to occlusions. An overview of our method is shown in Figure 3.1; in the following subsections, we detail how we perform unknown object segmentation and use segmented images for reward computation and how we use a novel auxiliary loss to make our method robust to occlusions.

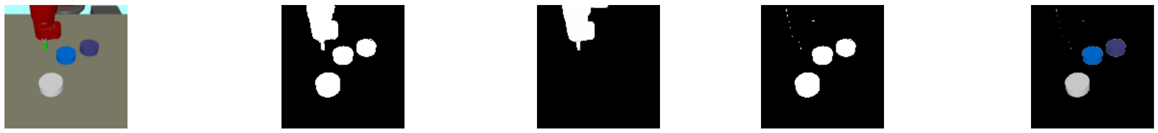

(a) original image (b) foreground mask (c) robot mask (d) object mask (e) segmented objects

Figure 3.2: An illustration of the unknown object segmentation process. The method works with various number of objects with different properties and does not require prior information about them. We do not evaluate ROLL on this environment; it's just used to visualize the segmentation process. Here we visualize the: (a) original image; (b) foreground mask obtained by background subtraction; (c) robot mask predicted by robot segmentation; (d) object mask obtained by subtracting the robot mask from the foreground mask (thresholded at 0 to remove negative values); (e) final segmented objects. Note that the unknown object segmentation process does not explicitly reason about independent object instances or object relations.

3.2.2 Unknown Object Segmentation

Our method for unknown object segmentation requires no prior information of the target object beforehand. Instead, it works by removing the pixels of everything else but the object from the scene. We note that other methods for unknown object segmentation could be used here [124, 126, 127]; the method we use was chosen for its robustness and simplicity.

We assume that the background is static, which is often the case for robot manipulation tasks. At training time, our first goal is to learn a model to segment out the robot from the scene. To do so, we make the robot move with random actions, and we record images during this movement. We then train a background subtraction module on this scene [13, 134, 135] (see Appendix for details). Because the robot is the only moving object in the scene during this data collection, it will be the only object not included as part of the background. We use this background subtraction module to generate "ground-truth" segmentation labels to segment the robot from the background; we then use these labels to train a robot segmentation network (see Appendix for details).

At test time, we add an unknown target object into the scene for robot manipulation. The background subtraction module removes the static background, leaving the pixels for the robot and the newly placed objects. The robot segmentation network is used to remove the robot pixels, leaving only the object pixels. The full test-time segmentation process is shown in Figure 3.2. As noted above, other methods could also be used for unknown object segmentation [124, 126, 127].

3.2.3 Using Segmented Images for Reward Computation

We next describe how we can use these segmented images for a more robust reward computation. See Figure 3.1 for an overview of our system. Our method employs two encoders: given the current observation I_t, a "scene-VAE" encoder f_ϕ operates on I_t and encodes it to a latent vector $z_t = f_\phi(I_t)$ that is used as the policy / Q-function input (top part of Figure 3.1). The "object-VAE" f_{ϕ^s} operates on the segmented image I_t^s (bottom part of Figure 3.1), obtained through the above method for unknown object segmentation which creates an image that contains only the objects. The object-VAE encodes the segmented image into another latent vector $\hat{z}_t^s = f_{\phi^s}(I_t^s)$. Occlusions can cause the object to not appear in the image, leading to an "empty" segmented observation or goal image. To overcome this, we augment our network with occlusion reasoning, using an LSTM as well as a novel occlusion-aware loss function (details in Section 3.2.4). The final object encoding z_t^s is the output of the LSTM using the object-VAE latent $\hat{z}_t^s$ as its input. A similar procedure is applied to the goal image I_g to obtain an encoding z_g^s of the segmented goal image (see Figure 3.1).

The object-VAE is used by our method in four ways for reward computation and goal conditioning. First, the reward is computed as the negative L2 distance between the object latent encoding of the segmented observation and goal images: $r_t = -||z_t^s - z_g^s||_2^2$. Second, the policy and Q-function take the scene-VAE latent encoding z_t as input, but they are both goal-conditioned on the object latent encoding z_g^s. Third, during training, the goal object latent encoding is sampled from the object-VAE latent manifold and encoded using the LSTM. Last, at test time, the test goal image is first segmented and then encoded using the object-VAE and the LSTM to obtain the goal object latent encoding.

By employing unknown object segmentation and using the segmented object latent encoding for reward computation and goal conditioning, we ensure that only the object information is used for guiding the learning of the policy, allowing the policy to ignore the the robot arm and the background for reward computation and goal conditioning. Our experimental results demonstrate that using the segmented object latent encoding dramatically speeds up learning in many environments.

3.2.4 Robustness to Object Occlusions

As our reward computation and goal sampling depends solely on the segmented target object, occlusions of the objects become an issue of our method. When an object is largely or totally occluded, the segmented image contains only a few (or zero) pixels, making the corresponding latent encoding non-informative for reward computation. Furthermore, if the object-VAE is trained on many images with large occlusions, many of the goals sampled from its latent manifold will also contain large occlusions, making them improper goals for policy training. An example of the robot arm occluding the puck is shown in Figure 3.3(c). In this section, we describe how we use an LSTM with a novel "matching loss" and occlusion-aware training procedure, which allows the robot to implicitly estimate the object's current position by encoding the history of the object's previous positions.

Our intuition for using an LSTM [46] to estimate the object's current position under occlusion is based on the following two observations. First, objects usually move in a smooth motion. Second, during object manipulation, an occlusion (especially by the robot arm) usually lasts for just a small number of frames. Therefore, it is feasible that the object's position in the occluded frames can be inferred from the previous positions of the object, which are encoded in the LSTM. Specifically, the object-VAE latent vector $\hat{z}_t^s$ is input to the LSTM to obtain the final object latent encoding as $z_t^s = g_\psi(\hat{z}_t^s, h_{t-1})$, where h_{t-1} is the hidden state from the previous time step and g_ψ is the LSTM encoding with parameters ψ.

We use self-supervised losses to reduce the number of samples needed to train the LSTM. First, we train the LSTM with an auto-encoder loss to reconstruct its input: $L^{ae}(\psi) = ||z_t^s -$

$\hat{z}_t^s||_2^2$. To further make the latent encoding z_t^s robust to occlusions, we propose a novel *matching loss*: given a trajectory of segmented images $I_1^s, ..., I_t^s, ..., I_T^s$, we first use the LSTM to encode them as $z_1^s, ..., z_t^s, ..., z_T^s$. Next, we randomly pick an image in the trajectory I_t^s with $t > 1$, to which we add synthetic occlusions to obtain $I_t^{s,occ}$. Since I_t^s is a segmented object image (see Figure 3.2(e)), adding a synthetic occlusion is achieved by removing additional pixels from the segmented object. Then, we create a new "occluded" trajectory, in which we replace I_t^s by the occluded image $I_t^{s,occ}$; thus, the new trajectory is given by $I_1^s, ..., I_t^{s,occ}, ..., I_T^s$. We use the LSTM to encode this new occluded trajectory to obtain the new encodings $z_1^s, ..., z_t^{s,*}, ..., z_T^{s,*}$. Note that the encodings after time step t will change since the input at time t has changed and since the LSTM also encodes the history of previous inputs.

The matching loss is designed to ensure that the encodings of the occluded images, and the images thereafter, are the same as the encodings of the unoccluded images. This loss is computed as:

$$L^{matching}(\psi) = \frac{1}{T - t + 1} \sum_{i=t}^{T} ||z_i^s - z_i^{s,*}||_2^2 \tag{3.1}$$

In other words, we force the LSTM to encode every image after the manual occlusion in the occlusion trajectory to be the same as its counterpart in the original trajectory. Specifically, for timestep t in which the input I_t^s was replaced by an occluded input image $I_t^{s,occ}$, the encodings z_t^s and $z_t^{s,*}$ are enforced to be the same. Thus, the LSTM must use the history of the images before timestep t to estimate the position of the object at timestep t in which it is occluded.

We also add a similar matching loss to the latent encoding of the object-VAE: given an image I_t^s, we add manual occlusions to it to obtain $I_t^{s,occ}$, and we require the object-VAE to encode these images as closely as possible:

$$L^{matching}(\phi^s) = ||f_{\phi^s}(I_t^s) - f_{\phi^s}(I_t^{s,occ})||_2^2 \tag{3.2}$$

Though the object-VAE is before the LSTM and has no history information, we still find that this loss leads to more stable training when combined with the matching loss on the LSTM output. In our experiments, we show that this novel matching loss allows our method to be robust to occlusions.

3.2.5 Algorithm Summary

We now explain how we train each part of our method. For the scene-VAE, we first pretrain it on a dataset of images collected using a random policy. We also continue training it using images stored in the replay buffer during RL policy learning. The scene-VAE is trained using the standard β-VAE losses, i.e., image reconstruction loss and the KL regularization loss in the latent space. For the object-VAE, we also pretrain it on a dataset of segmented images, where we randomly move the object in the scene. The object-VAE is trained using the regular β-VAE loss as well as the matching loss described in the previous section. After we pretrain the object-VAE, it is fixed during the RL training process. We pretrain the LSTM using the auto-encoder loss using the dataset of segmented images mentioned above for training the object-VAE; we also train it online using trajectories generated by the learning RL policy. This removes the burden to pre-collect a set of trajectories for training the LSTM. We train the LSTM online using the auto-encoder loss and matching loss as described in the previous section. For RL policy training, we use SAC [38] with goal re-labeling as in Hindsight Experience Replay [5]. Algorithm 2 summarizes our method. We use blue text to mark the novel steps of our method compared to RIG [81].

Algorithm 1: ROLL: Visual Self-supervised RL with Object Reasoning

```
Input: scene-VAE $f_\phi$, object-VAE $f_{\phi^s}$, LSTM $g_\psi$, policy $\pi_\theta$, Q function $Q_w$.
Collect initial scene data $\mathcal{D} = \{I_{(i)}\}$ using random initial policy; train scene-VAE $f_\phi$ on $\mathcal{D}$
Collect initial object data $\hat{\mathcal{D}} = \{\hat{I}_{(i)}\}$ by randomly placing the object in the scene
Use unknown object segmentation on $\hat{\mathcal{D}}$ to obtain segmented images $\mathcal{D}^s = \{I^s_{(i)}\}$ ; Train object-VAE $f_{\phi^s}$
  on $\mathcal{D}^s$ with image reconstruction, KL regularization, and matching loss; Train LSTM $g_\psi$ on $\mathcal{D}^s$ with
  auto-encoder loss
for $n = 0, ..., N-1$ epochs do
    Sample goal embedding from object-VAE prior $\hat{z}^s_g \sim p(\hat{z}^s)$ and encode using LSTM: $z^s_g = g_\psi(\hat{z}^s_g)$
    for $t = 0, ..., T$ steps do
        // Collect data
        Embed the observation $I_t$ with the scene-VAE $f_\phi(I_t)$
        Get action $a_t \sim \pi_\theta(f_\phi(I_t), z^s_g)$ (conditioned on object latent goal embedding)
        Execute action $a_t$ to get next observation $I_{t+1}$; store $(I_t, a_t, I_{t+1}, z^s_g)$ into replay buffer $\mathcal{R}$
        // Train the policy
        Sample transition $(I, a, I', z^s_g) \sim \mathcal{R}$
        Encode images with scene-VAE $z = f_\phi(I), z' = f_\phi(I')$
        Perform unknown object segmentation on $I$ and $I'$ to obtain $I^s$, $I^{s\prime}$
        Encode segmented images with object-VAE and LSTM $z^s = g_\psi(f_{\phi^s}(I^s)), z^{s\prime} = g_\psi(f_{\phi^s}(I^{s\prime}))$
        Hindsight Experience Replay (RIG version): With probability 0.5, replace $z^s_g$ with $z^{s\prime}_g$, which is
          sampled from object-VAE prior $\hat{z}^{s\prime}_g \sim p(\hat{z}^s)$ and embedded with LSTM: $z^{s\prime}_g = g_\psi(\hat{z}^{s\prime}_g)$
        Compute reward as $r = -||z^{s\prime} - z^s_g||^2_2$ (distance between object latent embeddings for
          observation and goal)
        Train $\pi_\theta$, $Q_w$ using scene-VAE embeddings $z, z'$ and object latent goal embedding $z^s_g$:
          $(z, a, z', z^s_g, r)$.
    end
    Perform Hindsight Experience Replay with future observations
    Train scene-VAE $f_\phi$ with image reconstruction loss and KL regularization loss
    Train LSTM $g_\psi$ with auto-encoder loss and matching loss
end
```

3.3 Experiments

In our experiments, we seek answers to the following questions: (1) Does using unknown object segmentation provides a better reward function and improve upon the baseline? (2) Does using an LSTM trained with the matching loss make ROLL robust to occlusions?

3.3.1 Setups

We evaluate ROLL on five image-based continuous control tasks simulated using the MuJoCo [111] physics simulator, where the policy must learn to manipulate objects to achieve various goals us-

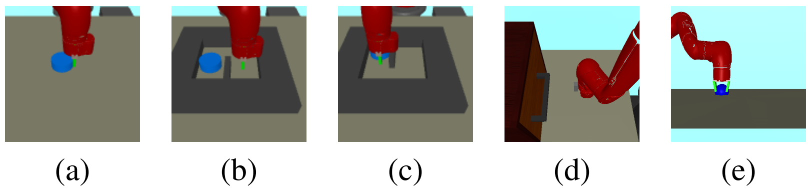


Figure 3.3: The robot view of different tasks: (a) Puck Pushing (b) Hurdle-Bottom Puck Pushing (c) Hurdle-Top Puck Pushing (d) Door Opening (e) Object Pickup.

ing only images as policy inputs, without any state-based or task-specific reward. The goals are specified using images. The robot views of different tasks are illustrated in Figure 3.3. **Puck Pushing:** a Sawyer robot arm must learn to push a puck to various goal locations. **Hurdle-Bottom Puck Pushing:** this task is similar to puck pushing, where we add hurdles. The robot arm and puck are initialized to the top right corner and the arm needs to push the puck to various locations in the left column. **Hurdle-Top Puck Pushing:** this task is the same as Hurdle-Bottom Puck Pushing except that the position of the hurdle is flipped. This poses the challenge of pushing the puck under occlusions, as the arm will largely occlude the puck when it pushes it from bottom to top (see Figure 3.3(c)). **Door Opening:** a Sawyer arm must learn to open the door to different goal angles. **Object Pickup:** a Sawyer arm must learn to pick up a ball object to various goal locations. More details on the environments are provided in the appendix.

We compare our method to Skew-Fit [91], a state-of-art visual self-supervised goal-conditioned learning algorithm which uses a single scene-VAE for both policy input and reward computation. Our method extends Skew-Fit by using an object-VAE to ignore distractors and by adding a matching loss for occlusion reasoning. In Skew-Fit, they do not pretrain the β-VAE on a set of images. As ROLL requires pre-training, for a fair comparison, we also pretrain the β-VAE in Skew-Fit using the same dataset we collect for pre-training the scene-VAE in ROLL. Implementation details and the full list of hyper-parameters and network architectures can be found in the appendix.

3.3.2 Results

ROLL significantly outperforms the baseline. Figure 3.4 shows the learning curves of all tasks. The evaluation metric is the object distance (e.g., puck distance or door angle distance) between the object in the goal image and the final object state achieved by the policy. We can see that ROLL outperforms the Skew-Fit baseline by a large margin in all tasks except for Door Opening, where it achieves a similar final performance. For all other tasks, ROLL not only obtains better final performance but also learns dramatically faster. We note that Skew-Fit performs especially poorly in the Hurdle Puck Pushing tasks. This is because, when provided a goal image where the arm and puck are at different locations, the policy learned by Skew-Fit always aligns the arm instead of the puck (see Figure 3.6, right). Instead, by focusing on the object in the reward computation and goal conditioning, the policy learned by ROLL always aligns the puck and ignores the arm.

Segmented images provide a better reward function. To analyze why our method performs so well, we verify if the reward function derived from the latent space of the object-VAE

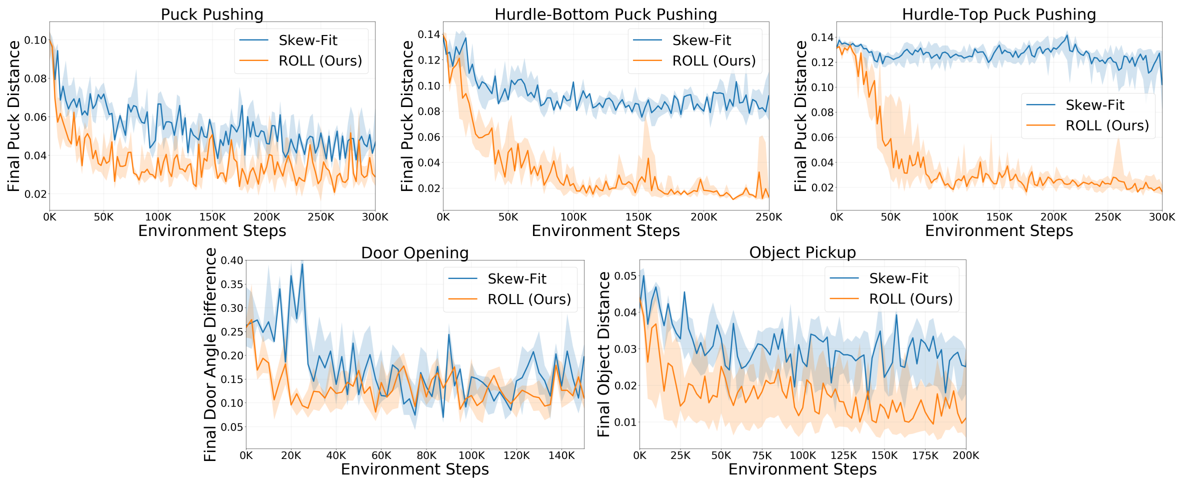


Figure 3.4: Learning curves on 5 simulation tasks. The solid line shows the median of 6 seeds, and the dashed region shows the 25% and 75% percentile.

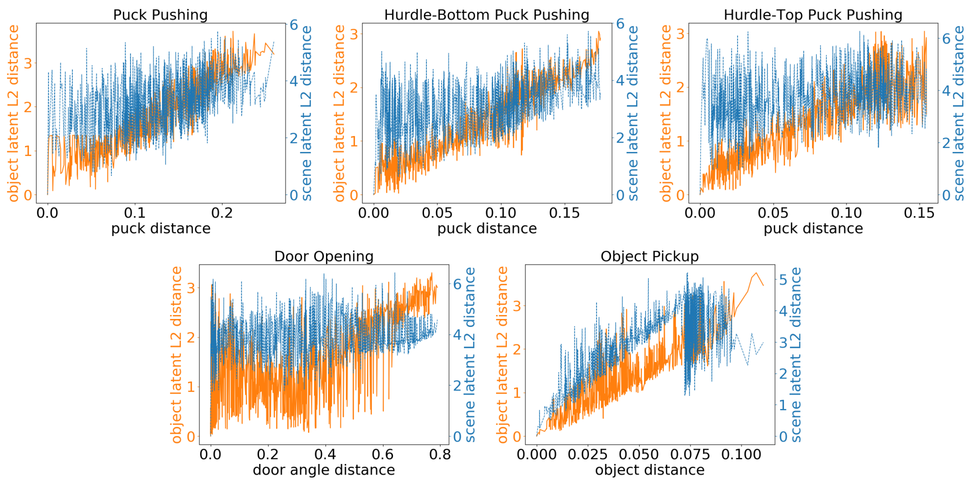
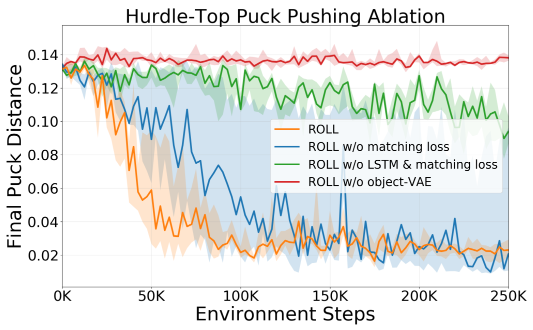


Figure 3.5: Comparison of the manipulated object distance (x-axis) against the distance in scene latent embedding (blue) and the distance in object latent embedding (orange). As can be seen, the object latent distance correlates much more strongly with the manipulated object distance than the scene-VAE latent distance used in prior work.



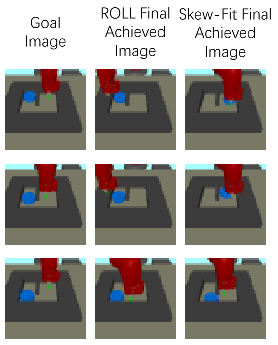


Figure 3.6: Left: Learning curves on ablations of not using the object-VAE, the LSTM, or the matching loss on the Hurdle-Top Puck Pushing task. Right: Policy visualizations in the Hurdle-Top Puck Pushing task. We see that Skew-Fit often ignores the puck and only chooses to align the arm, while ROLL always successfully aligns the puck.

is better than that derived from the scene-VAE. For a better reward function, the distance in the latent space should better approximate the distance to the real object, e.g., the puck/ball distance in the pushing/pickup tasks and the door angle distance in the door opening task. In Figure 3.5, we plot the object distance along the x-axis and the latent distance along the y-axis, where the distance is measured between a set of observation images and a single goal image. A good latent distance should correlate with the real object distance. As can be seen, the latent distance from the object-VAE is much more accurate and stable in approximating the real object distance in all five tasks. This verifies our intuition that the reward latent encoding should only include information from the target object, rather than the robot arm or other static parts in the scene.

Ablation study. To test whether each component of our method is necessary, we perform ablations of our method in the Hurdle-Top Puck Pushing task, which has large occlusions on the optimal path. We test three variants of our method: *ROLL without matching loss*, which does not add the matching loss to the LSTM output z^s or the object-VAE latent embedding $\hat{z}^s$; *ROLL without LSTM and matching loss*, which does not add an LSTM after the object-VAE and uses no matching loss; *ROLL without object-VAE*, which replaces the object-VAE in ROLL with the scene-VAE but still uses an LSTM and the matching loss. The results are shown in the left subplot of Figure 3.6. We can see that without the the matching loss, the policy learns slower and has very large variance; without the LSTM, the policy learns very poorly; without the object-VAE, the policy cannot learn at all. This verifies the necessity of using the object-VAE, the LSTM, and the matching loss in ROLL.

Chapter 4

f-IRL: Inverse Reinforcement Learning via State Marginal Matching

4.1 Preliminaries

In this section, we review notation on maximum entropy (MaxEnt) RL [63] and state marginal matching (SMM) [61] that we build upon in this work.

MaxEnt RL. Consider a Markov Decision Process (MDP) represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, T)$ with state-space $\mathcal{S}$, action-space $\mathcal{A}$, dynamics $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, reward function $r(s, a)$, initial state distribution ρ_0, and horizon T. The optimal policy π under the maximum entropy framework [132] maximizes the objective $\sum_{t=1}^{T} \mathbb{E}_{\rho_{\pi,t}(s_t,a_t)}[r(s_t, a_t) + \alpha H(\cdot|s_t)]$. Here $\rho_{\pi,t}$ is the state-action marginal distribution of policy π at timestamp t, and $\alpha > 0$ is the entropy temperature.

Let $r_\theta(s)$ be a parameterized differentiable reward function only dependent on state. Let trajectory τ be a time series of visited states $\tau = (s_0, s_1, \ldots, s_T)$. The optimal MaxEnt trajectory distribution $\rho_\theta(\tau)$ under reward r_θ can be computed as $\rho_\theta(\tau) = \frac{1}{Z} p(\tau) e^{r_\theta(\tau)/\alpha}$, where

$$p(\tau) = \rho_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t) \ , \quad r_\theta(\tau) = \sum_{t=1}^{T} r_\theta(s_t), \quad Z = \int p(\tau) e^{r_\theta(\tau)/\alpha} d\tau.$$

Slightly overloading the notation, the optimal MaxEnt state marginal distribution $\rho_\theta(s)$ under reward r_θ is obtained by marginalization:

$$\rho_\theta(s) \propto \int p(\tau) e^{r_\theta(\tau)/\alpha} \eta_\tau(s) d\tau \tag{4.1}$$

where $\eta_\tau(s) \triangleq \sum_{t=1}^{T} 1(s_t = s)$ is the visitation count of a state s in a particular trajectory τ.

State Marginal Matching. Given the expert state density $p_E(s)$, one can train a policy to match the expert behavior by minimizing the following f-divergence objective:

$$L_f(\theta) = D_f(\rho_E(s) \,||\, \rho_\theta(s)) \tag{4.2}$$

where common choices for the f-divergence D_f [4, 34] include forward KL divergence, reverse KL divergence, and Jensen-Shannon divergence. Our proposed f-IRL algorithm will compute

Name	f-divergence $D_f(P \mid\mid Q)$	Generator $f(u)$	$h_f(u)$
FKL	$\int p(x) \log \frac{p(x)}{q(x)} dx$	$u \log u$	$-u$
RKL	$\int q(x) \log \frac{q(x)}{p(x)} dx$	$-\log u$	$1 - \log u$
JS	$\frac{1}{2} \int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} dx$	$u \log u - (1+u) \log \frac{1+u}{2}$	$-\log(1+u)$

Table 4.1: Selected list of f-divergences $D_f(P \mid\mid Q)$ with generator functions f and h_f defined in Theorem 4.2.1, where f is convex, lower-semicontinuous and $f(1) = 0$.

the analytical gradient of Eq. 4.2 w.r.t. θ and use it to optimize the reward function via gradient descent.

4.2 Learning Stationary Rewards via State-Marginal Matching

In this section, we describe our algorithm f-IRL, which takes the expert state density as input, and optimizes the f-divergence objective (Eq. 4.2) via gradient descent. Our algorithm trains a policy whose state marginal is close to that of the expert, and a corresponding stationary reward function that would produce the same policy if the policy were trained with MaxEnt RL from scratch.

4.2.1 Analytic Gradient for State Marginal Matching in f-divergence

One of our main contributions is the exact gradient of the f-divergence objective (Eq. 4.2) w.r.t. the reward parameters θ. This gradient will be used by f-IRL to optimize Eq. 4.2 via gradient descent. The proof is provided in Appendix B.1.

Theorem 4.2.1 (**f-divergence analytic gradient**). *The analytic gradient of the f-divergence $L_f(\theta)$ between state marginals of the expert (ρ_E) and the soft-optimal agent w.r.t. the reward parameters θ is given by:*

$$\nabla_\theta L_f(\theta) = \frac{1}{\alpha T} \text{cov}_{\tau \sim \rho_\theta(\tau)} \left(\sum_{t=1}^{T} h_f \left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)} \right), \sum_{t=1}^{T} \nabla_\theta r_\theta(s_t) \right) \tag{4.3}$$

where $h_f(u) \triangleq f(u) - f'(u)u$, $\rho_E(s)$ is the expert state marginal and $\rho_\theta(s)$ is the state marginal of the soft-optimal agent under the reward function r_θ, and the covariance is taken under the agent's trajectory distribution $\rho_\theta(\tau)$.[1]

Choosing the f-divergence to be Forward Kullback-Leibler (FKL), Reverse Kullback-Leibler (RKL), or Jensen-Shannon (JS) instantiates h_f (see Table 4.1). Note that the gradient of the RKL objective has a special property in that we can specify the expert as an *unnormalized* log-density (i.e. energy), since in $h_{\text{RKL}}(\frac{\rho_E(s)}{\rho_\theta(s)}) = 1 - \log \rho_E(s) + \log \rho_\theta(s)$, the normalizing factor of $\rho_E(s)$ does not change the gradient (by linearity of covariance). This makes density specification much

[1]Here we assume f is differentiable, which is often the case for common f-divergence (e.g. KL divergence).

easier in a number of scenarios. Intuitively, since h_f is a monotonically decreasing function ($h'_f(u) = -f''(u)u < 0$) over $\mathbb{R}^+$, the gradient descent tells the reward function to increase the rewards of those state trajectories that have higher sum of density ratios $\sum_{t=1}^{T} \frac{\rho_E(s_t)}{\rho_\theta(s_t)}$ so as to minimize the objective.

4.2.2 Learning a Stationary Reward by Gradient Descent

We now build upon Theorem 4.2.1 to design a practical algorithm for learning the reward function r_θ (Algorithm. 2). Given expert information (state density or observation samples) and an arbitrary f-divergence, the algorithm alternates between using MaxEnt RL with the current reward, and updating the reward parameter using gradient descent based on the analytic gradient.

If the provided expert data is in the form of expert state density $\rho_E(s)$, we can fit a density model $\hat{\rho}_\theta(s)$ to estimate agent state density $\rho_\theta(s)$ and thus estimate the density ratio required in gradient. If we are given samples from expert observations s_E, we can fit a discriminator $D_\omega(s)$ in each iteration to estimate the density ratio by optimizing the binary cross-entropy loss:

$$\max_\omega \mathbb{E}_{s\sim s_E}[\log D_\omega(s)] + \mathbb{E}_{s\sim\rho_\theta(s)}[\log(1 - D_\omega(s)] \tag{4.4}$$

where the optimal discriminator satisfies $D^*_\omega(s) = \frac{\rho_E(s)}{\rho_E(s)+\rho_\theta(s)}$ [35], thus the density ratio can be estimated by $\frac{\rho_E(s)}{\rho_\theta(s)} \approx \frac{D_\omega(s)}{1-D_\omega(s)}$, which is the input to h_f.

Algorithm 2: Inverse RL via State Marginal Matching (f-IRL)

```
Input  : Expert state density $\rho_E(s)$ or expert observations $s_E$ , $f$-divergence
Output: Learned reward $r_\theta$, Policy $\pi_\theta$
Initialize $r_\theta$, and density estimation model (provided $\rho_E(s)$) or disciminator $D_\omega$
  (provided $s_E$)
for $i \leftarrow 1$ to $Iter$ do
    $\pi_\theta \leftarrow$ MaxEntRL($r_\theta$) and collect agent trajectories $\tau_\theta$
    if provided $\rho_E(s)$ then
        Fit the density model $\hat{\rho}_\theta(s)$ to the state samples from $\tau_\theta$
    end
    else
        // provided $s_E$
        Fit the discriminator $D_\omega$ by Eq. 4.4 using expert and agent state samples from $s_E$
          and $\tau_\theta$
    end
    Compute sample gradient $\hat{\nabla}_\theta L_f(\theta)$ for Eq. 4.3 over $\tau_\theta$
    $\theta \leftarrow \theta - \lambda\hat{\nabla}_\theta L_f(\theta)$
end
```

4.2.3 Robust Reward Recovery under State-only Ground-truth Reward

IRL methods are different from IL methods in that they recover a reward function in addition to the policy. A hurdle in this process is often the reward ambiguity problem, explored in [32, 82]. This ambiguity arises due to the fact that the optimal policy remains unchanged under the following reward transformation [82]:

$$\hat{r}(s, a, s') = r_{\text{gt}}(s, a, s') + \gamma\Phi(s') - \Phi(s) \tag{4.5}$$

for any function Φ. In the case where the ground-truth reward is a function over states only (i.e., $r_{\text{gt}}(s)$), f-IRL is able to recover the *disentangled* reward function (r_{IRL}) that matches the ground truth reward r_{gt} up to a constant. The obtained reward function is robust to different dynamics – for any underlying dynamics, r_{IRL} will produce the same optimal policy as r_{gt}. We formalize this claim in Appendix B.1.4 (based on Theorem 5.1 of AIRL [32]).

AIRL uses a special parameterization of the discriminator to learn state-only rewards. A disadvantage of their approach is that AIRL needs to approximate a separate reward-shaping network apart from the reward network. In contrast, our method naturally recovers a state-only reward function.

4.2.4 Practical Modification in the Exact Gradient

In practice with high-dimensional observations, when the agent's current trajectory distribution is far off from the expert trajectory distribution, we find that there is little supervision available through our derived gradient, leading to slow learning. Therefore, when expert trajectories are provided, we bias the gradient (Eq. 4.3) using a mixture of agent and expert trajectories inspired by GCL [30], which allows for richer supervision and faster convergence. Note that at convergence, the gradient becomes unbiased as the agent's and expert's trajectory distribution matches.

$$\tilde{\nabla}_\theta L_f(\theta) := \frac{1}{\alpha T}\text{cov}_{\tau\sim\frac{1}{2}(\rho_\theta(\tau)+\rho_E(\tau))}\left(\sum_{t=1}^{T} h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right), \sum_{t=1}^{T}\nabla_\theta r_\theta(s_t)\right) \tag{4.6}$$

where the expert trajectory distribution $\rho_E(\tau)$ is uniform over samples τ_E.

4.3 Experiments

In our experiments, we seek answers to the following questions:

1. Can f-IRL learn a policy that matches the given expert state density?
2. Can f-IRL learn good policies on high-dimensional continuous control tasks in a sample-efficient manner?
3. Can f-IRL learn a reward function that induces the expert policy?
4. How can learning a stationary reward function help solve downstream tasks?

Comparisons. To answer these questions, we compare f-IRL against two classes of existing imitation learning algorithms: (1) those that learn only the policy, including Behavior Cloning (BC), GAIL [45], and f-MAX-RKL[2] [34]; and (2) IRL methods that learn both a reward and a policy simultaneously, including MaxEnt IRL [133] and AIRL [32]. The rewards/discriminators of the baselines are parameterized to be state-only. We use SAC [36] as the base MaxEnt RL algorithm. Since the original AIRL uses TRPO [103], we re-implement a version of AIRL that uses SAC as the underlying RL algorithm for fair comparison. For our method (f-IRL), MaxEnt IRL, and AIRL, we use a MLP for reward parameterization.

Tasks. We evaluate the algorithms on several tasks:

- **Matching Expert State Density**: In Section 4.3.1, the task is to learn a policy that matches the given expert state density.
- **Inverse Reinforcement Learning Benchmarks**: In Section 4.3.2, the task is to learn a reward function and a policy from expert trajectory samples. We collected expert trajectories by training SAC [36] to convergence on each environment. We trained all the methods using varying numbers of expert trajectories $\{1, 4, 16\}$ to test the robustness of each method to the amount of available expert data.
- **Using the Learned Reward for Downstream Tasks**: In Section 4.3.3, we first train each algorithm to convergence, then use the learned reward function to train a new policy on a related downstream task. We measure the performance on downstream tasks for evaluation.

We use five MuJoCo continuous control locomotion environments [15, 111] with joint torque actions, illustrated in Figure 4.1. Further details about the environment, expert information (samples or density specification), and hyperparameter choices can be found in Appendix B.3.

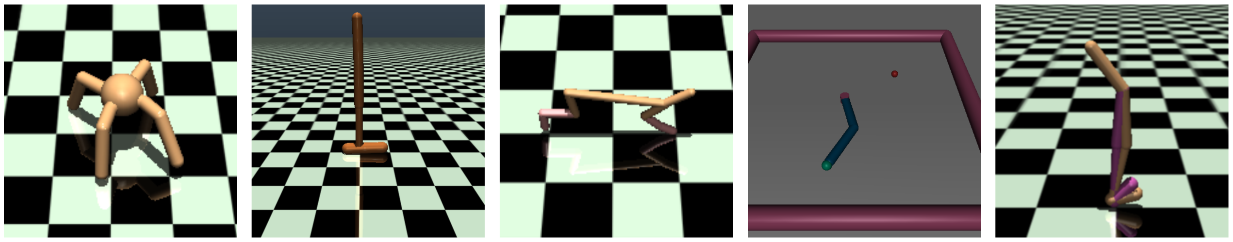
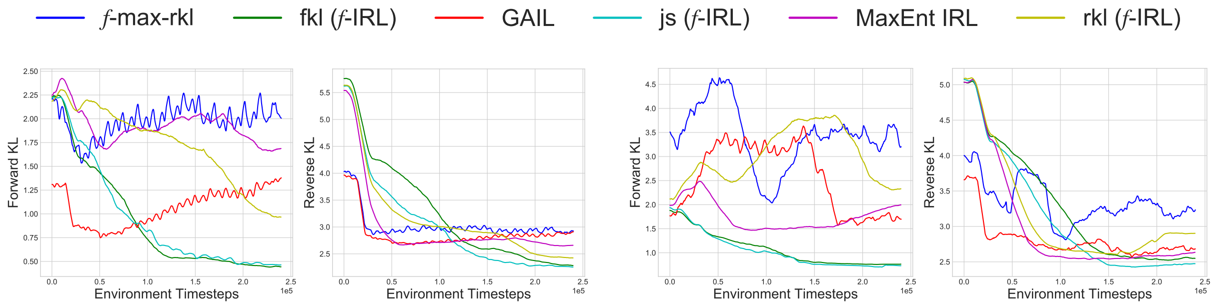


Figure 4.1: **Environments**: (left to right) Ant-v2, Hopper-v2, HalfCheetah-v2, Reacher-v2, and Walker2d-v2.

4.3.1 Matching the Specified Expert State Density

First, we check whether f-IRL can learn a policy that matches the given expert state density of the fingertip of the robotic arm in the 2-DOF Reacher environment. We evaluate the algorithms using two different expert state marginals: (1) a Gaussian distribution centered at the goal for single goal-reaching, and (2) a mixture of two Gaussians, each centered at one goal. Since this problem setting assumes access to the expert density only, we use importance sampling to generate expert samples required by the baselines.

[2] A variant of AIRL [32] proposed in [34] only learns a policy and does not learn a reward.



(a) Expert Density: Gaussian (b) Expert Density: Mixture of two Gaussians

Figure 4.2: Forward (left) and Reverse (right) KL curves in the Reacher environment for different expert densities of all methods. Curves are smoothed in a window of 120 evaluations.

In Figure 4.2, we report the estimated forward and reverse KL divergences in state marginals between the expert and the learned policy. For f-IRL and MaxEnt IRL, we use Kernel Density Estimation (KDE) to estimate the agent's state marginal. We observe that the baselines demonstrate unstable convergence, which might be because those methods optimize the f-divergence approximately. Our method $\{$FKL, JS$\}$ f-IRL outperforms the baselines in the forward KL and the reverse KL metric, respectively.

4.3.2 Inverse Reinforcement Learning Benchmarks

Next, we compare f-IRL and the baselines on IRL benchmarks, where the task is to learn a reward function and a policy from expert trajectory samples. We use the modification proposed in Section 4.2.4 to alleviate the difficulty in optimizing the f-IRL objective with high-dimensional states.

Policy Performance. We check whether f-IRL can learn good policies on high-dimensional continuous control tasks in a sample-efficient manner from expert trajectories.

Figure 4.3 shows the learning curves of each method in the four environments with *one* expert trajectory provided. f-IRL and MaxEnt IRL demonstrate much faster convergence in most of the tasks than f-MAX-RKL. Table 4.2 shows the final performance of each method in the four tasks, measured by the ratio of agent returns (evaluated using the ground-truth reward) to expert returns.[3] While MaxEnt IRL provides a strong baseline, f-IRL outperforms all baselines on most tasks especially in Ant, where the FKL (f-IRL) has much higher final performance and is less sensitive to the number of expert trajectories compared to the baselines. In contrast, we found the original implementation of f-MAX-RKL to be extremely sensitive to hyperparameter settings. We also found that AIRL performs poorly even after tremendous tuning, similar to the findings in [69, 70].

Recovering the Stationary Reward Function. We also evaluate whether f-IRL can recover a stationary reward function that induces the expert policy.

To do so, we train a SAC agent from scratch to convergence using the reward model obtained from each IRL method. We then evaluate the trained agents using the ground-truth reward to test

[3]The unnormalized agent and expert returns are reported in Appendix B.4.

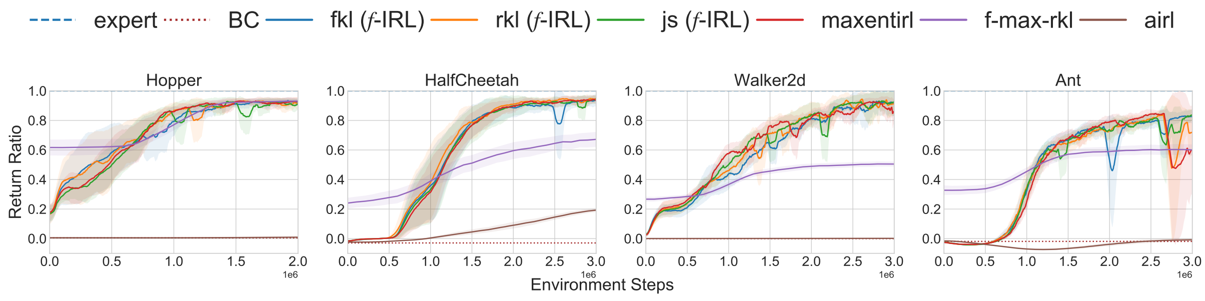


Figure 4.3: Training curves for f-IRL and 4 other baselines - BC, MaxEnt IRL, f-MAX-RKL and AIRL with one expert demonstration. Solid curves depict the mean of 3 trials and the shaded area shows the standard deviation. The dashed blue line represents the expert performance and the dashed red line shows the performance of a BC agent at convergence.

Method	Hopper			Walker2d			HalfCheetah			Ant		
Expert return	3592.63 ± 19.21			5344.21 ± 84.45			12427.49 ± 486.38			5926.18 ± 124.56		
# Expert traj	1	4	16	1	4	16	1	4	16	1	4	16
BC	0.00	0.13	0.16	0.00	0.05	0.08	0.00	0.01	0.02	0.00	0.22	0.47
MaxEnt IRL	0.93	0.92	**0.94**	0.88	0.88	**0.91**	**0.95**	**0.98**	0.91	0.54	0.71	0.81
f-MAX-RKL	**0.94**	**0.93**	0.91	0.49	0.49	0.47	0.71	0.41	0.65	0.60	0.65	0.62
AIRL	0.01	0.01	0.01	0.00	0.00	0.00	0.19	0.19	0.19	0.00	0.00	0.00
FKL (f-IRL)	0.93	0.90	0.93	**0.90**	0.90	0.90	0.94	0.97	0.94	**0.82**	**0.83**	**0.84**
RKL (f-IRL)	0.93	0.92	0.93	0.89	0.90	0.85	**0.95**	0.97	**0.96**	0.63	0.82	0.81
JS (f-IRL)	0.92	**0.93**	**0.94**	0.89	**0.92**	0.88	0.93	**0.98**	0.94	0.77	0.81	0.73

Table 4.2: We report the ratio between the average return of the trained (stochastic) policy vs. that of the expert policy for different IRL algorithms using 1, 4 and 16 expert trajectories. All results are averaged across 3 seeds. Negative ratios are clipped to zero.

whether the learned reward functions are good at inducing the expert policies.

Table 4.3 shows the ratio of the final returns of policy trained from scratch using the rewards learned from different IRL methods with one expert trajectory provided, to expert returns. Our results show that MaxEnt IRL and f-IRL are able to learn *stationary* rewards that can induce a policy close to the optimal expert policy.

4.3.3 Using the Learned Stationary Reward for Downstream Tasks

Finally, we investigate how the learned stationary reward can be used to learn related, downstream tasks.

Reward prior for downstream hard-exploration tasks. We first demonstrate the utility of the learned stationary reward by using it as a prior reward for the downstream task. Specifically, we construct a didactic point mass environment that operates under linear dynamics in a 2D 6×6 room, and actions are restricted to $[-1, 1]$. The prior reward is obtained from a *uniform* expert

Method	Hopper	Walker2d	HalfCheetah	Ant
AIRL	-	-	-0.03	-
MaxEntIRL	**0.93**	**0.92**	0.96	0.79
f-IRL	**0.93**	0.88	**1.02**	**0.82**

Table 4.3: The ratios of final return of the obtained policy against expert return across IRL methods. We average f-IRL over FKL, RKL, and JS. '-' indicates that we do not test learned rewards since AIRL does poorly at these tasks in Table 4.2.

Policy Transfer using GAIL	AIRL	MaxEntIRL	f-IRL	Ground-truth Reward
-29.9	130.3	**145.5**	141.1	315.5

Table 4.4: Returns obtained after transferring the policy/reward on modified Ant environment using different IL methods.

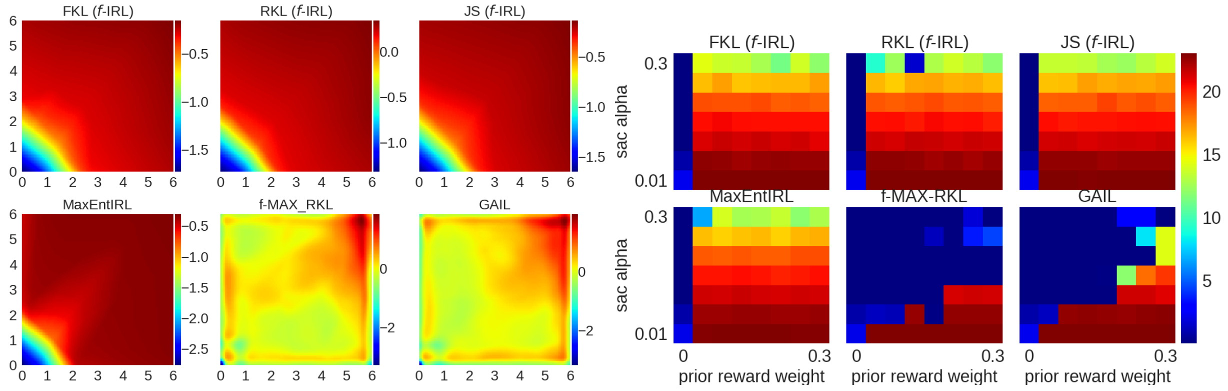


Figure 4.4: Left: Extracted final reward of all compared methods for the uniform expert density in the point environment. Right: The task return (in terms of r_{task}) with different α and prior reward weight λ. The performance of vanilla SAC is shown in the leftmost column with $\lambda = 0$ in each subplot.

density over the whole state space, and is used to ease the learning in the hard-exploration task, where we design a difficult goal to reach with distraction rewards (full details in appendix B.3).

We use the learned prior reward r_{prior} to augment the task reward r_{task} as follows: $r(s) = r_{\text{task}}(s) + \lambda(\gamma r_{\text{prior}}(s') - r_{\text{prior}}(s))$. The main theoretical result of [82] dictates that adding a potential-based reward in this form will not change the optimal policy.

GAIL and f-MAX-RKL do not extract a reward function but rather a discriminator, so we derive a prior reward from the discriminator in the same way as [34, 45].

Figure 4.4 illustrates that the reward recovered by $\{$FKL, RKL, JS$\}$ f-IRL and the baseline MaxEnt IRL are similar: the reward increases as the distance to the agent's start position, the bottom left corner, increases. This is intuitive for achieving the target uniform density: states farther away should have higher rewards. f-MAX-RKL and GAIL's discriminator demonstrate a different pattern which does not induce a uniform state distribution.

The leftmost column in the Figure 4.4 (Right) shows the poor performance of SAC training without reward augmentation ($\lambda = 0$). This verifies the difficulty in exploration for solving the task. We vary λ in the x-axis, and α in SAC in the y-axis, and plot the final task return (in terms of r_{task}) as a heatmap in the figure. The presence of larger red region in the heatmap shows that our method can extract a prior reward that is more robust and effective in helping the downstream task attain better final performance with its original reward.

Reward transfer across changing dynamics. Lastly, we evaluate the algorithms on transfer learning across different environment dynamics, following the setup from [32]. In this setup, IL algorithms are provided expert trajectories from a quadrupedal ant agent which runs forward. The algorithms are tested on an ant with two of its legs being disabled and shrunk. This requires the ant to significantly change its gait to adapt to the disabled legs for running forward.

We found that a forward-running policy obtained by GAIL fails to transfer to the disabled ant. In contrast, IRL algorithms such as f-IRL are successfully able to learn the expert's reward function using expert demonstrations from the quadrupedal ant, and use the reward to train a policy on the disabled ant. The results in Table 4.4 show that the reward learned by f-IRL is robust and enables the agent to learn to move forward with just the remaining two legs.

Chapter 5

SoftGym: Benchmarking Deep Reinforcement Learning for Deformable Object Manipulation

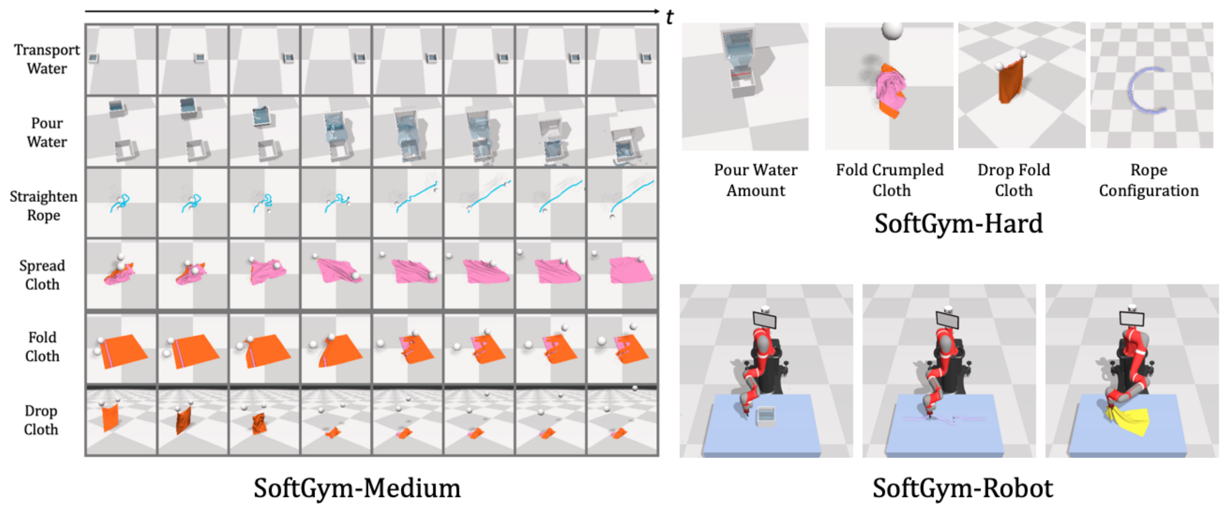


Figure 5.1: Visualizations of all tasks in SoftGym. These tasks can be used to evaluate how well an algorithm works on a variety of deformable object manipulation tasks.

5.1 Background: Deformable Object Modeling in FleX

SoftGym builds on top of Nvidia FleX physics simulator. Nvidia's FleX simulator models deformable objects in a particle and position based dynamical system [72, 79]. Each object is represented by a set of particles and the internal constraints among these particles. Each particle p_i has at least three attributes: position x_i, velocity v_i, and inverse mass w_i. Different physical properties of the objects are characterized by the constraints. A constraint is represented in the form of $C(x_1, ..., x_n) \geq 0$ or $C(x_1, ..., x_n) = 0$, where $C(x)$ is a function of all the positions of the relevant particles. Given the current particle positions p_i and velocities v_i, FleX first computes a predicted position $\hat{p}_i = p_i + \Delta t v_i$ by integrating the velocity. The predicted positions are then projected onto the feasible set given all the constraints to obtain the new positions of the particles in the next step.

Fluids can naturally be modeled in a particle system, as detailed in [71]; specifically, a constant density constraint is applied to each particle to enforce the incompressibility of the fluid. For each particle, the constant density constraint is based on the position of that particle, as well as the positions of its neighboring particles.

Rope is modeled as a sequence of particles, where each pair of neighbouring particles are connected by a spring. Cloth is modeled as a grid of particles. Each particle is connected to its eight neighbors by a spring, i.e. a stretching constraint. Additionally, for particles that are two-steps away from each other, a bending constraint is used to model the resistance against bending deformation. Additional constraints for modeling self-collision are applied. We refer the readers to [79] for more details.

5.2 SoftGym

To advance research in reinforcement learning in complex environments with an inherently high dimensional state, we propose SoftGym. SoftGym includes a set of tasks related to manipulating deformable objects including rope, cloth, and fluids. As a result, the underlying state representation for the dynamics has a dimension ranging from hundreds to thousands, depending on the number of particles that are used.

SoftGym consists of three parts: SoftGym-Medium, SoftGym-Hard and SoftGym-Robot, visualized in Figure 5.1. SoftGym-Medium includes six tasks where we provide extensive benchmarking results. Four more challenging tasks are included in SoftGym-Hard. SoftGym-Medium and SoftGym-Hard using an abstract action space while SoftGym-Robot includes tasks with a Sawyer or Franka robot as the action space. [1] We describe the details of the action space below.

5.2.1 Action Space

We aim to decouple the challenges in learning low-level grasping skills from high-level planning. As such, we employ abstract action spaces for tasks in SoftGym-Medium and SoftGym-Hard. For rope and cloth manipulation, we use pickers, which are simplifications of a robot gripper and are modeled as spheres which can move freely in the space. A picker can be activated, in which case if it is close to any object, the particle on the object that is the closest to the picker will be attached to the picker and moves with it. More specifically, the action of the agent is a vector of length $4n$, where n is the number of the pickers. For each picker, the agent outputs (d_x, d_y, d_z, p), where d_x, d_y, d_z determine the movement of the picker and p determines whether the picker is activated (picking cloth, $p \geq 0.5$) or deactivated (not picking cloth, $p < 0.5$). For fluid related tasks, we directly actuate the cup holding the fluid. This action space is designed to enable the user to focus on the challenges of high-level planning and to abstract away the low-level manipulation.

[1] SoftGym-Robot will only be released after obtaining permission from Nvidia.

Still, in order to reflect the challenges of robotic manipulation, we also provide SoftGym-Robot, where either a Sawyer robot or a Franka robot is used to manipulate objects in the environment (Figure 5.1, bottom-right). Cartesian control of the end-effector is used.

5.2.2 Tasks

SoftGym-Medium includes six tasks (see Appendix for more details):

TransportWater Move a cup of water to a target position as fast as possible without spilling out the water. The movement of the cup is restricted in a straight line; thus the action of the agent is just a scalar $a = d_x$ that specifies the displacement of the cup. The reward of this task is the negative distance to the target position, with a penalty on the fraction of water that is spilled out.

PourWater Pour a cup of water into a target cup. The agent directly controls the position and rotation of the cup at hand. Specifically, the action space of the agent is a vector $a = (d_x, d_y, d_\theta)$, representing the displacement of the cup in the x, y dimension and its rotation around its geometric center. The reward is the fraction of water that is successfully poured into the target cup.

StraightenRope Straighten a rope starting from a random configuration. The agent controls two pickers. The reward is the negative absolute difference between the current distance of the two endpoints of the rope, and the rope length when it is straightened.

SpreadCloth Spread a crumpled cloth on the floor. The agent controls two pickers to spread the cloth. The reward for this task is the area covered by the cloth when viewed top-down.

FoldCloth Fold a piece of flattened cloth in half. The agent controls two pickers. The reward for this task is the distance of the corresponding particles between the two halves of the cloth; we also add a penalty based on the displacement of the cloth from its original position, i.e., we do not want to agent to drag the cloth too far away while folding it.

DropCloth This task begins with the agent holding two corners of a piece of cloth with two pickers in the air, and the goal is to lay the cloth flat on the floor. The action space of the agent is the same as that in SpreadCloth, with the additional constraint that the pickers cannot move too close to the floor (i.e. below a given height threshold). As such, a swinging and dropping motion is required to perform the task; dropping the cloth without swinging will result in the cloth being crumpled. The reward of this task is the mean particle-wise L_2 distance between the current cloth and a target cloth configuration flattened on the floor.

SoftGym-Hard contains four more tasks:

PourWaterAmount This task is similar to PourWater but requires a specific amount of water poured into the target cup, indicated either in the state representation and marked by a red line in the visual observation.

FoldCrumpledCloth This task is similar to FoldCloth but the cloth is initially crumpled. Thus, the agent may need to spread and fold the cloth at the same time.

DropFoldCloth This task has the same initial state as DropCloth but requires the agent to fold the cloth instead of just laying it on the ground.

RopeConfiguration This task is similar to StraightenCloth but the agent needs to manipulate the rope into a specific configuration from different starting locations. Different goal configurations in the shape of letters of the alphabet can be specified. The reward is computed by finding the minimum bipartite matching distance between the current particle positions and the goal particle positions [119].

FleX uses a GPU for accelerating simulation. On a Nvidia 2080Ti GPU, all SoftGym tasks run about 4x faster than real time, with rendering. One million simulation steps takes 6 hours (wall-clock time) and corresponds to at least 35 hours for a real robot to collect. More details on the environments, including the variations for each task, can be found in the Appendix.

5.3 Methods Evaluated

We benchmark a few representative policy search algorithms on the tasks in SoftGym. We group these algorithms into categories which make different assumptions regarding knowledge about the underlying dynamics or position of particles in the environments. These algorithms allow us to analyze different aspects of the challenges in learning to manipulate deformable objects. We use the ground-truth reward function for all the baselines, although such rewards are not readily available outside of the simulation environment; computing rewards from high-dimensional observations is an additional challenge that is outside the scope of this paper. We refer to [31, 68, 81, 108] for recent works towards this challenge.

5.3.1 Dynamics Oracle

The Dynamics Oracle has access to the ground-truth positions and velocities of all particles in the deformable objects, as well as access to the ground-truth dynamics model. This information is only accessible in simulation. Given this information, we can use gradient free optimization to maximize the return. In this category we benchmark the cross entropy method (CEM) [97] as a representative random shooting algorithm. CEM optimizes over action sequences to find the trajectory with the highest return, given the action sequence and the ground-truth dynamics model. Because the ground-truth dynamics model is used, no training is required (i.e. no parameters are learned) for this method. We use model predictive control (MPC) for executing the planned trajectories. This baseline shows what can be achieved from a trajectory optimization approach given ground-truth particle positions, velocities, and dynamics; we would expect that methods that only have access to visual observations would have worse performance. However, the performance of the Dynamics Oracle may still be limited due to the exploration strategy employed by CEM.

5.3.2 State Oracle

Many robotic systems follow the paradigm of first performing state estimation and then using the estimated state as input to a policy. Deformable objects present a unique challenge for manipulation where the dynamical system has a high dimensional state representation, i.e. the position of all particles in the deformable objects. For such high dimensional systems, state estimation becomes harder; furthermore, even assuming perfect state estimation, the high dimensional state space is challenging for any reinforcement learning agent. We explore two state-based methods to explore these challenges.

Full State Oracle This method has access to the ground-truth positions of all of the particles in the target object as well as any proprioception information of the robot or the picker, but it does not have access to the ground-truth dynamics. We use these positions as input to a policy trained using SAC [37]; we use the standard multi-layer perceptron (MLP) as the architecture for the agent. As the observational dimension can vary, e.g. cloths with different sizes can have different numbers of particles, we take the maximum number of particles as the fixed input dimension and pad the observation with 0 when it has fewer dimensions. Generally, we expect this baseline to perform poorly, since concatenating the positions of all particles in a single vector as input forces the network to learn the geometric structure of the particles, which adds significantly to the complexity of the learning problem. Given additional task-specific information about the connectivity among the particles, alternative architectures such as graph neural networks may be better at capturing the structure of the particles [66].

Reduced State Oracle To avoid the challenges of RL from high-dimensional state spaces, this method uses a hand-defined reduced set of the full state as input to the policy. This baseline uses the same SAC algorithm as the Full State Oracle to train the policy. Estimating this reduced state from a high-dimensional observation of a deformable object, such as from an image of crumpled cloth, may be challenging; for this oracle baseline, we assume that such a reduced state is perfectly estimated. Thus, this baseline provides an upper bound on the performance of a real system, assuming that the reduced state captures all of the relevant information needed for the task. Unlike the Dynamics Oracle, this baseline does not assume access to the ground-truth system dynamics.

For all of the cloth related environments, the reduced state is the positions of the four corners of the cloth. For the StraightenRope environment, we pick 10 evenly-spaced keypoints on the rope, including the two end points, and use the positions of these key points as the reduced state representation. For TransportWater, the reduced state is the size (width, length, height) of the cup, the target cup position, and the initial height of the water in the cup. For PourWater, the reduced state is the sizes of both cups, the position of the target cup, the position and rotation of the controlled cup, and the initial height of the water in the cup. For any environment with pickers, the positions of the pickers are also included as part of the reduced state. We note that the set of reduced state we picked may not be sufficient for performing all of the manipulation tasks. For example, knowing the positions of the four corners of a crumpled cloth is not sufficient to infer the full configuration of the cloth, so some information is lost in this reduced state representation.

5.3.3 Image Based Observations

We also evaluate state-of-the-art RL algorithms that directly operate on high dimensional observations. It is important to evaluate methods that use high dimensional observations as input, since it cannot be assumed that a low dimensional state representation (such as that used by the Reduced State Oracle) can always be accurately inferred.

Recent works [55, 59, 60] show evidence that the gap between image-based RL and state-based RL can be closed on a range of tasks with the data augmentation in reinforcement learning. Among these, we benchmark CURL-SAC [60], which uses a model-free approach with a contrastive loss among randomly cropped images, and DrQ [55], which applies data augmentation and regularization to standard RL. We also evaluate PlaNet [41], which learns a latent state space dynamics model for planning. For SpreadCloth and FoldCloth, we additionally benchmark Wu et al. [121], which learns a pick-and-place policy with model-free RL.

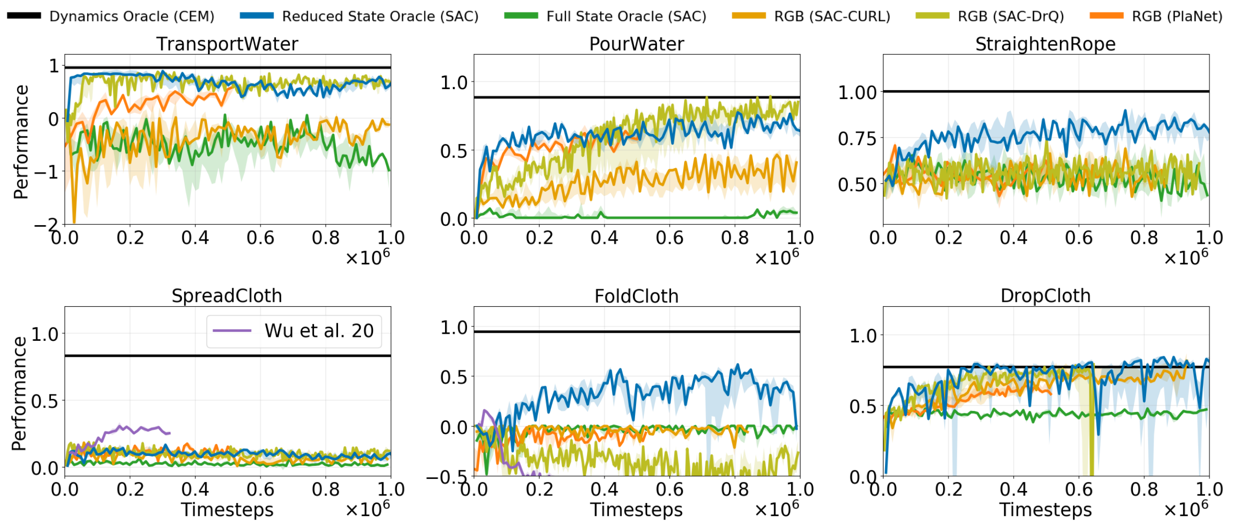


Figure 5.2: Normalized performance at the last time step of the episode of all the algorithms on the evaluation set. The x-axis shows the number of training time steps.

5.4 Experiments

In this section, we perform experiments with an aim to answer the following questions:

- Are SoftGym tasks challenging for current reinforcement learning algorithms?
- Is learning with state as sample efficient as learning from high-dimensional observations on SoftGym tasks?
- Are the environments realistic enough to reflect the difficulty of learning on a real deformable object dynamical system?

5.4.1 Experimental Setup

For each task, we compute a lower bound and upper bound on performance so that we can more easily analyze the performance of each method (see Appendix for details). The lower bound is obtained from a policy that always does nothing. Using these bounds, the performance of each method can then be normalized into $[0, 1]$, although the performance of a policy that performs worse than doing nothing can drop below 0. We run each algorithm for 5 random seeds and plot the median of the normalized performance. Any shaded area in the plots denotes the 25 and 75 percentile. In each task, we pre-sample 1000 variations of the environment. We then separate these task variations into a set of training tasks with 800 variations and a set of evaluation tasks of 200 variations. For CEM, no parameters are trained, so we modify this procedure: instead, we randomly sample 10 task variations from the evaluation set and compute the average performance across the variations. All methods are trained for 10^6 time steps, except PlaNet, which is trained for 5×10^5 time steps due to computation time. Please refer to the Appendix for more details of the algorithms and training procedure. Most of the experiments are run on an Nvidia 2080Ti GPU, with 4 virtual CPUs and 40G RAM.

5.4.2 Benchmarking Results on SoftGym-Medium

A summary of the final normalized performance of all baselines on the evaluation set is shown in Figure 5.2. As expected, the Dynamics Oracle performs the best and is able to solve most of the tasks. As the dynamics and ground-truth position of all particles are usually unknown in the real world, this method serves as an upper bound on the performance.

The Reduced State Oracle performs well on tasks where the reduced state captures the task relevant information, such as StraightenRope, TransportWater, FoldCloth, and performs poorly on the tasks which may require more information beyond the reduced state, such as in SpreadCloth, where the positions of the four cloth corners are not sufficient to reason about the configuration of the cloth. We note that the reduced states can be hard to obtain in the real world, such as in TransportWater, where the reduced state includes the amount of the water in the cup.

More interestingly, we also examine the performance of methods that assume that the agent only has access to image observations. A robot in the real world will not have access to ground-truth state information and must use these high dimensional observations as inputs to its policy. We observe that the performance of image-based reinforcement learning (PlaNet, SAC-CURL, or SAC-DrQ) is far below the optimal performance on many tasks. This is especially true for StraightenRope, SpreadCloth, and FoldCloth, and the learning curves for these tasks seem to imply that even with more training time, performance would still not improve. These methods also have a performance far below the upper bound of 1 on the other tasks (TransportWater, PourWater, DropCloth). Thus, this evaluation points to a clear need for new methods development for image-based robot manipulation of deformable objects. Compared to the reduced state oracle, image based methods have much worse performance in certain tasks such as FoldCloth or StraightenRope, indicating that there is still a gap between learning from high dimensional observation and learning from state representation.

The Full State Oracle, which uses the position of all particles in the object as input to a policy, performs poorly on all tasks. This further demonstrates the challenges for current RL methods in learning to manipulate deformable objects which have a variable size and high dimensional state representation.

For the SpreadCloth task, we additionally compare to previous work [121] that learns a model-free agent for spreading the cloth from image observation. A pick-and-place action space is used here. During training, for collecting data, a random location on the cloth is selected based on a segmentation map and the agent learns to select a place location. The picker will then move to the pick location above the cloth, pick up the cloth, move to the place location and then drop the cloth. In Figure 5.2, we show the final performance of this method with 20 pick-and-place steps for each episode. While it outperforms the rest of the baselines due to the use of the segmentation map and a better action space for exploration, the result shows that there still exists a large room for improvement. On the other hand, this method does not perform very well on the FoldCloth task.

5.4.3 Difficult Future Prediction

Why is learning with deformable objects challenging? Since PlaNet [41] learns an autoencoder and a dynamics model, we can visualize the future predicted observations of a trained PlaNet model. We input to the trained model the current frame and the planned action sequence and visualize the open-loop prediction of the future observation. Figure 5.3 shows that PlaNet fails to predict the spilled water or the shape of the cloth in deformable manipulation tasks. This provides evidence that since deformable objects have complex visual observations and dynamics, learning their dynamics is difficult.

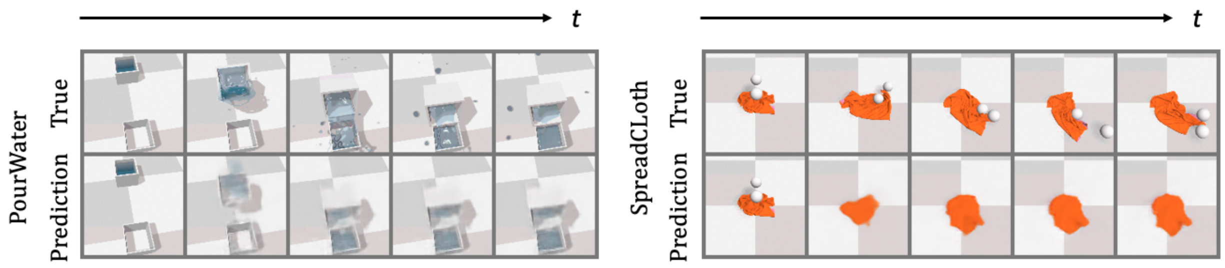


Figure 5.3: Bottom row: Open-loop prediction of PlaNet. Given an initial set of five frames, PlaNet predicts the following 30 frames. Here we show the last observed frame in the first column and four evenly spaced key frames out of the 30 predicted frames in the last four columns. Top row: Ground-truth future observations.

5.4.4 Reality Gap

Do SoftGym environments reflect the challenges in real world manipulation of deformable objects? Here we take the cloth environments as an example and show that both our cloth modeling and the picker abstraction can be transferred to the real world. We set up a real world cloth manipulation environment with a Sawyer robot with a Weiss gripper, as shown in Figure 5.4. We perform a series of pick and place actions both in simulation and on the real robot. We can see that the simulated cloth shows similar behaviour to the real one. This demonstration suggests that the simulation environment can reflect the complex dynamics in the real world and that algorithmic improvements of methods developed in SoftGym are likely to correspond to improvements for methods trained in the real world; however, direct sim2real transfer of learned policies is still expected to present a challenge.

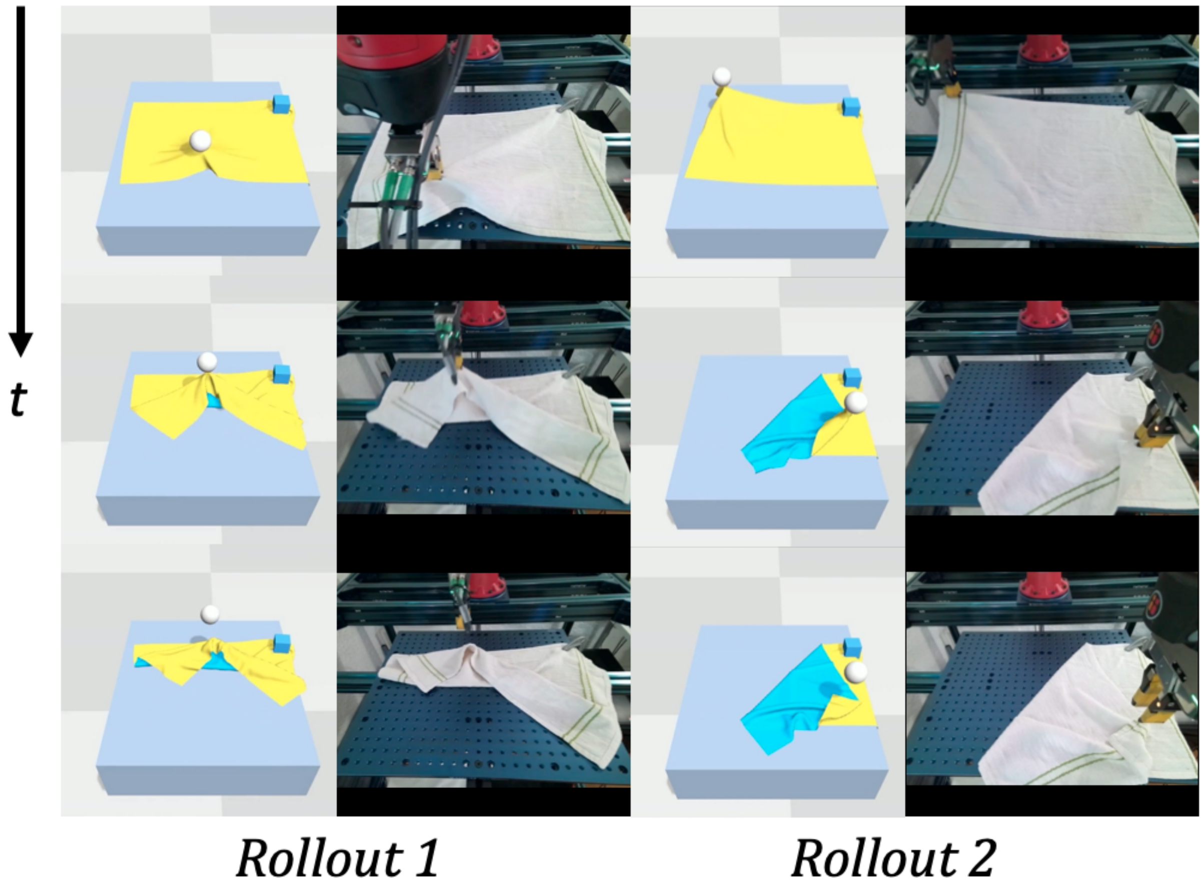


Figure 5.4: Two pick-and-place rollouts both in simulation and in the real world for a cloth manipulation task. For each rollout, the left column shows the simulation; the right shows the real world.

Chapter 6

Conclusion

In this thesis, we research how to perform object manipulation via deep reinforcement learning. Specifically, we studied the problem of inferring a reward function directly from images without human engineering, and benchmarking deep reinforcement learning for deformable object manipulation. Specifically, for learning a reward function from images, we propose ROLL, a goal-conditioned visual self-supervised reinforcement learning algorithm that incorporates object reasoning. We segment out the objects in the scene with unknown object segmentation to train an object-VAE; this provides a better reward function and goal sampling for self-supervised learning. We further employ an LSTM and a novel matching loss to make the method robust to object occlusions. Our experimental results show that ROLL significantly outperforms the baselines in terms of both final performance and learning speed. This showcases the importance of an object-centric view for learning robotic manipulation tasks. We hope our work can raise more research interest in exploring visual learning with object reasoning. In addition, we have proposed f-IRL, a practical IRL algorithm that distills an expert's state distribution into a stationary reward function. Our f-IRL algorithm can learn from either expert samples (as in traditional IRL), or a specified expert density (as in SMM [61]), which opens the door to supervising IRL with different types of data. These types of supervision can assist agents in solving tasks faster, encode preferences for how tasks are performed, and indicate which states are unsafe and should be avoided. Our experiments demonstrate that f-IRL is more sample efficient in the number of expert trajectories and environment timesteps as demonstrated on MuJoCo benchmarks. We also present SoftGym, a set of benchmark environments for deformable object manipulation. We show that manipulating deformable objects presents great challenges in learning from high dimensional observations and complex dynamics. We believe that our benchmark presents challenges to current reinforcement learning algorithms; further, our benchmark should help to ease the comparison of different approaches for deformable object manipulation, which should assist with algorithmic development in this area.

Appendix A

Appendix for ROLL

Proposition

A.1 Occlusion Analysis

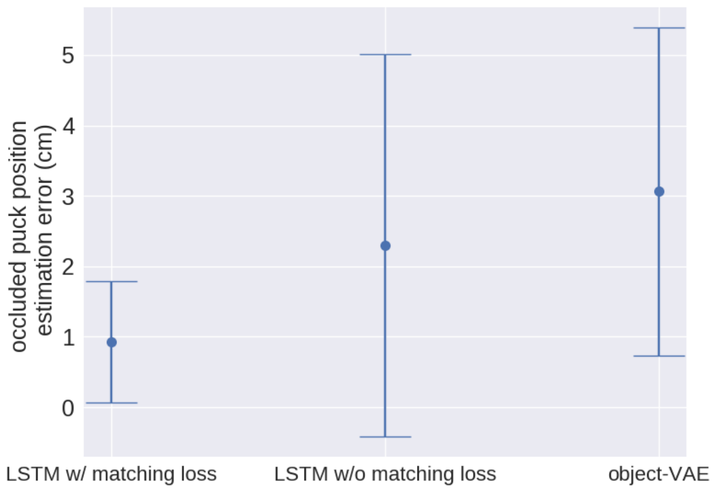
Here we perform more analysis on how the LSTM and matching loss enable ROLL to be robust to object occlusions. To analyze this, we generate a large number of puck moving trajectories in the Hurdle-Top Puck Pushing environment. Next, we train three different models on these trajectories:

- LSTM with matching loss
- LSTM without matching loss
- object-VAE (with no LSTM)

For each trajectory, we add synthetic occlusions to a randomly selected frame (by removing 85% of the pixels) and we use the three models to compute the latent encodings of the occluded frame (using the LSTM with the previous trajectory for models 1 and 2). We then use this embedding to retrieve the nearest neighbor frame in the collection of unoccluded trajectories whose latent embedding has the closest distance to the occluded latent embedding. This retrieved frame allows us to visualize the position that the model "thinks" the occluded puck is located at. For models 1 and 2, the model can use the LSTM and the previous trajectory to infer the location of the puck; for model 3, it cannot.

Finally, to evaluate this prediction, we compute the real puck distance between the location of the puck in the retrieved frame and the location of the puck in the occluded frame (using the simulator to obtain the true puck position, although this information is not available to the model). This distance can be interpreted as the estimation error of the puck position under occlusions. We report the mean and standard deviation of the estimation errors for three models in Supplementary Figure A.1(a). As shown, using the LSTM + matching loss achieves the lowest average estimation error of roughly just 1cm, while LSTM without matching loss has a larger error of 2.2 cm and object-VAE has the largest error of 3.1 cm.

We can also visualize these retrievals, as shown in Supplementary Figure A.2. We see that in all demonstrations, LSTM with matching loss almost perfectly retrieved the true unoccluded frames, while LSTM without matching loss and object-VAE retrieved incorrect frames with shifted puck positions. This shows that, even under severe occlusions (in the example, 85% of the pixels are dropped), with the LSTM and matching loss, ROLL can still correctly reason about the location of the object.



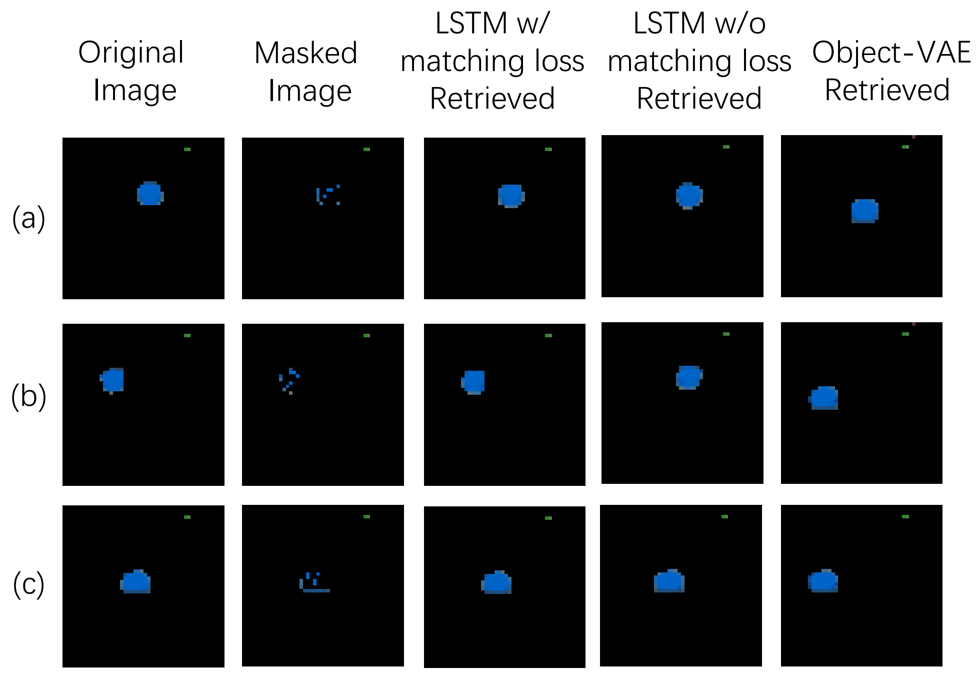
Supplementary Figure A.1: The estimation error of the puck position under occlusion of different methods.

A.2 More Policy Visualizations

Supplementary Figure A.3 shows more policy visualizations of ROLL and Skew-Fit. We can see that in most cases, ROLL achieves better manipulation results than Skew-Fit, aligning the object better with the target object position in the goal image. Skew-fit does not reason about objects and instead embeds the entire scene into a latent vector; further, Skew-fit does not reason about occlusions. Videos of the learned policies on all tasks can be found on the project website.

A.3 Sensitivity on Matching Loss Coefficient

We further test how sensitive ROLL is to the matching loss coefficient, on the Hurdle-Top Puck Pushing and the Hurdle-Bottom Puck Pushing task. The result is shown in Supplementary Figure A.4. From the results we can see that ROLL is only sensitive to the VAE matching loss coefficient when the task has large occlusions, i.e., in the Hurdle-Top Puck Pushing task. We also observe that in this task, the larger the VAE matching loss coefficient, the better the learning results. ROLL is more robust to the VAE matching loss coefficient in the Hurdle-Bottom Puck Pushing task, and we observe that larger VAE matching loss coefficients lead to slightly worse learning results. This is because the Hurdle-Bottom Puck Pushing task has a very small chance of object occlusions; thus too large of a VAE matching loss coefficient might instead slightly hurt the learned latent embedding. An intermediate VAE matching loss of 600 appears to perform



Original Image
Masked Image
LSTM w/ matching loss Retrieved
LSTM w/o matching loss Retrieved
Object-VAE Retrieved
(a)
(b)
(c)

Supplementary Figure A.2: Three demonstrations on what frames different models retrieve. We can see that LSTM with matching loss can accurately retrieve the true frames of the occluded puck, while LSTM without matching loss have small errors in the retrieved frames, and object-VAE has very large errors in the retrieved frames.

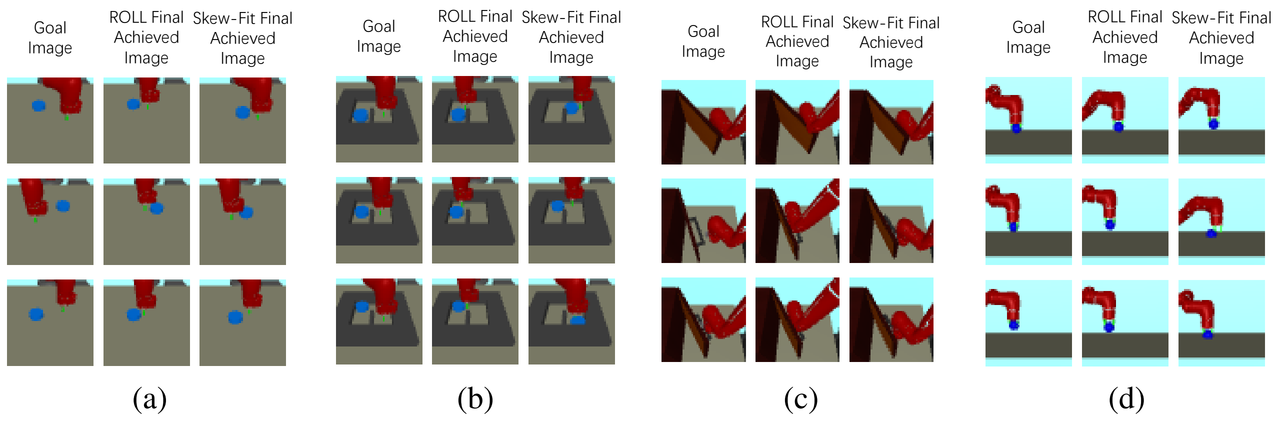
well for both tasks. Additionally, we see that ROLL is quite robust to the LSTM matching loss coefficient in both tasks.

A.4 Details on Unknown Object Segmentation

We now detail how we train the background subtraction module and the robot segmentation network in unknown object segmentation.

To obtain a background subtraction module, we cause the robot to perform random actions in an environment that is object free, and we record images during this movement. We then train a background subtraction module using the recorded images. Specifically, we use the Gaussian Mixture-based Background/Foreground Segmentation algorithm [134, 135] implemented in OpenCV [13]. In more detail, we use `BackgroundSubtractorMOG2` implemented in OpenCV. We record 2000 images of the robot randomly moving in the scene, set the tracking history of `BackgroundSubtractorMOG2` to 2000, and then train it on these images with an automatically chosen learning rate and variance threshold implemented by OpenCV. The background subtraction module is fixed after this training procedure.

The `BackgroundSubtractorMOG2` learns to classify non-moving objects in a scene as background, and any pixel values that fall outside a variance threshold of the Gaussian Mixture Model are classified as foreground. Illustrations of the learned background model are shown in



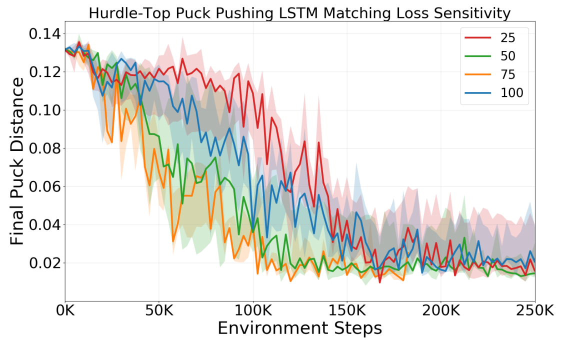
Supplementary Figure A.3: Policy visualizations on more tasks. (a) Puck Pushing. (b) Hurdle-Bottom Puck Pushing. (c) Door Opening. (d) Object Pickup.

Supplementary Figure A.5. At test time, objects placed in the environment appear as foreground as their pixel values fall outside the threshold. Similarly, the robot also appears as foreground, for the same reason. We address this issue using a robot segmentation network, explained in detail below.

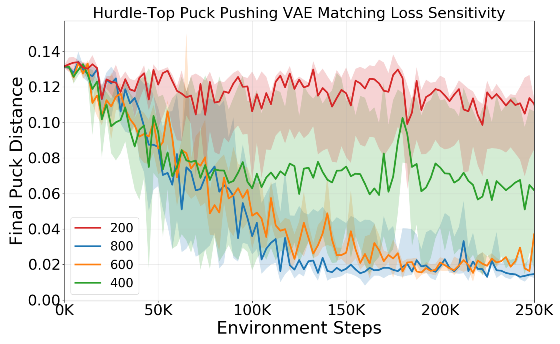
In order to remove the robot from the scene, we train a robot segmentation network. To generate training labels, we use the trained background subtraction module, described above. Using the same dataset used to train the OpenCV background subtraction module (with no objects in the scene), we run the OpenCV background subtraction module. Points that are classified as foreground belong to the robot. We use this output as “ground-truth” segmentation labels. Using these labels, we train a network to segment the robot from the background. We use U-Net [95] as the segmentation network. The U-Net model we use has 4 blocks of down-sampling convolutions and then 4 blocks of up-sampling covolutions. Every block has a max-pool layer, two convolutional layers each followed by a batch normalization layer and a ReLU activation. Each up-sampling layer has input channels concatenated from the outputs of its down-sampling counterpart. These additional features concatenated from the input convolutions help propagate context information to the higher resolution up-sampling layers. The kernel size is 3x3, with stride 1 and padding 1 for all the convolutional layers.

We train the network using a binary cross entropy loss. The optimization is performed using Nesterov momentum gradient descent for 30 epochs with a learning rate of 1e-3, momentum of 0.9, and a weight decay of 5e-4.

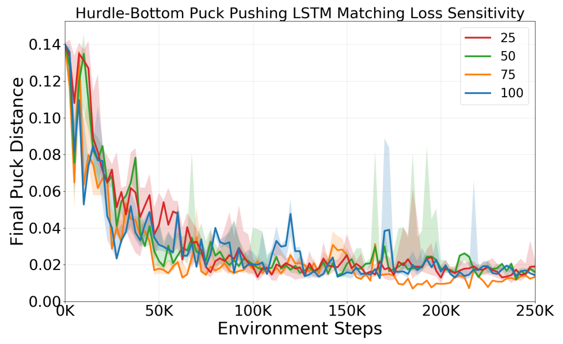
One potential issue of the above method is that the robot segmentation module has only been trained on images without objects in the scene. We find that adding synthetic distractors to the scene helps to improve performance. In this work, we use distractors created by masks of objects similar to those at test time. In future work, we will instead use diverse distractors taken from the COCO dataset [67].



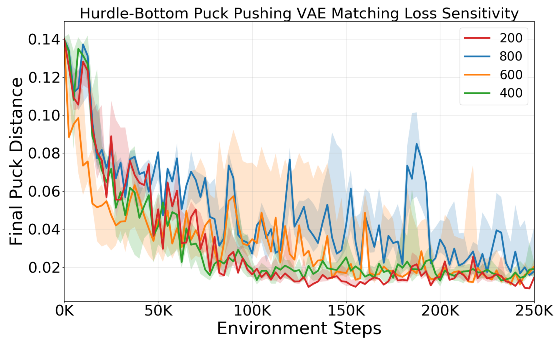
(a)



(b)



(c)



(d)

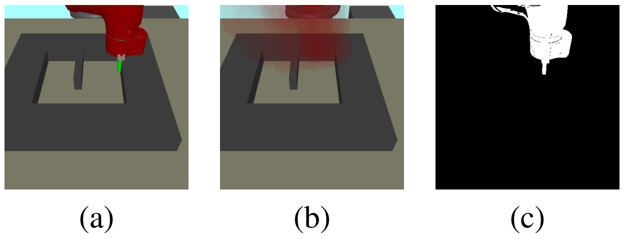
Supplementary Figure A.4: Sensitivity of ROLL to the LSTM / VAE matching loss coefficient on the Hurdle-Top Puck Pushing and Hurdle-Bottom Puck Pushing tasks. (a) & (b): results on the Hurdle-Top Puck Pushing task. (c) & (d): results on the Hurlde-Bottom Puck Pushing task. (a) & (c): test the sensitivity to the VAE matching loss coefficient. (b) & (d): test the sensitivity to the LSTM matching loss coefficient. For (a), the VAE matching loss coefficient is fixed at 800; For (b), the LSTM matching loss coefficient is fixed at 50. For (c), the VAE matching loss coefficient is fixed at 400. For (d), the LSTM matching loss coefficient is fixed at 25.

A.5 Simulated Task Details

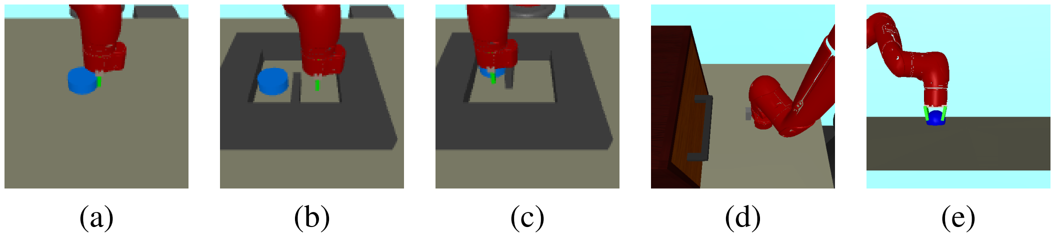
All the tasks are simulated using the MuJoCo [111] physics engine. The Puck Pushing, Door Opening, and Object Pickup tasks are identical to those used in Skew-Fit [91]. Illustrations of the environments are shown in Supplementary Figure A.6(a), (d), and (e). We also added two additional environments with obstacles and challenging occlusions, shown in Supplementary Figure A.6(b) and (c).

For the coordinates used in the puck pushing tasks, the x-axis goes towards the right direction, and the y-axis goes towards the bottom direction in Supplementary Figure A.6.

Puck Pushing: A 7-DoF Sawyer arm must push a small puck on a table to various target positions. The agent controls the arm by commanding the $\delta x, \delta y$ movement of the end effector (EE). The underlying state is the EE position e and the puck position p. The evaluation metric is the distance between the goal and final achieved puck positions. The hand goal/state space is a box $[-0.1, 0.1] \times [0.55, 0.65]$. The puck goal/state space is a box $[-0.15, 0.15] \times [0.5, 0.7]$. The action space ranges in the interval $[-1, 1]$ in the x, y dimensions. The arm is always reset to



Supplementary Figure A.5: (a) RGB input (b) RGB background model learned by `BackgroundSubtractorMOG2` (c) Predicted foreground on the Hurdle-Top Puck Pushing task.



Supplementary Figure A.6: The robot view of different tasks: (a) Puck Pushing (b) Hurdle-Bottom Puck Pushing (c) Hurdle-Top Puck Pushing (d) Door Opening (e) Object Pickup.

$(-0.02, 0.5)$ and the puck is always reset to $(0, 0.6)$.

Hurdle-Bottom Puck Pushing: The task is similar to that of Puck Pushing, except we add hurdles on the table to restrict the movement of the puck. The coordinates of the inner corners of the hurdle are top-left $(0.11, 0.52)$, top-right $(-0.025, 0.52)$, bottom-left $(0.11, 0.67)$, bottom-right $(-0.025, 0.67)$. The left corridor to the middle-hurdle has a width of 0.06, and the right corridor to the middle-hurdle has a width of 0.065. The corridor up to the hurdle has a width of 0.07. The middle-hurdle has a width of 0.01 and a length of 0.08. The arm is always reset to the location of $(-0.02, 0.5)$ and the puck is reset to $(-0.02, 0.54)$. The puck goal space is $[0.1, 0.11] \times [0.55, 0.65]$ (i.e., roughly the range of the left corridor). The hand goal space is $[-0.025, 0] \times [0.6, 0.65]$ (i.e., roughly the bottom part of the right corridor).

Hurdle-Top Puck Pushing: The task is similar to that of Hurdle-Bottom Puck Pushing, except the position of the middle-hurdle is flipped. The arm is always reset to $(-0.02, 0.5)$ and the puck is randomly reset to be in $[-0.04, -0.02] \times [0.55, 0.63]$ (i.e., roughly the top part of the right corridor). The puck goal space is $[0.1, 0.11] \times [0.55, 0.6]$ (i.e., roughly the top part of the left corridor), and the hand goal space is $[-0.03, 0] \times [0.54, 0.6]$ (i.e., roughly the top part of the right corridor).

Door Opening: A 7-DoF Sawyer arm must pull a door on a table to various target angles. The agent control is the same as in Puck Pushing, i.e., the $\delta x, \delta y$ movement of the end effector. The evaluation metric is the distance between the goal and final door angle, measured in radians. In this environment, we do not reset the position of the hand or door at the end of each trajectory. The state/goal space is a $5 \times 20 \times 15$ cm^3 box in the x, y, z dimension respectively for the arm and an angle between $[0, .83]$ radians for the door. The action space ranges in the interval $[-1, 1]$ in the x, y and z dimensions.

Object Pickup: A 7-DoF Sawyer arm must pick up an object on a table to various target positions. The object is cube-shaped, but a larger intangible sphere is overlaid on top so that it is easier for the agent to see. Moreover, the robot is constrained to move in 2 dimension: it only controls the y, z arm positions. The x position of both the arm and the object is fixed. The evaluation metric is the distance between the goal and final object position. For the purpose of evaluation, 75% of the goals have the object in the air and 25% have the object on the ground. The state/goal space for both the object and the arm is 10cm in the y dimension and 13cm in the z dimension. The action space ranges in the interval $[-1, 1]$ in the y and z dimensions.

A.6 Implementation Details

Our implementation of ROLL is based on the open-source implementation of Skew-Fit in RLkit[1]. For all simulated tasks, the image size is 48×48. A summary of the task specific hyper-parameters of ROLL are shown in Supplementary Table A.1. The first 4 rows use the same hyper-parameters as in Skew-Fit [91], and the last two rows are hyper-parameters that we tune in ROLL, described in more detail below. A summary of the task-independent hyper-parameters of ROLL for training the scene-VAE, the object-VAE and the LSTM are shown in Supplementary Table A.2, with detailed descriptions in the text below.

A.6.1 Network Architectures

We first describe the network architecture of each component in ROLL. For the scene-VAE, we use the same architecture as that in Skew-Fit. In more detail, the VAE encoder has three convolutional layers with kernel sizes: 5×5, 3×3, and 3×3, number of output filters: 16, 32, and 64; and strides: 3, 2, and 2. The final feature map is mapped by a fully connected layer into a final feature vector of size 576, and then we have another fully connected layer to output the final latent embedding. The decoder has a fully connected layer that maps the latent embedding into a vector of dimension 576. This vector is then reshaped into a feature map of size $3 \times 3 \times 64$. The decoder has 3 de-convolution layers with kernel sizes 3×3, 3×3, 6×6, number of output filters 32, 16, and 3, and strides 2, 2, and 3.

The object-VAE has almost the same architecture as the scene-VAE. However, the object-VAE has a simpler task that it only needs to encode the segmented object, rather than the entire scene. Thus, we use a smaller final feature vector – the final encoder feature vector is of size 6 instead of 576 as used in the scene-VAE. Both VAEs have a Gaussian decoder with identity variance; thus the log likelihood loss used to train the decoder is equivalent to a mean-squared error loss.

For the scenc-VAE, we vary the value of β as in Skew-Fit for different tasks (shown in Supplementary Table A.1). For the object-VAE, we use the same value of 20 for β, and the same latent dimension size of 6 in all tasks, as shown in Supplementary Table A.2. For the scene-VAE in Skew-Fit, we use a latent dimension size of 16 for the Door Opening task and the Object Pickup task, and use a latent dimension size of 4 for Puck Pushing tasks (which is the same

[1]https://github.com/vitchyr/rlkit

Hyper-parameter	**Puck Pushing**	**Hurdle-Bottom Puck Pushing**	**Hurdle-Top Puck Pushing**	**Door Opening**	**Object Pickup**
Trajectory Length	50	50	50	100	50
β for scene-VAE	20	20	20	20	30
Scene-VAE Latent Dimension Size in Skew-Fit	4	6	6	16	16
Skew-Fit α for scene-VAE	-1	-1	-1	-0.5	-1
Scene-VAE Latent Dimension Size in ROLL	6	6	6	16	16
VAE matching loss coefficient	400	400	800	50	50

Supplementary Table A.1: Task specific hyper-parameters. The first four rows use the same hyper-parameters as in Skew-Fit. The fifth row shows that for ROLL, we increase the scene-VAE latent dimension size in Puck Pushing from 4 to 6, as to keep it the same as the object-VAE latent size that we use, which is 6 for all tasks. We also observe that using a latent dimension of 6 for the scene-VAE performs better than using a latent dimension of 4 in ROLL for this task. However, using a latent dimension of 6 for the scene-VAE in Skew-Fit performs slightly worse than using a latent dimension of 4 in this task. The last row is one new hyper-parameter that we introduce in ROLL.

Hyper-parameter	**Value**
Scene-VAE Batch Size (for both Skew-Fit and ROLL)	64
Object-VAE Batch Size	128
β for Object-VAE	20
Obect-VAE Latent Dimension Size	6
LSTM Matching Loss Coefficient	50

Supplementary Table A.2: Scene-VAE, object-VAE and LSTM training hyper-parameters for all tasks.

as in the original Skew-Fit implementation). For the Puck Pushing with hurdle tasks, we use a latent dimension of size 6. For ROLL, the scene-VAE latent dimension size is the same as that of Skew-Fit except for the Puck Pushing task, where we increase the latent dimension from 4 to 6 to make it the same as the object-VAE latent dimension. For this task, we also observe that using a latent dimension of 6 for the scene-VAE performs better than a latent dimension of 4 in ROLL. However, using a latent dimension of 6 for the scene-VAE in Skew-Fit performs slightly worse than using a latent dimension of 4 in this task.

The input to the LSTM is the latent vector from the object-VAE. The LSTM for all tasks has 2 layers and a hidden size of 128 units.

For the policy and Q-network used in SAC, we use exactly the same architecture as in Skew-Fit. For both networks, we use fully connected networks with two hidden layers of size 400 and 300 each, and use ReLU as the activation function.

A.6.2 Training schedules

We train the scene-VAE using the regular β-VAE loss, i.e., the image reconstruction loss and the KL regularization loss. We pre-train it using 2000 images obtained by running a random policy for 2000 epochs. In each epoch we train for 25 batches with a batch size of 64 and a learning rate of $1e-3$. We also continue to train the scene-VAE alongside during RL training, using images stored in the replay buffer. We sample images from the replay buffer using a skewed distribution as implemented in Skew-Fit. For different tasks we use different skewness α as shown in Supplementary Table A.1, which is the same as in Skew-Fit. For online training of the scene-VAE, Skew-Fit use three different training schedules for different tasks, and we follow the same training schedule as in Skew-Fit. For details on the training schedule, please refer to appendix C.5 of the Skew-Fit paper.

We train the object-VAE using the image reconstruction loss, the KL regularization loss, and the matching loss. For different tasks, we use different coefficients for the VAE-matching loss, as shown in Supplementary Table A.1. We pre-train the object-VAE with 2000 segmented images obtained by randomly putting the object in the scene. The object-VAE is trained for 2000 epochs, where in each epoch we train for 25 batches with a batch size of 128. We use a learning rate of $1e-3$. After the pre-training, the object-VAE is fixed during RL learning. For the synthetic occlusions we add for computing the matching loss, we randomly drop 50% pixels in the segmented objects.

We train the LSTM using an auto-encoder loss and matching loss. For all tasks, we use the same LSTM-matching loss coefficient of 50, as shown in Supplementary Table A.2. We pre-train the LSTM on the same dataset we use to pre-train the object-VAE, using the auto-encoder loss for 2000 epochs. In each epoch we train for 25 batches with a batch size of 128 and a learning rate of $1e-3$. We continue training the LSTM during the RL learning process. During online training, the training trajectories are sampled uniformly from the SAC replay buffer, and we use both the matching loss and the auto-encoder loss to train the LSTM. We use a learning rate of $1e-3$ for training the LSTM. The online training scheme for LSTM is: for the first 5k time steps, we train the LSTM every 500 time steps for 80 batches, where each batch has 25 trajectories. For 5k - 50k time steps, we train the LSTM every 500 time steps for 20 batches. After 50k time steps, we train the LSTM every 1000 time steps for 20 batches. For the SAC training schedule, we use the default values as in Skew-Fit; these values are summarized in Supplementary Table A.3.

A.7 Generalization to Real-world Robot

Due to disruptions to lab access caused by the COVID-19 pandemic, we were not able to validate our method on real robots. However, we believe our proposed method can work in the real-world for the following reasons:

ROLL applies simple modifications upon prior work which has been demonstrated to work in the real world. Skew-fit, upon which ROLL is based, has been demonstrated on robots in the real-world, for both door opening [91] and puck pushing [81]. ROLL applies the following modifications upon Skew-Fit, which should not affect the ability of ROLL to work in the real

Hyper-parameter	Value
# training batches per time step	2
RL Batch Size	1024
Discount Factor	0.99
Reward Scaling	1
Replay Buffer Size	100000
Soft Target τ	$1e-3$
Target Update Period	1
Use Automatic α tuning	True
Policy Learning Rate	$1e-3$
Q-function Learning Rate	$1e-3$

Supplementary Table A.3: SAC training hyper-parameters for all tasks; these are the same values as used in previous work [91].

world:

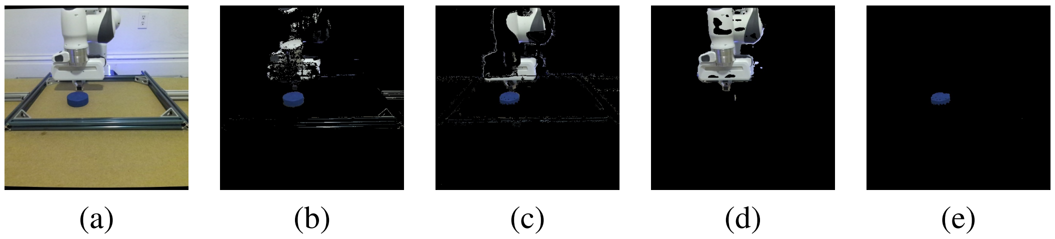
ROLL uses an object-VAE and an LSTM for reward computation. To train the object-VAE, ROLL requires segmented images. As described in section 4.2, the process of unknown object segmentation is fairly simple (see below for a demonstration of our method in the real world). Alternative methods can also be used for segmentation, such as [125].

Additionally, ROLL needs to pre-collect two datasets for training the scene- and object-VAE. The data collection for the scene-VAE is the same as in prior work [81]. It is also easy to collect the dataset for pre-training the object-VAE: we just need to manually place the target objects at different locations in the scene. We can further reduce the number of manual placements of the object by using data augmentation such as translation, rotation and cropping to create more training samples.

ROLL is more sample-efficient than prior work which has been demonstrated to work in the real world. As shown in Figure 4 in the paper, ROLL is much more sample-efficient than Skew-Fit in 4 out of the 5 simulated tasks (and similar efficiency in the 5th). Sample efficiency is often a major bottleneck for robot learning in the real world. As Skew-Fit works on real robots and ROLL is much more sample-efficient, we believe that it will be even easier to apply ROLL to real-world robots.

ROLL consistently works well on 5 different simulated tasks. We have tested ROLL in 5 different simulation tasks (using the well-known MuJoCo [111] physics engine), which involves manipulating different objects (pushing a puck, opening a door, and picking up a ball), and with different environments (puck pushing without hurdles and with different kinds of hurdles). The diversity of these tasks provides encouragement that ROLL will work well on similar tasks in the real world (as also demonstrated by the real-world experiments in the previous work that we build on [81, 91]).

Unknown object segmentation works in the real world. To enable our background subtraction method to work robustly in the real wold under shadows and reflections, we train both an RGB and depth-based background subtraction module using `BackgroundSubtractorMOG2` in OpenCV (Supp Fig. A.7b,c). Similar to our approach in simulation, we train a segmentation network to remove the robot from the foreground (Supp Fig. A.7d). After removing the background and the robot, what remains is a segmentation of the object (Supp Fig. A.7e).



Supplementary Figure A.7: Segmentation pipeline in the real world: (a) RGB Input (b) RGB background subtraction (c) Depth background subtraction (d) Predicted robot mask (e) Object segmentation.

Appendix B

Appendix for f-IRL

B.1 Derivation and Proof

This section provides the derivation and proof for the main paper. Section B.1.1 and B.1.2 provide the derivation of Theorem 4.2.1, and section B.1.4 provides the details about section 4.2.3.

B.1.1 Analytical Gradient of State Marginal Distribution

In this subsection, we start by deriving a general result - gradient of state marginal distribution w.r.t. parameters of the reward function. We will use this gradient in the next subsection B.1.2 where we derive the gradient of f-divergence objective.

Based on the notation introduced in section 4.1, we start by writing the probability of trajectory $\tau = (s_0, s_1, \dots, s_T)$ of fixed horizon T under the optimal MaxEnt trajectory distribution for $r_\theta(s)$ [132].

$$\rho_\theta(\tau) \propto \rho_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \tag{B.1}$$

Let $p(\tau) = \rho_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)$, which is the probability of the trajectory under the dynamics of the environment.

Explicitly computing the normalizing factor, we can write the distribution over trajectories as follows:

$$\rho_\theta(\tau) = \frac{p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha}}{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} d\tau} \tag{B.2}$$

Let $\eta_\tau(s)$ denote the number of times a state occurs in a trajectory τ. We now compute the marginal distribution of all states in the trajectory:

$$\rho_\theta(s) \propto \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau \tag{B.3}$$

where

$$\eta_\tau(s) = \sum_{t=1}^{T} 1(s_t = s) \tag{B.4}$$

is the empirical frequency of state s in trajectory τ (omitting the starting state s_0 as the policy cannot control the initial state distribution).

The marginal distribution over states can now be written as:

$$\rho_\theta(s) \propto \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau \tag{B.5}$$

In the following derivation, we will use s_t to denote states in trajectory τ and s'_t to denote states from trajectory τ'. Explicitly computing the normalizing factor, the marginal distribution can be written as follows:

$$\begin{aligned}\rho_\theta(s) &= \frac{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau}{\int \int p(\tau') e^{\sum_{t=1}^{T} r_\theta(s'_t)/\alpha} \eta_{\tau'}(s') d\tau' ds'} \\ &= \frac{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau}{\int p(\tau') e^{\sum_{t=1}^{T} r_\theta(s'_t)/\alpha} \int \eta_{\tau'}(s') ds' d\tau'} \\ &= \frac{\int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau}{T \int p(\tau') e^{\sum_{t=1}^{T} r_\theta(s'_t)/\alpha} d\tau'}\end{aligned} \tag{B.6}$$

In the second step we swap the order of integration in the denominator. The last line follows because only the T states in τ satisfy $s \in \tau$. Finally, we define $f(s)$ and Z to denote the numerator (dependent on s) and denominator (normalizing constant), to simplify notation in further calculations.

$$\begin{aligned}f(s) &= \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) d\tau \\ Z &= T \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} d\tau \\ \rho_\theta(s) &= \frac{f(s)}{Z}\end{aligned} \tag{B.7}$$

As an initial step, we compute the derivatives of $f(s)$ and Z w.r.t reward function at some state $r_\theta(s^*)$.

$$\frac{df(s)}{dr_\theta(s^*)} = \frac{1}{\alpha} \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s) \eta_\tau(s^*) d\tau \tag{B.8}$$

$$\frac{dZ}{dr_\theta(s^*)} = \frac{T}{\alpha} \int p(\tau) e^{\sum_{t=1}^{T} r_\theta(s_t)/\alpha} \eta_\tau(s^*) d\tau = \frac{T}{\alpha} f(s^*) \tag{B.9}$$

We can then apply the quotient rule to compute the derivative of policy marginal distribution w.r.t. the reward function.

$$\begin{aligned}\frac{d\rho_\theta(s)}{dr_\theta(s^*)} &= \frac{Z\frac{df(s)}{dr_\theta(s^*)} - f(s)\frac{dZ}{dr_\theta(s^*)}}{Z^2} \\ &= \frac{\int p(\tau)e^{\sum_{t=1}^T r_\theta(s_t)/\alpha}\eta_\tau(s)\eta_\tau(s^*)d\tau}{\alpha Z} - \frac{f(s)}{Z}\frac{Tf(s^*)}{\alpha Z} \\ &= \frac{\int p(\tau)e^{\sum_{t=1}^T r_\theta(s_t)/\alpha}\eta_\tau(s)\eta_\tau(s^*)d\tau}{\alpha Z} - \frac{T}{\alpha}\rho_\theta(s)\rho_\theta(s^*)\end{aligned} \tag{B.10}$$

Now we have all the tools needed to get the derivative of ρ_θ w.r.t. θ by the chain rule.

$$\begin{aligned}\frac{d\rho_\theta(s)}{d\theta} &= \int \frac{d\rho_\theta(s)}{dr_\theta(s^*)}\frac{dr_\theta(s^*)}{d\theta}ds^* \\ &= \frac{1}{\alpha}\int \left(\frac{\int p(\tau)e^{\sum_{t=1}^T r_\theta(s_t)/\alpha}\eta_\tau(s)\eta_\tau(s^*)d\tau}{Z} - T\rho_\theta(s)\rho_\theta(s^*)\right)\frac{dr_\theta(s^*)}{d\theta}ds^* \\ &= \frac{1}{\alpha Z}\int\int p(\tau)e^{\sum_{t=1}^T r_\theta(s_t)/\alpha}\eta_\tau(s)\eta_\tau(s^*)\frac{dr_\theta(s^*)}{d\theta}ds^* d\tau - \frac{T}{\alpha}\rho_\theta(s)\int \rho_\theta(s^*)\frac{dr_\theta(s^*)}{d\theta}ds^* \\ &= \frac{1}{\alpha Z}\int p(\tau)e^{\sum_{t=1}^T r_\theta(s_t)/\alpha}\eta_\tau(s)\sum_{t=1}^T\frac{dr_\theta(s_t)}{d\theta}d\tau - \frac{T}{\alpha}\rho_\theta(s)\int \rho_\theta(s^*)\frac{dr_\theta(s^*)}{d\theta}ds^*\end{aligned} \tag{B.11}$$

B.1.2 Analytical Gradient of f-divergence objective

f-divergence [4] is a family of divergence, which generalizes forward/reverse KL divergence. Formally, let P and Q be two probability distributions over a space Ω, then for a convex and lower-semicontinuous function f such that $f(1) = 0$, the f-divergence of P from Q is defined as:

$$D_f(P \,||\, Q) := \int_\Omega f\left(\frac{dP}{dQ}\right)dQ \tag{B.12}$$

Applied to state marginal matching between expert density $\rho_E(s)$ and agent density $\rho_\theta(s)$ over state space $\mathcal{S}$, the f-divergence objective is:

$$\min_\theta L_f(\theta) := D_f(\rho_E \,||\, \rho_\theta) = \int_\mathcal{S} f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right)\rho_\theta(s)ds \tag{B.13}$$

Now we show the proof of **Theorem 4.2.1** on the gradient of f-divergence objective:

Proof. The gradient of the f-divergence objective can be derived by chain rule:

$$
\begin{aligned}
\nabla_\theta L_f(\theta) &= \int \nabla_\theta \left(f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \rho_\theta(s) \right) ds \\
&= \int \left(f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) - f'\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right)\frac{\rho_E(s)}{\rho_\theta(s)} \right) \frac{d\rho_\theta(s)}{d\theta} ds \\
&\triangleq \int h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \frac{d\rho_\theta(s)}{d\theta} ds
\end{aligned}
\tag{B.14}
$$

where we denote $h_f(u) \triangleq f(u) - f'(u)u.$ for convenience.[1]

Substituting the gradient of state marginal distribution w.r.t θ in Eq. B.11, we have:

$$
\begin{aligned}
&\nabla_\theta L_f(\theta) \\
&= \int h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \left(\frac{1}{\alpha Z} \int p(\tau) e^{\sum_{t=1}^T r_\theta(s_t)/\alpha} \eta_\tau(s) \sum_{t=1}^T \frac{dr_\theta(s_t)}{d\theta} d\tau - \frac{T}{\alpha} \rho_\theta(s) \int \rho_\theta(s^*) \frac{dr_\theta(s^*)}{d\theta} ds^* \right) ds \\
&= \frac{1}{\alpha Z} \int p(\tau) e^{\sum_{t=1}^T r_\theta(s_t)/\alpha} \sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right) \sum_{t=1}^T \frac{dr_\theta(s_t)}{d\theta} d\tau \\
&\quad - \frac{T}{\alpha} \int h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \rho_\theta(s) \left(\int \rho_\theta(s^*) \frac{dr_\theta(s^*)}{d\theta} ds^* \right) ds \\
&= \frac{1}{\alpha T} \int \rho_\theta(\tau) \sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right) \sum_{t=1}^T \frac{dr_\theta(s_t)}{d\theta} d\tau \\
&\quad - \frac{T}{\alpha} \left(\int h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \rho_\theta(s) ds \right) \left(\int \rho_\theta(s^*) \frac{dr_\theta(s^*)}{d\theta} ds^* \right) \\
&= \frac{1}{\alpha T} \mathbb{E}_{\tau \sim \rho_\theta(\tau)} \left[\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right) \sum_{t=1}^T \frac{dr_\theta(s_t)}{d\theta} \right] - \frac{T}{\alpha} \mathbb{E}_{s \sim \rho_\theta(s)} \left[h_f\left(\frac{\rho_E(s)}{\rho_\theta(s)}\right) \right] \mathbb{E}_{s \sim \rho_\theta(s)} \left[\frac{dr_\theta(s)}{d\theta} \right]
\end{aligned}
\tag{B.15}
$$

To gain more intuition about this equation, we can convert all the expectations to be over the trajectories:

$$
\begin{aligned}
&\nabla_\theta L_f(\theta) \\
&= \frac{1}{\alpha T} \left(\mathbb{E}_{\rho_\theta(\tau)} \left[\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right) \sum_{t=1}^T \nabla_\theta r_\theta(s_t) \right] - \mathbb{E}_{\rho_\theta(\tau)} \left[\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right) \right] \mathbb{E}_{\rho_\theta(\tau)} \left[\sum_{t=1}^T \nabla_\theta r_\theta(s_t) \right] \right) \\
&= \frac{1}{\alpha T} \mathrm{cov}_{\tau \sim \rho_\theta(\tau)} \left(\sum_{t=1}^T h_f\left(\frac{\rho_E(s_t)}{\rho_\theta(s_t)}\right), \sum_{t=1}^T \nabla_\theta r_\theta(s_t) \right)
\end{aligned}
\tag{B.16}
$$

Thus we have derived the analytic gradient of f-divergence for state-marginal matching as shown in Theorem 4.2.1. $\square$

[1]Note that if $f(u)$ is non-differentiable at some points, such as $f(u) = |u - 1|/2$ at $u = 1$ for Total Variation distance, we take one of its subderivatives.

B.1.3 Extension to Integral Probability Metrics in f-IRL

Integral Probability Metrics (IPM) [78] is another class of divergence based on dual norm, examples of which include Wasserstein distance [7] and MMD [64]. We can use Kantorovich-Rubinstein duality [115] to rewrite the IPM-based state marginal matching as:

$$L_{\mathrm{B}}(\theta) = \|\rho_E(s) - \rho_\theta(s)\|_B := \max_{D_\omega \in B} \mathbb{E}_{\rho_E(s)}[D_\omega(s)] - \mathbb{E}_{\rho_\theta(s)}[D_\omega(s)] \tag{B.17}$$

where B is a symmetric convex set of functions and D_ω is the critic function in [7].

Then the analytical gradient of the objective $L_{\mathrm{B}}(\theta)$ can be derived to be:

$$\nabla_\theta L_{\mathrm{B}}(\theta) = -\frac{1}{\alpha T} \mathrm{cov}_{\tau\sim\rho_\theta(\tau)} \left(\sum_{t=1}^{T} D_\omega(s_t), \sum_{t=1}^{T} \nabla_\theta r_\theta(s_t) \right) \tag{B.18}$$

where the derivation directly follows the proof of Theorem 4.2.1.

B.1.4 f-IRL Learns Disentangled Rewards w.r.t. Dynamics

We follow the derivation and definitions as given in Fu et al. [32] to show that f-IRL learns disentangled rewards. We show the definitions and theorem here for completeness. For more information, please refer to Fu et al. [32].

We first redefine the notion of "disentangled rewards".

Definition 1 (Disentangled rewards)**.** *A reward function $r'(s,a,s')$ is (perfectly) disentangled with respect to ground truth reward $r_{gt}(s,a,s')$ and a set of dynamics $\mathcal{T}$ such that under all dynamics in $T \in \mathcal{T}$, the optimal policy is the same: $\pi^*_{r',T}(a|s) = \pi^*_{r_{gt},T}(a|s)$*

Disentangled rewards can be loosely understood as learning a reward function which will produce the same optimal policy as the ground truth reward for the environment, on any underlying dynamics.

To show how f-IRL recovers a disentangled reward function, we need go through the definition of "Decomposability condition"

Definition 2 (Decomposability condition)**.** *Two states s_1,s_2 are defined as "1-step linked" under a dyanamics or transition distribution $T(s'|a,s)$, if there exists a state that can reach s_1 and s_2 with positive probability in one timestep. Also, this relationship can transfer through transitivity: if s_1 and s_2 are linked, and s_2 and s_3 are linked then we can consider s_1 and s_3 to be linked.*
A transition distribution T satisfies the decomposibility condition if all states in the MDP are linked with all other states.

This condition is mild and can be satisfied by any of the environments used in our experiments.

Theorem B.1.1 and B.1.2 formalize the claim that f-IRL recovers disentangled reward functions with respect to the dynamics. The notation $Q^*_{r,T}$ denotes the optimal Q function under reward function r and dynamics T, and similarly $\pi^*_{r,T}$ is the optimal policy under reward function r and dynamics T.

Theorem B.1.1. *Let $r_{gt}(s)$ be the expert reward, and T be a dynamics satisfying the decomposability condition as defined in [32]. Suppose f-IRL learns a reward r_{IRL} such that it produces an* *optimal policy in T: $Q^*_{r_{IRL},T}(s,a) = Q^*_{r_{gt},T}(s,a) - f(s)$,where $f(s)$ is an arbitrary function of the state. Then we have:*

$r_{IRL}(s) = r_{gt}(s) + C$ for some constant C, and thus $r_{IRL}(s)$ is robust to all dynamics.

Proof. Refer to Theorem 5.1 of AIRL [32]. □

Theorem B.1.2. *If a reward function $r'(s,a,s')$ is disentangled with respect to all dynamics functions, then it must be state-only.*

Proof. Refer to Theorem 5.2 of AIRL [32]. □

B.2 What Objective is Optimized by Previous IL Algorithms?

In this section, we discuss previous IL methods and analyze which objectives they may truly optimize. Our analysis shows that AIRL and GAN-GCL methods possibly optimize for a different objective than they claim, due to their usage of biased importance sampling weights.

B.2.1 MaxEntIRL [133], Deep MaxEntIRL [122], GCL [30]

Classical IRL methods [83, 98] obtain a policy by learning a reward function from sampled trajectories of an expert policy. MaxEntIRL [133] learns a stationary reward by maximizing likelihood on expert trajectories, i.e., it minimizes forward KL divergence in trajectory space under the maximum entropy RL framework. A trajectory is a temporal collection of state-action pairs, and this makes the trajectory distribution different from state-action marginal or state marginal distribution. Each objective - minimizing divergence in trajectory space τ, in state-action marginal space (s,a) and state marginal s are different IL methods in their own sense.

MaxEntIRL derives a surrogate objective w.r.t. reward parameter as the difference in cumulative rewards of the trajectories between the expert and the soft-optimal policy under current reward function. To train the soft-optimal policy, it requires running MaxEnt RL in an inner loop after every reward update. This algorithm has been successfully applied for predicting behaviors of taxi drivers with a linear parameterization of reward. Wulfmeier et al. [122] shows that MaxEntIRL reward function can be parameterized as deep neural networks as well.

Guided cost learning (GCL) [30] is one of the first methods to train rewards using neural network directly through experiences from real robots. They achieve this result by leveraging guided policy search for policy optimization, employing importance sampling to correct for distribution shift when the policy has not converged, and using novel regularizations in reward network. GCL optimizes for the same objective as MaxEntIRL and Deep MaxEntIRL. To summarize these three works, we have the following observation:

Observation B.2.0.1. *MaxEntIRL, Deep MaxEntIRL, GCL all optimize for the forward KL divergence in trajectory space, i.e. $D_{\mathrm{KL}}(\rho_E(\tau) \,||\, \rho_\theta(\tau))$.*

B.2.2 GAN-GCL [29], AIRL [32], EAIRL [92]

Finn et al. [29] shows that GCL is equivalent to training GANs with a special structure in the discriminator (GAN-GCL). Note that this result uses an approximation in importance sampling, and hence the gradient estimator is *biased*. Fu et al. [32] shows that GAN-GCL does not perform well in practice since its discriminator models density ratio over trajectories which leads to high variance. They propose an algorithm AIRL in which the discriminator estimates the density ratio of state-action marginal, and shows that AIRL empirically performs better than GAN-GCL. AIRL also uses approximate importance sampling in its derivation, and therefore its gradient is also *biased*. GAN-GCL and AIRL claim to be able to recover a reward function due to the special structure in the discriminator. EAIRL [92] uses empowerment regularization on policy objective based on AIRL.

All the above algorithm intend to optimize for same objective as MaxEntIRL. However, there is an approximation involved in the procedure and let us analyze what that is, by going through the derivation for *equivalence of AIRL to MaxEntIRL* as shown in Fu et al. [32] (Appendix A of that paper).

The authors start from writing down the objective for MaxEntIRL: $\max_\theta L_{\text{MaxEntIRL}}(\theta) = \mathbb{E}_{\tau\sim D}[\log p_\theta(\tau)]$, where D is the collection of expert demonstrations, and reward function is parameterized by θ.

When the trajectory distribution is induced by the soft-optimal policy under reward r_θ, it can be parameterized as $p_\theta(\tau) \propto p(s_0)\prod_{t=0}^{T-1} p(s_{t+1}|s_t,a_t)e^{r_\theta(s_t,a_t)}$, then its gradient is derived as follows:

$$\begin{aligned}\frac{d}{d\theta}L_{\text{MaxEntIRL}}(\theta) &= \mathbb{E}_D\left[\frac{d}{d\theta}r_\theta(s_t,a_t)\right] - \frac{d}{d\theta}\log(Z_\theta)\\ &= \mathbb{E}_D\left[\sum_{t=1}^{T}\frac{d}{d\theta}r_\theta(s_t,a_t)\right] - \mathbb{E}_{p_\theta}\left[\sum_{t=1}^{T}\frac{d}{d\theta}r_\theta(s_t,a_t)\right]\\ &= \sum_{t=1}^{T}\mathbb{E}_D\left[\frac{d}{d\theta}r_\theta(s_t,a_t)\right] - \mathbb{E}_{p_{\theta,t}}\left[\frac{d}{d\theta}r_\theta(s_t,a_t)\right]\end{aligned} \tag{B.19}$$

where Z_θ is the normalizing factor of $p_\theta(\tau)$, and $p_{\theta,t}(s_t,a_t) = \int_{s_{t'!=t},a_{t'!=t}} p_\theta(\tau)$ denote the state action marginal at time t.

As it is difficult to draw samples from p_θ, the authors instead train a separate importance sampling distribution $\mu(\tau)$. For the choice of distribution they follow [30] and use a mixture policy $\mu(a|s) = 0.5\pi(a|s) + 0.5\hat{q}(a|s)$ where $\hat{q}(a|s)$ is the rough density estimate trained on the demonstrations. This is justified as reducing the variance of the importance sampling distribution. Thus the new gradient becomes:

$$\frac{d}{d\theta}L_{\text{MaxEntIRL}}(\theta) = \sum_{t=1}^{T}\mathbb{E}_D\left[\frac{d}{d\theta}r_\theta(s_t,a_t)\right] - \mathbb{E}_{\mu_t}\left[\frac{p_{\theta,t}(s_t,a_t)}{\mu_t(s_t,a_t)}\frac{d}{d\theta}r_\theta(s_t,a_t)\right] \tag{B.20}$$

We emphasize here $\hat{q}(a|s)$ is the density estimate trained on the demonstrations.

They additionally aim to adapt the importance sampling distribution to reduce variance by minimizing $D_{\mathrm{KL}}(\pi(\tau) \,||\, p_\theta(\tau))$, and this KL objective can be simplified to the following MaxEnt RL objective:

$$\max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{T} r_\theta(s_t, a_t) - \log \pi(a_t|s_t) \right] \tag{B.21}$$

This ends the derivation of gradient of MaxEntIRL. Now, AIRL tries to show that the gradient of AIRL matches the gradient for MaxEntIRL objective shown above, i.e. $\frac{d}{d\theta} L_{\mathrm{MaxEntIRL}}(\theta) = \frac{d}{d\theta} L_{\mathrm{AIRL}}(\theta)$, then AIRL is equivalent to MaxEntIRL to a constant, i.e. $L_{\mathrm{MaxEntIRL}}(\theta) = L_{\mathrm{AIRL}}(\theta) + C$.

In AIRL, the cost learning objective is replaced by training a discriminator of the following form:

$$D_\theta(s, a) = \frac{e^{f_\theta(s,a)}}{e^{f_\theta(s,a)} + \pi(a|s)} \tag{B.22}$$

The objective of the discriminator is to maximize the cross-entropy between the expert demonstrations and the generated samples:

$$\begin{aligned}
\max_\theta L_{\mathrm{AIRL}}(\theta) &= \sum_{t=1}^{T} \mathbb{E}_D[\log D_\theta(s_t, a_t)] + \mathbb{E}_{\pi_t}[\log(1 - D_\theta(s_t, a_t))] \\
&= \sum_{t=1}^{T} \mathbb{E}_D \left[\log \frac{e^{f_\theta(s_t,a_t)}}{e^{f_\theta(s_t,a_t)} + \pi(a_t|s_t)} \right] + \mathbb{E}_{\pi_t} \left[\log \frac{\pi(a_t|s_t)}{\pi(a_t|s_t) + e^{f_\theta(s_t,a_t)}} \right] \\
&= \sum_{t=1}^{T} \mathbb{E}_D[f_\theta(s_t, a_t)] + \mathbb{E}_{\pi_t}[\log \pi(a_t|s_t)] - 2\mathbb{E}_{\mu_t}\left[\log(\pi(a_t|s_t)) + e^{f_\theta(s_t,a_t)}\right]
\end{aligned} \tag{B.23}$$

where μ_t is the mixture of state-action marginal from expert demonstrations and from state-action marginal induced by current policy π at time t.

In AIRL, the policy π is optimized with the following reward:

$$\begin{aligned}
\hat{r}(s, a) &= \log(D_\theta(s, a)) - \log(1 - D_\theta(s, a)) \\
&= f_\theta(s, a) - \log \pi(a|s)
\end{aligned} \tag{B.24}$$

Taking the derivative with respect to θ,

$$\frac{d}{d\theta} L_{\mathrm{AIRL}}(\theta) = \sum_{t=1}^{T} \mathbb{E}_D \left[\frac{d}{d\theta} f_\theta(s_t, a_t) \right] - \mathbb{E}_{\mu_t} \left[\frac{e^{f_\theta(s_t,a_t)}}{(e^{f_\theta(s_t,a_t)} + \pi(a_t|s_t))/2} \frac{d}{d\theta} f_\theta(s_t, a_t) \right] \tag{B.25}$$

The authors multiply state marginal $\pi(s_t) = \int_a \pi_t(s_t, a_t)$ to the fraction term in the second expectation, and denote that $\hat{p}_{\theta,t}(s_t, a_t) \triangleq e^{f_\theta(s_t,a_t)} \pi(s_t)$ and $\hat{\mu}_t(s_t, a_t) \triangleq (e^{f_\theta(s_t,a_t)} + \pi(a_t|s_t))\pi(s_t)/2$.

Thus the gradient of the discriminator becomes:

$$\frac{d}{d\theta} L_{\mathrm{AIRL}}(\theta) = \sum_{t=1}^{T} \mathbb{E}_D \left[\frac{d}{d\theta} f_\theta(s_t, a_t) \right] - \mathbb{E}_{\mu_t} \left[\frac{\hat{p}_{\theta,t}(s_t, a_t)}{\hat{\mu}_t(s_t, a_t)} \frac{d}{d\theta} f_\theta(s_t, a_t) \right] \tag{B.26}$$

AIRL is Not Equivalent to MaxEntIRL

The issues occurs when AIRL tried to match Eq. B.26 with Eq. B.36, i.e. $\frac{d}{d\theta}L_{\text{MaxEntIRL}}(\theta) \overset{?}{=} \frac{d}{d\theta}L_{\text{AIRL}}(\theta)$ with same reward parameterization $f_\theta = r_\theta$.

If they are equivalent, then we have the importance weights equality:

$$\boxed{\hat{p}_{\theta,t}(s_t, a_t) = p_{\theta,t}(s_t, a_t), \hat{\mu}_t(s_t, a_t) = \mu_t(s_t, a_t)} \tag{B.27}$$

Then given the definitions, we have:

$$\begin{aligned}\hat{p}_{\theta,t}(s_t, a_t) &\triangleq e^{f_\theta(s_t,a_t)}\pi(s_t) = \pi^*_\theta(s_t)\pi^*_\theta(a_t|s_t) \triangleq p_{\theta,t}(s, a)\\ \hat{\mu}_t(s_t, a_t) &\triangleq (e^{f_\theta(s_t,a_t)} + \pi(a_t|s_t))\pi(s_t)/2 = (\pi(a_t|s_t) + \hat{q}(a_t|s_t))\pi(s_t)/2 \triangleq \mu_t(s_t, a_t)\end{aligned} \tag{B.28}$$

where π^*_θ is soft-optimal policy under reward $r_\theta = f_\theta$ (assumption), thus $\log \pi^*_\theta(a_t|s_t) = f_\theta(s_t, a_t)$. Then equivalently,

$$\boxed{e^{f_\theta(s_t,a_t)} = \hat{q}(a_t|s_t) = \pi^*_\theta(a_t|s_t) = \pi(a_t|s_t)} \tag{B.29}$$

Unfortunately these equivalences hold only at the global optimum of the algorithm, when the policy π reaches the expert policy $\pi_E \approx \hat{q}$ and the discriminator is also optimal. This issue also applies to GAN-GCL and EAIRL. Therefore, we have the following conclusion:

Observation B.2.0.2. *GAN-GCL, AIRL, EAIRL are* ***not*** *equivalent to MaxEntIRL, i.e.* ***not*** *minimizing forward KL in trajectory space and possibly optimizing a biased objective.*

AIRL is Not Optimizing Reverse KL in State-Action Marginal

f-MAX [34] (refer to their Appendix C) states that AIRL is equivalent to f-MAX with $f = -\log u$. This would imply that AIRL minimizes reverse KL in state-action marginal space.

However, there are some differences in AIRL algorithm and the f-MAX algorithm with reverse KL divergence. f-MAX[34] considers a vanilla discriminator. This is different than the original AIRL [32], which uses a specially parameterized discriminator. To highlight this difference we refer to f-MAX with $f = -logu$ (called AIRL in their paper) as f-MAX-RKL in this paper, since it aims to minimize reverse-KL between state-action marginal. We see below that using f-MAX method with special discriminator(instead of vanilla) might not correspond to reverse KL minimization in state-action marginal which shows that AIRL does not truly minimize reverse KL divergence.

To show the equivalence of AIRL to reverse KL matching objective, Ghasemipour et al. [34] considers that the AIRL discriminator can be trained till convergence. With the special discriminator of AIRL, at convergence the following equality holds:

$$\frac{e^{f_\theta(s,a)}}{e^{f_\theta(s,a)} + \pi(a|s)} \equiv \frac{\rho_E(s, a)}{\rho_E(s, a) + \rho_\theta(s, a)} \text{ (at convergence)} \tag{B.30}$$

but if this is true then $f_\theta(s, a)$ can no longer be interpreted as the stationary reward function as it is a function of current policy:

$$f_\theta(s, a) = \frac{\rho_E(s, a)}{\rho_\theta(s, a)}\pi(a|s) \tag{B.31}$$

Observation B.2.0.3. *AIRL is **not** optimizing reverse KL in state-action marginal space.*

B.2.3 GAIL [45], FAIRL, f-MAX-RKL [34]

Generative Adversarial Imitation Learning (GAIL) [45] addresses the issue of running RL in an inner step by adversarial training [35]. A discriminator learns to differentiate over state-action marginal and a policy learns to maximize the rewards acquired from the discriminator in an alternating fashion. It can be further shown that the GAIL is minimizing the Jensen-Shannon divergence over state-action marginal given optimal discriminator.

Recently the idea of minimizing the divergence between expert and policy's marginal distribution is further comprehensively studied and summarized in Ke et al. [51] and Ghasemipour et al. [34], where the authors show that any f-divergence can be minimized for imitation through f-GAN framework [84]. f-MAX proposes several instantiations of f-divergence: forward KL for f-MAX-FKL (FAIRL), reverse KL for f-MAX-RKL, and Jensen-Shannon for original GAIL. Their objectives are summarized as below, where θ is policy parameter, f^* is the convex conjugate of f and T_ω is the discriminator.

$$\min_\theta D_f(\rho_E(s,a) \,||\, \rho_\theta(s,a)) = \min_\theta \max_\omega \mathbb{E}_{(s,a)\sim\rho_E(s,a)}[T_\omega(s,a)] - \mathbb{E}_{(s,a)\sim\rho_\theta(s,a)}[f^*(T_\omega(s,a))] \tag{B.32}$$

These adversarial IRL methods cannot recover a reward function because they do minimax optimization with discriminator in the inner-loop (when optimal, the discriminator predicts $\frac{1}{2}$ everywhere), and have poorer convergence guarantees opposed to using an analytical gradient.

Observation B.2.0.4. *GAIL, FAIRL, f-MAX-RKL are optimizing JS, forward KL, and reverse KL in state-action marginal space, respectively without recovering a reward function.*

B.2.4 SMM [61]

Lee et al. [61] presents state marginal matching (SMM) for efficient exploration by minimizing reverse KL between expert and policy's state marginals (Eq B.33). However, their method cannot recover the stationary reward function because it uses fictitious play between policy π_θ and variational density q, and requires storing a historical average of policies and densities over previous iterations.

$$\max_\theta -D_{\mathrm{KL}}(\rho_\theta(s) \,||\, \rho_E(s)) = \max_\theta \mathbb{E}_{\rho_\theta(s)}\left[\log \frac{\rho_E(s)}{\rho_\theta(s)}\right] = \max_\theta \min_q \mathbb{E}_{\rho_\theta(s)}\left[\log \frac{\rho_E(s)}{q(s)}\right] \tag{B.33}$$

B.2.5 Summary of IL/IRL Methods: Two Classes of Bilevel Optimization

Now we generalize the related works including our method into *reward-dependent* and *policy-dependent* classes from the viewpoint of optimization objective.

For the **reward-dependent** (IRL) methods such as MaxEntIRL, AIRL, and our method, the objective of reward/discriminator r_θ and policy π_ϕ can be viewed as a bilevel optimization:

$$\begin{aligned} &\min_{\theta,\phi} L(r_\theta, \pi_\phi) \\ &\text{s.t. } \phi \in \arg\max_{\phi} g(r_\theta, \pi_\phi) \end{aligned} \tag{B.34}$$

where $L(\cdot,\cdot)$ is the joint loss function of reward and policy, and $g(r,\cdot)$ is the objective of policy given reward r. Thus the optimal policy is dependent on current reward, and training on the final reward does produce optimal policy, i.e. recovering the reward.

For the **policy-dependent** (IL) method such as f-MAX, GAIL, and SMM, the objective of reward/discriminator r_θ and policy π_ϕ can be viewed as:

$$\max_{\phi} \min_{\theta} L(r_\theta, \pi_\phi) \tag{B.35}$$

This is a special case of bilevel optimization, minimax game. The optimal reward is dependent on current policy as the inner objective is on reward, thus it is non-stationary and cannot guarantee to recover the reward.

B.3 Implementation Details

B.3.1 Matching the Specified Expert State Density on Reacher (Sec 4.3.1)

Environment: The OpenAI gym `Reacher-v2` environment [15] has a robotic arm with 2 DOF on a 2D arena. The state space is 8-dimensional: sine and cosine of both joint angles, and the position and velocity of the arm fingertip in x and y direction. The action controls the torques for both joints. The lengths of two bodies are $r_1 = 0.1, r_2 = 0.11$, thus the trace space of the fingertip is an annulus with $R = r_1 + r_2 = 0.21$ and $r = r_2 - r_1 = 0.01$. Since r is very small, it can be approximated as a disc with radius $R = 0.21$. The time horizon is $T = 30$. We remove the object in original reacher environment as we only focus on the fingertip trajectories.

Expert State Density: The domain is x-y coordinate of fingertip position. We experiment with the following expert densities:

- *Single Gaussian:* $\mu = (-R, 0) = (-0.21, 0), \sigma = 0.05$.
- *Mixture of two equally-weighted Gaussians:* $\mu_1 = (-R/\sqrt{2}, -R/\sqrt{2}), \mu_2 = (-R/\sqrt{2}, R/\sqrt{2}), \sigma_1 = \sigma_2 = 0.05$

Training Details: We use SAC as the underlying RL algorithm for all compared methods. The policy network is a tanh squashed Gaussian, where the mean and std is parameterized by a (64, 64) ReLU MLP with two output heads. The Q-network is a (64, 64) ReLU MLP. We use Adam to optmize both the policy and the Q-network with a learning rate of 0.003. The temperature parameter α is fixed to be 1. The replay buffer has a size of 12000, and we use a batch size of 256.

For f-IRL and MaxEntIRL, the reward function is a (64, 64) ReLU MLP. We clamp the output of the network to be within the range [-10, 10]. We also use Adam to optimize the reward network with a learning rate of 0.001.

For other baselines including AIRL, f-MAX-RKL, GAIL, we refer to the f-MAX [34] authors' official implementation[2]. We use the default discriminator architecture as in [34]. In detail, first the input is linearly embedded into a 128-dim vector. This hidden state then passes through 6 Resnet blocks of 128-dimensions; the residual path uses batch normalization and tanh activation. The last hidden state is then linearly embedded into a single-dimensional output, which is the logits of the discriminator. The logit is clipped to be within the range $[-10, 10]$. The discriminator is optimized using Adam with a learning rate of 0.0003 and a batch size of 128.

At each epoch, for all methods, we train SAC for 10 episodes using the current reward/discriminator. We warm-start SAC policy and critic networks from networks trained at previous iteration. We do not empty the replay buffer, and leverage data collected in earlier iterations for training SAC. We found this to be effective empirically, while saving lots of computation time for the bilevel optimization.

For f-IRL and MaxEntIRL, we update the reward for 2 gradient steps in each iteration. For AIRL, f-MAX-RKL and GAIL, the discriminator takes 60 gradient steps per epoch. We train all methods for 800 epochs.

f-IRL and MaxEntIRL require an estimation of the agent state density. We use kernel density estimation to fit the agent's density, using epanechnikov kernel with a bandwidth of 0.2 for pointmass, and a bandwidth of 0.02 for Reacher. At each epoch, we sample 1000 trajectories (30000 states) from the trained SAC to fit the kernel density model.

Baselines: Since we assume only access to expert density instead of expert trajectories in traditional IL framework, we use *importance sampling* for the expert term in the objectives of baselines.

- *For MaxEntIRL*: Given the reward is only dependent on state, its reward gradient can be transformed into covariance in state marginal space using importance sampling from agent states:

$$\begin{aligned}\nabla_\theta L_{\text{MaxEntIRL}}(\theta) &= \frac{1}{\alpha}\sum_{t=1}^{T}\left(\mathbb{E}_{s_t\sim\rho_{E,t}}[\nabla r_\theta(s_t)] - \mathbb{E}_{s_t\sim\rho_{\theta,t}}[\nabla r_\theta(s_t)]\right) \\ &= \frac{T}{\alpha}\left(\mathbb{E}_{s\sim\rho_E}[\nabla r_\theta(s)] - \mathbb{E}_{s\sim\rho_\theta}[\nabla r_\theta(s)]\right) \\ &= \frac{T}{\alpha}\left(\mathbb{E}_{s\sim\rho_\theta}\left[\frac{\rho_E(s)}{\hat{\rho}_\theta(s)}\nabla r_\theta(s)\right] - \mathbb{E}_{s\sim\rho_\theta}[\nabla r_\theta(s)]\right)\end{aligned} \tag{B.36}$$

  where $\rho_t(s)$ is state marginal at timestamp t, and $\rho(s) = \sum_{t=1}^{T}\rho_t(s)/T$ is state marginal averaged over all timestamps, and we fit a density model to the agent distribution as $\hat{\rho}_\theta$.

- *For GAIL, AIRL, f-MAX-RKL*: Original discriminator needs to be trained using expert samples, thus we use the same density model as described above, and then use importance sampling to compute the discriminator objective:

$$\max_D L(D) = \mathbb{E}_{s\sim\rho_\theta}\left[\frac{\rho_E(s)}{\hat{\rho}_\theta(s)}\log D(s)\right] + \mathbb{E}_{s\sim\rho_\theta}[\log(1 - D(s))] \tag{B.37}$$

[2] `https://github.com/KamyarGh/rl_swiss`

Evaluation: For the approximation of both forward and reverse KL divergence, we use non-parametric Kozachenko-Leonenko estimator [56, 57] with lower error [107] compared to plug-in estimators using density models. Suggested by [114][3], we choose $k = 3$ in k-nearest neighbor for Kozachenko-Leonenko estimator. Thus for each evaluation, we need to collect agent state samples and expert samples for computing the estimators.

In our experiments, before training we sample $M = 10000$ expert samples and keep the valid ones within observation space. For agent, we collect 1000 trajectories of $N = 1000 * T = 30000$ state samples. Then we use these two batches of samples to estimate KL divergence for every epoch during training.

B.3.2 Inverse Reinforcement Learning Benchmarks (Sec 4.3.2)

Environment: We use the `Hopper-v2`, `Ant-v2`, `HalfCheetah-v2`, `Walker2d-v2` environments from OpenAI Gym.

Expert Samples: We use SAC to train expert policies for each environment. SAC uses the same policy and critic networks, and the learning rate as section B.3.1. We train using a batch size of 100, a replay buffer of size 1 million, and set the temperature parameter α to be 0.2. The policy is trained for 1 million timesteps on Hopper, and for 3 million timesteps on the other environments. All algorithms are tested on 1, 4, and 16 trajectories collected from the expert stochastic policy.

Training Details: We train f-IRL, Behavior Cloning (BC), MaxEntIRL, AIRL, and f-MAX-RKL to imitate the expert using the provided expert trajectories.

We train f-IRL using Algorithm 2. Since we have access to expert samples, we use the practical modification described in section 4.2.4 for training f-IRL, where we feed a mixture of 10 agent and 10 expert trajectories (resampled with replacement from provided expert trajectories) into the reward objective.

SAC uses the same hyperparameters used for training expert policies. Similar to the previous section, we warm-start the SAC policy and critic using trained networks from previous iterations, and train them for 10 episodes. At each iteration, we update the reward parameters once using Adam optimizer. For the reward network of f-IRL and MaxEntIRL, we use the same reward structure as section B.3.1 with the learning rate of 0.0001, and ℓ_2 weight decay of 0.001. We take one gradient step for the reward update.

MaxEntIRL is trained in the standard manner, where the expert samples are used for estimating reward gradient.

For Behavior cloning, we use the expert state-action pairs to learn a stochastic policy that maximizes the likelihood on expert data. The policy network is same as the one used in SAC for training expert policies.

For f-MAX-RKL and AIRL, we tuned the hyperparameters based on the code provided by f-MAX that is used for *state-action* marginal matching in Mujoco benchmarks. For f-MAX-RKL, we fix SAC temperature $\alpha = 0.2$, and tuned reward scale c and gradient penalty coefficient λ suggested by the authors, and found that $c = 0.2, \lambda = 4.0$ worked for $\{$Ant, Hopper, Walker2d$\}$ *with* the normalization in each dimension of states and with a replay buffer of size 200000.

[3]`https://github.com/gregversteeg/NPEET`

However, for HalfCheetah, we found it only worked with $c = 2.0, \lambda = 2.0$ *without* normalization in states and with a replay buffer of size 20000. For the other hyperparameters and training schedule, we keep them same as f-MAX original code: e.g. the discriminator is parameterized as a two-layer MLP of hidden size 128 with tanh activation and the output clipped within [-10,10]; the discriminator and policy are alternatively trained once for 100 iterations per 1000 environment timesteps.

For AIRL, we re-implement a version that uses SAC as the underlying RL algorithm for a fair comparison, whereas the original paper uses TRPO. Both the reward and the value model are parameterized as a two-layer MLP of hidden size 256 and use ReLU as the activation function. For SAC training, we tune the learning rates and replay buffer sizes for different environments, but find it cannot work on all environments other than HalfCheetah even after tremendous tuning. For reward and value model training, we tune the learning rate for different environments. These hyper-parameters are summarized in table B.1. We set $\alpha = 1$ in SAC for all environments. For every 1000 environment steps, we alternatively train the policy and the reward/value model once, using a batch size of 100 and 256.

Hyper-parameter	Ant	Hopper	Walker	HalfCheetah
SAC learning rate	$3e-4$	$1e-5$	$1e-5$	$3e-4$
SAC replay buffer size	1000000	1000000	1000000	10000
Reward/Value model learning rate	$1e-4$	$1e-5$	$1e-5$	$1e-4$

Supplementary Table B.1: AIRL IRL benchmarks task-specific hyper-parameters.

Evaluation: We compare the trained policies by f-IRL, BC, MaxEntIRL, AIRL, and f-MAX-RKL by computing their returns according to the ground truth return on each environment. We report the mean of their performance across 3 seeds.

For the IRL methods, f-IRL, MaxEntIRL, and AIRL, we also evaluate the learned reward functions. We train SAC on the learned rewards, and evaluate the performance of learned policies according to ground-truth rewards.

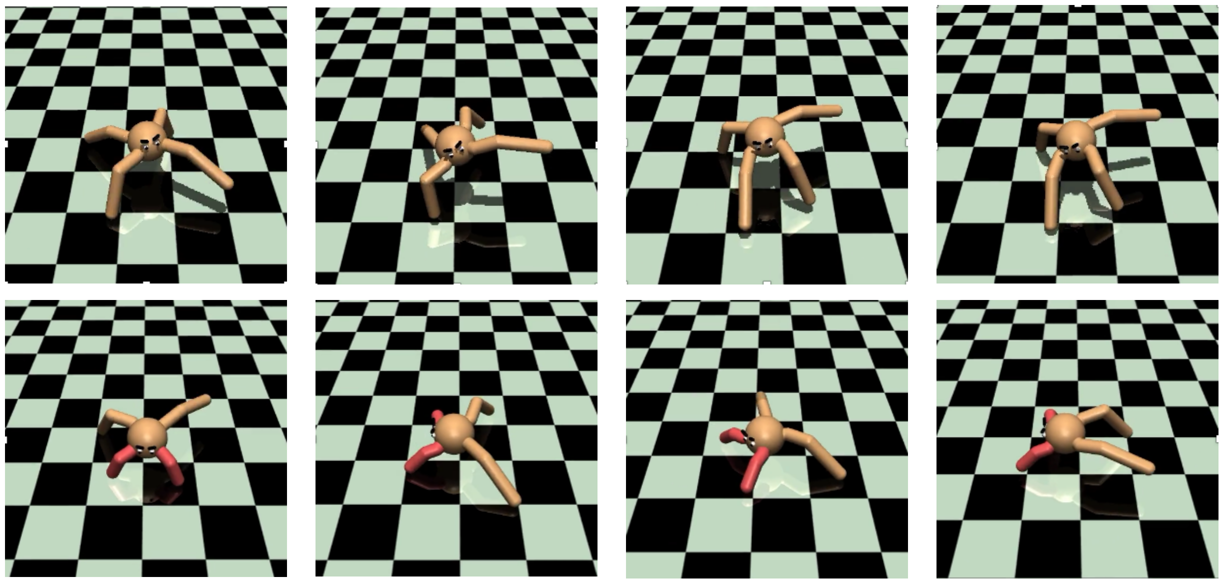
B.3.3 Reward Prior for Downstream Hard-exploration Tasks (Sec 4.3.3.1)

Environment: The pointmass environment has 2D square state space with range $[0, 6]^2$, and 2D actions that control the delta movement of the agent in each dimension. The agent starts from the bottom left corner at coordinate $(0, 0)$.

Task Details: We designed a hard-to-explore task for the pointmass. The grid size is 6×6, the agent is always born at $[0, 0]$, and the goal is to reach the region $[5.95, 6] \times [5.95, 6]$. The time horizon is $T = 30$. The agent only receives a reward of 1 if it reaches the goal region. To make the task more difficult, we add two distraction goals: one is at $[5.95, 6] \times [0, 0.05]$, and the other at $[0, 0.05] \times [5.95, 6]$. The agent receives a reward of 0.1 if it reaches one of these distraction goals. Vanilla SAC always converges to reaching one of the distraction goals instead of the real goal.

Training Details: We use SAC as the RL algorithm. We train SAC for 270 episodes, with a batch size of 256, a learning rate of 0.003, and a replay buffer size of 12000. To encourage the exploration of SAC, we use a random policy for the first 100 episodes.

B.3.4 Reward Transfer across Changing Dynamics (Sec 4.3.3.2)



Supplementary Figure B.1: **Top row**: A healthy Ant executing a forward walk. **Bottom row**: A successful transfer of walking behavior to disabled Ant with 2 legs active. The disabled Ant learns to use the two disabled legs as support and crawl forward, executing a very different gait than previously seen in healthy Ant.

Environment: In this experiment, we use Mujoco to simulate a healthy Ant, and a disabled Ant with two broken legs (Figure B.1). We use the code provided by Fu et al. [32]. Note that this Ant environment is a slightly modified version of the `Ant-v2` available in OpenAI gym.

Expert Samples: We use SAC to obtain a forward-running policy for the Ant. We use the same network structure and training parameters as section B.3.2 for training this policy. We use 16 trajectories from this policy as expert demonstrations for the task.

Training Details: We train f-IRL and MaxEntIRL using the same network structure and training parameters as section B.3.2. We also run AIRL, but couldn't match the performance reported in Fu et al. [32].

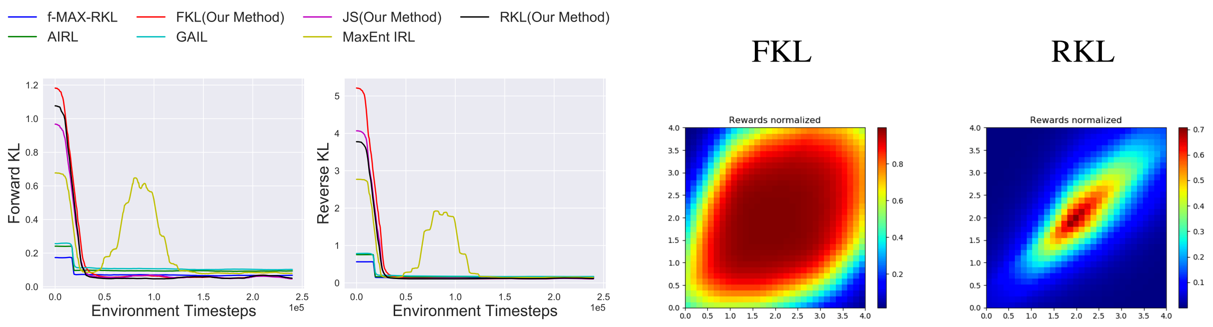
Evaluation: We evaluate f-IRL and MaxEntIRL by training a policy on their learned rewards using SAC. We report the return of this policy on the disabled Ant environment according to the ground-truth reward for forward-running task. Note that we directly report results for policy transfer using GAIL, and AIRL from Fu et al. [32].

B.4 Additional Experiment Results

B.4.1 Inverse RL Benchmark Unnormalized Performance

In this section, we report the unnormalized return of the agent stochastic policy for ease of comparison to expert in Table B.2. We analyze situations when we are provided with 1,4 and 16 expert trajectories respectively. For IL/IRL methods, all the results are averaged across three random seeds to show the mean and standard deviation in the last 10% training iterations.

Note that for the row of "Expert return", we compute the mean and std among the expert trajectories (by stochastic policy) we collected, so for one expert trajectory, it does not have std. Moreover, since we pick the best expert trajectories for training IL/IRL algorithms, the std of "Expert return" is often lower than that of IL/IRL.



Supplementary Figure B.2: Left: Forward and Reverse KL curves in pointmass environment for Gaussian target density for all the methods during training. Smoothed in a window of 120 evaluations. Right: Learned rewards by f-IRL by optimizing for Forward KL (left) and Reverse KL (right) objective in pointmass goal-reaching task.

B.4.2 Additional Result of Reward Transfer across Changing Dynamics

Policy Transfer using GAIL	AIRL	MaxEntIRL	f-IRL	Ground-truth Reward
-29.9	130.3	145.5	**232.5**	315.5

Supplementary Table B.3: Returns obtained after transferring the policy/reward on modified Ant environment using different IL methods. In this case, we report the performance of best seed with a maximum of 50 expert trajectories.

In section 4.3.3 ("Reward transfer across changing dynamics") we show the result of the setting with 32 expert trajectories provided. We follow AIRL paper [32] setting for this experiment, but the number of experiment trajectories used in their experiment is *unknown*. We use a maximum of 50 expert trajectories and show the best seed performance in Table B.3. Note that this table has same values as Table 4.4 except for our method. We see that with more expert trajectories f-IRL is able to outperform baselines with a large margin. The disabled Ant agent is able to learn a behavior to walk while taking support from both of its disabled legs.

B.4.3 Matching the Specified Expert State Density on PointMass

We also conducted the experiment described in section B.3.1 on the pointmass environment similar to that in section B.3.3. This environment has size $[4, 4]$, and the target density is a unimodal Gaussian with $\mu = (2, 2), \sigma = 0.5$ for goal-reaching task.

This experiment is didactic in purpose. In the left of Figure B.2, we observe that all methods converge (MaxEntIRL is slightly unstable) and are able to reduce the FKL and RKL to near zero.

In the right of Figure B.2, we observe that rewards learned by f-IRL using Forward KL and Reverse KL divergence objective demonstrate the expected *mode-covering* and *mode-seeking* behavior, respectively.

Method	Hopper		
# Expert traj	1	4	16
Expert return	3570.87	3585.59 ± 12.39	3496.62 ± 10.13
BC	17.39 ± 5.99	468.49 ± 83.94	553.56 ± 46.70
MaxEntIRL	3309.72 ± 171.28	3300.81 ± 229.84	**3298.50** ± 255.35
f-MAX-RKL	**3349.62** ± 68.89	3326.83 ± 85.42	3165.51 ± 102.83
AIRL	49.12 ± 2.58	49.33 ± 3.93	48.63 ± 5.88
FKL (f-IRL)	3329.94 ± 152.33	3243.83 ± 312.44	3260.35 ± 175.58
RKL (f-IRL)	3276.55 ± 221.27	3303.44 ± 286.26	3250.74 ± 161.89
JS (f-IRL)	3282.37 ± 202.30	**3351.99** ± 172.70	3269.49 ± 160.99

Method	Walker2d		
# Expert traj	1	4	16
Expert return	5468.36	5337.85 ± 92.46	5368.01 ± 78.99
BC	-2.03 ± 1.05	303.24 ± 6.95	431.60 ± 63.68
MaxEntIRL	4823.82 ± 512.58	4697.11 ± 852.19	**4884.30** ± 467.16
f-MAX-RKL	2683.11 ± 128.14	2628.10 ± 548.93	2498.78 ± 824.26
AIRL	9.8 ± 1.82	9.24 ± 2.28	8.45 ± 1.56
FKL (f-IRL)	**4927.02** ± 615.34	4809.80 ± 750.05	4851.81 ± 547.12
RKL (f-IRL)	4847.12 ± 806.61	4806.72 ± 433.02	4578.39 ± 564.17
JS (f-IRL)	4888.09 ± 664.86	**4935.42** ± 384.15	4725.78 ± 613.45

Method	HalfCheetah		
# Expert traj	1	4	16
Expert return	12258.71	11944.45 ± 985.08	12406.29 ± 614.02
BC	-367.56 ± 23.57	209.59 ± 178.71	287.05 ± 109.32
MaxEntIRL	**11637.41** ± 438.16	11685.92 ± 478.95	11228.32 ± 1752.32
f-MAX-RKL	8688.37 ± 633.58	4920.66 ± 2996.15	8108.81 ± 1186.77
AIRL	2366.84 ± 175.51	2343.17 ± 103.51	2267.68 ± 83.59
FKL (f-IRL)	11556.23 ± 539.83	11556.51 ± 673.13	11642.72 ± 629.29
RKL (f-IRL)	11612.46 ± 703.25	11644.19 ± 488.79	**11899.50** ± 605.43
JS (f-IRL)	11413.47 ± 1227.89	**11686.09** ± 748.30	11711.77 ± 1091.74

Method	Ant		
# Expert traj	1	4	16
Expert return	5926.18	5859.09 ± 88.72	5928.87 ± 136.44
BC	-113.60 ± 12.86	1321.69 ± 172.93	2799.34 ± 298.93
MaxEntIRL	3179.23 ± 2720.63	4171.28 ± 1911.67	4784.78 ± 482.01
f-MAX-RKL	3585.03 ± 255.91	3810.56 ± 252.57	3653.53 ± 403.73
AIRL	-54.7 ± 28.5	-14.15 ± 31.65	-49.68 ± 41.32
FKL (f-IRL)	**4859.86** ± 302.94	**4861.91** ± 452.38	**4971.11** ± 286.81
RKL (f-IRL)	3707.32 ± 2277.74	4814.58 ± 376.13	4813.80 ± 361.93
JS (f-IRL)	4590.11 ± 1091.22	4745.11 ± 348.97	4342.39 ± 1296.93

Supplementary Table B.2: Benchmark of Mujoco Environment, from top to bottom, Hopper-v2, Walker2d-v2, HalfCheetah-v2, Ant-v2.

Appendix C

Appendix for SoftGym

C.1 Environment Details

C.1.1 Observation Space

Each task supports three types of observation space: Full state of the particles, reduced states and image based observation. For image-based observation, the agent receives an RGB image of the environment rendered by the Flex simulator, with a size $d \times d \times 3$, where d is a controllable parameter. For all our image-based experiments we choose $d = 128$. For the full state observation, the state is the positions of all particles, as well as any state of the action space or other rigid objects in the scene.

We now detail the reduced state representation for each task in SoftGym.

TransportWater: the reduced states are the size (width, length, height) of the cup, the target cup position, height of the water in the cup, amount of water inside and outside of the cup.

PourWater and PourWaterAmount: the reduced states are the sizes of both cups, the x, y-position and rotation of the controlled cup, the initial distance between the controlled cup and the current cup, the height of the water in the cup and the amount of water in both cups. For PourWaterAmount, we have an additional value indicating the amount of water to be poured.

Rope Environments: For rope enviornments, including the StraightenRope and RopeConfiguration, we pick 10 evenly-spaced keypoints on the rope, including the two end points, and use the positions of these key points as the reduced state.

Cloth Environments: For all of the cloth related environments (SpreadCloth, FoldCloth(Crumpled), DropCloth, DropFoldCltoh), the reduced states are the positions of the four corners of the cloth.

For environments using any pickers or robots, the positions of the pickers or joint positions of the robot are included in the reduced state.

C.1.2 Action Space

For all environments, we normalize the action space to be within $[-1, 1]$ for the agents. Below we will describe the un-normalized action range for each environment, using meter or radian as the unit by default.

TransportWater: The motion of the cup is constrained to be in one dimension. The action is also in one dimension and is the increment of the the position of the cup along the dimension. The action range is $[-0.011, 0.011]$.

PourWater, PourWaterAmount: The action is $a = (dx, dy, d\theta)$, denoting the change of the position and rotation of the cup. $dx, dy \in [-0.01, 0.01]$ and $d\theta \in [-0.015, 0.015]$.

Picker: For the cloth and rope environments, we use either two pickers or one robot. A picker abstracts a controller which can pick and place objects. A picker is modeled as a sphere with a radius of 0.05. For each picker, the action is $a = (dx, dy, dz, d)$. $dx, dy, dz \in [-0.01, 0.01]$ indicate the change of the position of the picker. $d \in [0, 1]$ indicates the picking state of the picker. If $d > 0.5$, the particle closest to the picker will be attached to the picker and follow its movement. When $d < 0.5$, the picked particle will be released.

C.1.3 Task Variations

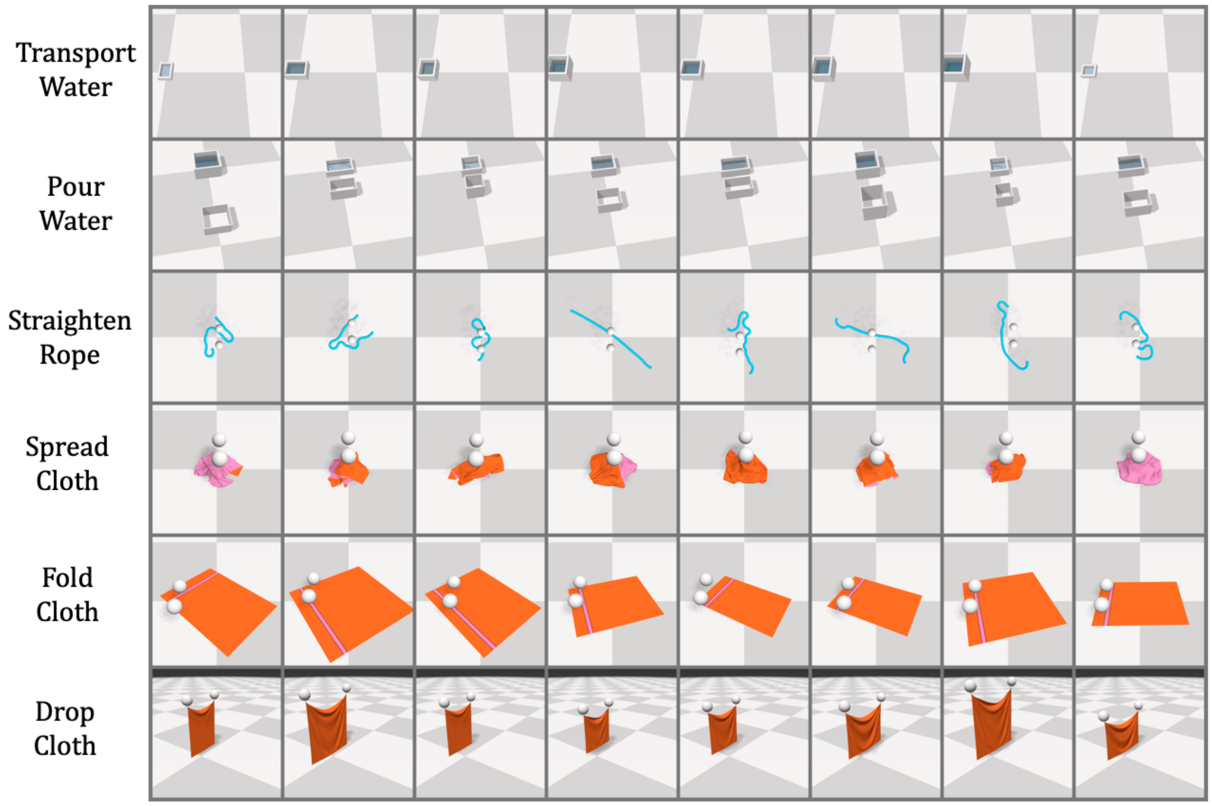
Environment	Variations
TransportWater	cup size, water volume
PourWater	size of both cups, target cup position, water volume
PourWaterAmount	size of both cups, target cup position, water volume, goal volume
StraightenRope, RopeConfiguration	Initial rope configuration
SpreadCloth, FoldClothCrumpled	Cloth size, initial configuration
FoldCloth	Cloth size, random rotation of the inital configuration
DropCloth, DropFoldCloth	Cloth size, initial height

Supplementary Table C.1: Different task variations in all tasks. Refer to the appendix for more details of the ranges of the variations and how they are generated.

In this section we detail how we generate the task variations. Supplementary Figure C.1 shows some of the task variations. Most of the practical tasks related to deformable object manipulation, such as laundry folding, require the agent to deal with variations of the objects in the environment, such as the size, shape, and physical properties. We summarize the variations of each task that we include in SoftGym in Table C.1.

PourWater, PourWaterAmount: For this task, both the controlled cup and the target cup are modeled as cuboids without the top face. We vary the height, length, and width of both cups, the distance between the controlled cup and the target cup, as well as the volume of water in the controlled cup. The water is initially generated in a shape of cuboid. Denote the number of particles along each dimension of the water cuboid as l_w, w_w, h_w respectively. We vary the width w_w and the length l_w of the water cuboid in the range $[4, 13]$. For the height h_w, we first randomly select a water level between 'medium' and 'large'. Let $m = \min(w_w, l_w)$. For level 'medium', the height is $h_w = \lfloor 3.5m \rfloor$. For level 'large', the height is $h_w = 4m$. The total number of water particles is $v = w_w \cdot h_w \cdot l_w$.

Given the volume of water, we then create a cup for holding that amount of water. Denote the radius of the water particles as $r = 0.033$. The width and length of the controlled cup is $w_{cc} = w_w \cdot r + 0.1$ and $l_{cc} = l_w \cdot r + 0.1$, and the width and length of the target cup is $w_{tc} = w_w \cdot r + 0.07$



Supplementary Figure C.1: Illustration of task variations. Each image shows the task after the initial reset. Variations of tasks in SoftGym-Hard are omitted here due to similarity to the ones shown.

and $l_{tc} = l_w \cdot r + 0.07$. Let $h = v/((w_w + 1)(l_w + 1))$ be the number of particles in the height of the water cuboid. For medium volume of water, we have $h_{cc} = h \cdot r/2 + 0.001 \cdot \text{Unif}[-0.5, 0.5]$. For large volume of water, we have $h_{cc} = h \cdot r/3 + 0.001 \cdot \text{Unif}[0, 1]$. The height of the target cup is simply computed as $h_{tc} = h_{cc} + \text{Unif}[0, 0.1]$.

The distance between the controlled cup and target cup is sampled $m \cdot \text{Unif}[0.05m, 0.09m+](w_w + 4)r/2$.

For PourWaterAmount, the goal volume is sampled from $0.1 + \text{Unif}[0, 1] \times 0.9$.

TransportWater: We vary the volume of water and size of cup in this task. The variation is generated almost exactly the same as in PourWater, with the following exceptions. For the medium volume of water, the cup height is computed as $h_{cc} = h \cdot r/2$, and for the large volume of water, the cup height is computed as $h_{cc} = h \cdot r/3 + 0.0015m$.

SpreadCloth, FoldClothCrumpled: We vary the size of the cloth. The cloth is modeled as a particle grid with width w and length l (the number of particles). We sample w and l from randint$(60, 120)$. We also vary the initial crumpled shape of the cloth. This is done by first randomly picking a particle on the cloth, lifting it up to a random height sampled from Unif$[0, 0.5]$, and then dropping it.

FoldCloth: In this task we only vary the size of the cloth. Similar to the SpreadCloth case, the width of the cloth w is sampled from randint$(60, 120)$. The initial state of the cloth is always flattened and centered at the origin.

DropCloth, DropFoldCloth: In this task we vary the size of the cloth in the same way in SpreadCloth. We lift the cloth up by picking up its two corners.

StraightenRope: In this task we use a rope with a fixed length and only vary its initial twisted shape. We generate different twisted shapes by randomly choosing a particle on the rope, picking it up, and then dropping it. We repeat this process for 4 times to make sure the generated shapes are different.

C.1.4 Training and Evaluation

For computation efficiency, we pre-compute 1000 task variations and their initial states for each environment. Out of the 1000 task variations, 800 variations are used during training and 200 variations are used for evaluation.

Performance Metric Besides the reward, we compute a performance metric for each task at each time step, which is the same as the reward without any scaling. At the beginning of each episode, we compute an upper-bound and a lower-bound for the performance metric. For example, for SpreadCloth task, the performance is the covered area of the cloth and the upper-bound is when the cloth is flattened. For any task where the performance is a negative distance function, its upper-bound would be zero. For all tasks except StraightenRope, the lower-bound is the performance achieved at the first time step, which corresponds to the achieved performance when the policy does nothing. For StraightenRope, the lower bound is the possible minimal reward, which is the negative value of the straightened rope's length. Given the upper-bound and lower-bound u, l, we normalize the performance at each time step by

$$\hat{s} = \frac{s - l}{u - l},$$

where $\hat{s}$ is the normalized performance. The normalized performnace at the last time step is reported throughout the paper unless explicitly specified.

C.2 Algorithm Details

For all the tasks and algorithms, we use a discounting factor of $\gamma = 0.99$ when it applies. The action repetition and task horizon are summarized in table C.2.

Parameter	Transport Water	Pour Water	Straighten Rope	Spread Cloth	Fold Cloth	Drop Cloth
Action Repetition	8	8	8	8	8	32
Task Horizon	75	100	75	100	100	15

Supplementary Table C.2: Action repetition and task horizon.

C.2.1 CEM with Dynamics Oracle

For CEM, we use 10 optimization iteration. Model predictive control is used. Different planning horizon is used for different environments, as summarized in Table C.3. A total of 21K environment steps are used for making each decision. The number of candidate trajectories during each planning is thus $21K/planning\_horizon$. The top 10% candidates are selected as the elites for fitting the posterier distribution within each optimization iteration.

Parameter	Transport Water	Pour Water	Straighten Rope	Spread Cloth	Fold Cloth	Drop Cloth
Planning Horizon	7	40	15	15	30	15

Supplementary Table C.3: Task specific planning horizon for CEM

C.2.2 SAC and CURL-SAC

We use the CURL-SAC implementation from the released code[1]. Both Q-value network and the policy network are MLPs with 2 hidden layers of 1024 neurons with ReLU as activation function. The hyper-parameters of SAC are summarized in Table C.4. To achieve learning stability, we tuned the reward scaling and learning rate for both SAC and CURL-SAC, for each environment. The parameters are summarized in Table C.5.

Parameter	SAC
batch size	128
initial steps	1000
replay buffer size	1e5
target smoothing coefficient	0.01
alpha	automatic tuning
delayed policy update period	2
target update interval	2

Supplementary Table C.4: General hyper-parameters for SAC.

C.2.3 DrQ

We use the author released code[2] for the benchmarking with mostly default hyper-parameters. The only change in the hyper-parameter is that we use images of size 128×128 instead of 84×84 as in the released code, so we change the padding of the image from 4 to 6. We also tune the reward scaling parameter for different tasks, as summarized in Table C.5.

[1]https://github.com/MishaLaskin/curl
[2]https://github.com/denisyarats/drq

Parameter	Transport Water	Pour Water	Straighten Rope	Spread Cloth	Fold Cloth	Drop Cloth
Reduced State						
learning rate	1e-3	1e-3	1e-3	1e-3	5e-4	1e-3
reward scaling	20	20	50	50	50	50
learning rate decay	-	-	yes	-	-	-
Image						
learning rate	3e-4	3e-4	3e-4	3e-4	1e-4	3e-4
learning rate decay	-	-	yes	yes	-	-
reward scaling	20	20	50	50	50	50

Supplementary Table C.5: SAC task dependent hyper-parameters. If learning rate decay is applied, the actor learning rate is halved every 75K steps and the critic learning rate is halved every 100K steps.

Parameter	Value
training	
optimizer	Adam [54]
learning rate	0.001
Adam ϵ	0.0001
experience replay size	10^6
explore noise	0.3
batch size	50
dynamics chunk size	50
free nats	3
CEM planning	
planning horizon	24
optimization iteration	10
candidate samples	1000
top candidate	100

Supplementary Table C.6: Hyper-parameters for PlaNet

C.2.4 PlaNet

PlaNet takes the image observation as input. The image is first processed by a convolutional neural network to produce an embedding vector. The architecture of the encoding CNN is shown in table C.8. After the final convolution layer, the extracted features are flattened to be a vector, then transformed to an embedding vector by a linear layer. Different algorithms use different sizes for the embedding vector. For PlaNet and RIG, the size is 1024; For SAC and TD3, the size is 256. Different algorithms then process this embedding vector in different ways.

We use a GRU [18] with 200 hidden nodes as the deterministic path in the dynamics model.

layer	input channel	output channel	kernel size	stride
1	1024	128	5	2
2	128	64	5	2
3	64	32	5	2
4	32	16	6	2
5	16	3	6	2

Supplementary Table C.7: Architecture of the deconvolutional neural network (VAE decoder) in PlaNet.

layer	input channel	output channel	kernel size	stride
1	3	16	4	2
2	16	32	4	2
3	32	64	4	2
4	64	128	4	2
5	128	256	4	2

Supplementary Table C.8: Architecture of the encoding CNN.

All functions are implemented as a two-layer MLP with 200 nodes for each layer and ReLU as the activation function. We refer to [40] for more details. We do not include the latent overshooting in our experiment as it does not improve much over the one-step case.

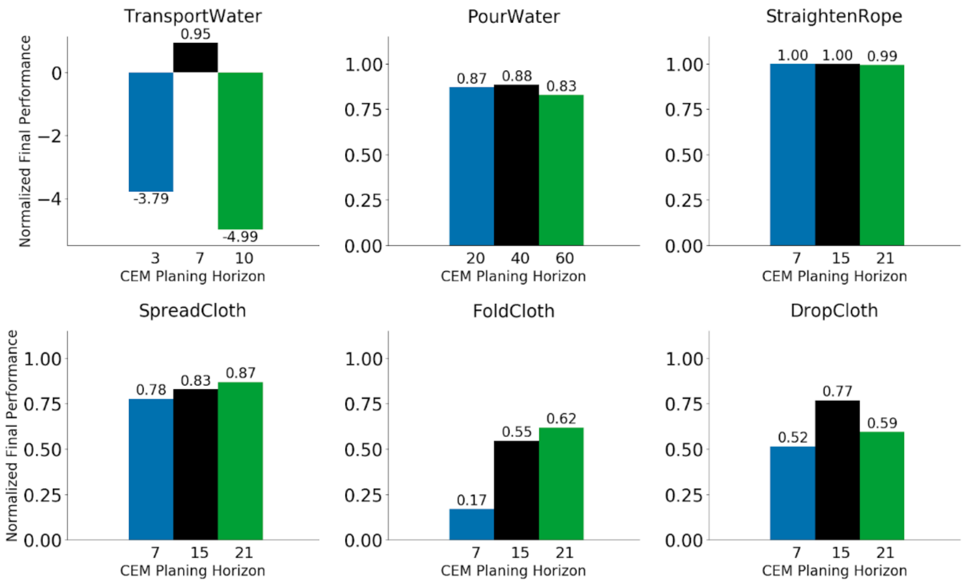
During training, we first collect 5 episodes from a random policy to warm-up the replay buffer. Then, for each training epoch, we first store 900 time steps of experiences collected from the current planning agent and perform 100 gradient updates. The full hyper-parameters are listed in Table C.6.

On an Nvidia 2080Ti GPU with 4 virtual CPUs and 40G RAM, training PlaNet for 1M steps takes around 120 hours.

C.2.5 Wu et al. 20

For the SpreadCloth and FoldCLoth task, we additionally compare to previous work [121] that learns a model-free agent for spreading the cloth from image observation. We take the official implementation from the authors [3]. Here, the action space is pick-and-place. Followed the approach in the paper, during exploration, a random point on the cloth is selected (with a heuristic method for cloth segmentation). The picker then goes to the picked location, picks up the cloth and moves to a place location given by the agent, waits for 20 steps and then drops the cloth. Default hyper-parameters in the original code are used.

[3]`https://github.com/wilson1yan/rlpyt`



Supplementary Figure C.2: Performance of CEM with different planning horizons for each task.

C.3 CEM with Different Planning Horizons

We additionally evaluate CEM with different planning horizons for each task. The results are shown in Figure C.2. We see that the performance of CEM is sensitive to the planning horizon in TransportWater, FoldCloth and DropCloth, whereas the performance is relatively stable in the other tasks. The black bar is the performance that we report in the main paper.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004. 1.3, 2

[2] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in neural information processing systems*, pages 5074–5082, 2016. 1.1, 1.4, 2, 2

[3] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019. 1.1

[4] Syed Mumtaz Ali and Samuel D Silvey. A general class of coefficients of divergence of one distribution from another. *Journal of the Royal Statistical Society: Series B (Methodological)*, 28(1):131–142, 1966. 4.1, B.1.2

[5] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017. 2, 3.2.5

[6] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 1.4

[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017. B.1.3, B.1.3

[8] Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20. Citeseer, 1997. 1.3

[9] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995. 1.3

[10] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. 2

[11] Cristian Bodnar, Adrian Li, Karol Hausman, Peter Pastor, and Mrinal Kalakrishnan. Quantile qt-opt for risk-aware vision-based robotic grasping. *arXiv preprint arXiv:1910.02787*,

2019. 2

[12] Mario Bollini, Stefanie Tellex, Tyler Thompson, Nicholas Roy, and Daniela Rus. Interpreting and executing recipes with a cooking robot. In *Experimental Robotics*, pages 481–495. Springer, 2013. 2

[13] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 3.2.2, A.4

[14] Kianté Brantley, Wen Sun, and Mikael Henaff. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*, 2019. 2

[15] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 1.4, 4.3, B.3.1

[16] Tiffany L Chen, Matei Ciocarlie, Steve Cousins, Phillip M Grice, Kelsey Hawkins, Kaijen Hsiao, Charles C Kemp, Chih-Hung King, Daniel A Lazewatsky, Adam E Leeper, et al. Robots for humanity: using assistive robotics to empower people with disabilities. *IEEE Robotics & Automation Magazine*, 20(1):30–39, 2013. 2

[17] Ricson Cheng, Arpit Agarwal, and Katerina Fragkiadaki. Reinforcement learning of active vision for manipulating objects under occlusions. *arXiv preprint arXiv:1811.08067*, 2018. 2

[18] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. C.2.4

[19] Luis C Cobo, Charles L Isbell Jr, and Andrea L Thomaz. Object focused q-learning for autonomous agents. Georgia Institute of Technology, 2013. 2

[20] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018. 1.3

[21] Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247, 2008. 2

[22] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016. 1.4, 2

[23] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *arXiv preprint arXiv:1710.05268*, 2017. 2

[24] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018. 2

[25] Zackory Erickson, Henry M Clever, Greg Turk, C Karen Liu, and Charles C Kemp. Deep haptic model predictive control for robot-assisted dressing. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018. 2

[26] Nabil Essahbi, Belhassen Chedli Bouzgarrou, and Grigore Gogu. Soft material modeling for robotic manipulation. In *Applied Mechanics and Materials*, volume 162, pages 184–193. Trans Tech Publ, 2012. 2

[27] Linxi Fan, Yuke Zhu, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning (CoRL)*, pages 767–782, 2018. 1.4, 2

[28] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017. 1.1

[29] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016. (document), 2, B.2.2

[30] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58, 2016. (document), 2, 4.2.4, B.2.1, B.2.2

[31] Carlos Florensa, Jonas Degrave, Nicolas Heess, Jost Tobias Springenberg, and Martin Riedmiller. Self-supervised learning of image embedding for continuous control. *workshop at Advances in Neural Information Processing Systems (NIPS) 2018*, 2019. arXiv:1901.00943. 5.3

[32] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017. (document), 1.3, 2, 4.2.3, 4.2.3, 4.3, 2, 4.3.3, B.1.4, B.1.1, B.1.4, B.1.4, B.2.2, B.2.2, B.3.4, B.4.2

[33] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016. 2

[34] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. *arXiv preprint arXiv:1911.02256*, 2019. (document), 1.3, 2, 4.1, 4.3, 2, 4.3.3, B.2.2, B.2.3, B.3.1

[35] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2, 4.2.2, B.2.3

[36] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018. 4.3

[37] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/haarnoja18b.html`. 5.3.2

[38] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic:

Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018. 3.2.5

[39] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning. In *Advances in neural information processing systems*, pages 3909–3917, 2016. 1.3

[40] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018. C.2.4

[41] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, 2019. 5.3.3, 5.4.3

[42] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017. 1.1

[43] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 1.3

[44] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016. 3.1

[45] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016. (document), 1.3, 2, 4.3, 4.3.3, B.2.3

[46] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 3.2.4

[47] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019. 1.3, 1.4

[48] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020. 2

[49] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019. 1.4

[50] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998. 2

[51] Liyiming Ke, Matt Barnes, Wen Sun, Gilwoo Lee, Sanjiban Choudhury, and Siddhartha Srinivasa. Imitation learning as f-divergence minimization. *arXiv preprint*

arXiv:1905.12888, 2019. B.2.3

[52] Ramtin Keramati, Jay Whang, Patrick Cho, and Emma Brunskill. Strategic object oriented reinforcement learning. *arXiv preprint arXiv:1806.00175*, 2018. 2

[53] Fouad F Khalil and Pierre Payeur. Dexterous robotic manipulation of deformable objects with multi-sensory feedback-a review. In *Robot Manipulators Trends and Development*. IntechOpen, 2010. 2

[54] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. **??**

[55] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020. 5.3.3

[56] LF Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987. B.3.1

[57] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004. B.3.1

[58] Sascha Lange, Martin Riedmiller, and Arne Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *The 2012 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2012. 2

[59] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learming with augmented data. arXiv:2004.14990. 5.3.3

[60] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119*, 2020. arXiv:2004.04136. 5.3.3

[61] Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019. (document), 1.3, 2, 4.1, 6, B.2.4

[62] Youngwoon Lee, Edward S Hu, Zhengyu Yang, Alex Yin, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. *arXiv preprint arXiv:1911.07246*, 2019. URL `https://clvrai.com/furniture`. 2

[63] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018. 4.1

[64] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pages 2203–2213, 2017. B.1.3

[65] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019. 1.4

[66] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *2019*

International Conference on Robotics and Automation (ICRA), pages 1205–1211. IEEE, 2019. 5.3.2

[67] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. A.4

[68] Xingyu Lin, Harjatin Singh Baweja, and David Held. Reinforcement learning without ground-truth state. *arXiv preprint arXiv:1905.07866*, 2019. 2, 3.1, 5.3

[69] Fangchen Liu, Zhan Ling, Tongzhou Mu, and Hao Su. State alignment-based imitation learning. *arXiv preprint arXiv:1911.10947*, 2019. 4.3.2

[70] Minghuan Liu, Tairan He, Minkai Xu, and Weinan Zhang. Energy-based imitation learning. *arXiv preprint arXiv:2004.09395*, 2020. 2, 4.3.2

[71] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013. 5.1

[72] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4): 1–12, 2014. 5.1

[73] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *2010 IEEE International Conference on Robotics and Automation*, pages 2308–2315. IEEE, 2010. 1.4, 2

[74] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. *Conference on Robot Learning (CoRL)*, 2018. 1.4, 2

[75] Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Parametrized shape models for clothing. In *2011 IEEE International Conference on Robotics and Automation*, pages 4861–4868. IEEE, 2011. 1.4

[76] Stephen Miller, Jur Van Den Berg, Mario Fritz, Trevor Darrell, Ken Goldberg, and Pieter Abbeel. A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31(2):249–267, 2012. 1.4

[77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 1.4

[78] Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, pages 429–443, 1997. B.1.3

[79] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007. 5.1

[80] Ashvin Nair, Shikhar Bahl, Alexander Khazatsky, Vitchyr Pong, Glen Berseth, and Sergey Levine. Contextual imagined goals for self-supervised robotic learning. In *Conference on Robot Learning*, pages 530–539, 2020. 2

[81] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pages 9191–9200, 2018. 1.2, 2, 3.1, 3.2.5, 5.3, A.7

[82] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999. 4.2.3, 4.3.3

[83] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pages 663–670, 2000. 1.3, 2, B.2.1

[84] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems*, pages 271–279, 2016. B.2.3

[85] Takayuki Osa, Naohiko Sugita, and Mamoru Mitsuishi. Online trajectory planning and force control for automation of surgical tasks. *IEEE Transactions on Automation Science and Engineering*, 15(2):675–691, 2017. 1.3

[86] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepezvari, Satinder Singh, et al. Behaviour suite for reinforcement learning. *International Conference on Leanring Representation (ICLR)*, 2019. 2

[87] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2050–2053, 2018. 2

[88] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016. 1.1

[89] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017. 1.1, 2

[90] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989. 1.3

[91] Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019. (document), 1.2, 2, 3.1, 3.3.1, A.5, A.6, A.7, A.3

[92] Ahmed H Qureshi, Byron Boots, and Michael C Yip. Adversarial imitation via variational inverse reinforcement learning. *arXiv preprint arXiv:1809.06404*, 2018. (document), 2, B.2.2

[93] Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019. 2

[94] Samuel Rodriguez, Xinyu Tang, Jyh-Ming Lien, and Nancy M Amato. An obstacle-based rapidly-exploring random tree. In *Proceedings 2006 IEEE International Conference on*

Robotics and Automation, 2006. ICRA 2006., pages 895–900. IEEE, 2006. 2

[95] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. A.4

[96] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010. 1.3

[97] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999. 5.3.1

[98] Stuart Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998. 1.3, 2, B.2.1

[99] Mitul Saha and Pekka Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, 2007. 2

[100] Jose Sanchez, Juan-Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey. *The International Journal of Robotics Research*, 37(7): 688–716, 2018. 2

[101] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320, 2015. 2

[102] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019. 1.4

[103] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015. 4.3

[104] Jeremy Schwartz, Seth Alter, James J. DiCarlo, Josh H. McDermott, Joshua B. Tenenabum, Daniel L.K. Yamins, Dan Gutfreund, Chuang Gan, James Traer, Jonas Kubilius, Martin Schrimpf, Abhishek Bhandwaldar, Julian De Freitas, Damian Mrowca, Michael Lingelbach, Megumi Sano, Daniel Bear, Kuno Kim, Nick Haber, and Chaofei Fan. Threedworld (tdw): A high-fidelity, multi-modal platform for interactive physical simulation. 07/2020 2020. URL `http://www.threedworld.org/`. 2

[105] Daniel Seita, Aditya Ganapathi, Ryan Hoque, Minho Hwang, Edward Cen, Ajay Kumar Tanwani, Ashwin Balakrishna, Brijen Thananjeyan, Jeffrey Ichnowski, Nawid Jamali, et al. Deep imitation learning of sequential fabric smoothing policies. *arXiv preprint arXiv:1910.04854*, 2019. 2

[106] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*,

pages 1134–1141. IEEE, 2018. 1.4, 2

[107] Shashank Singh and Barnabás Póczos. Analysis of k-nearest neighbor distances with application to entropy estimation. *arXiv preprint arXiv:1603.08578*, 2016. B.3.1

[108] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. volume 80 of *Proceedings of Machine Learning Research*, pages 4732–4741, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/srinivas18b.html`. 5.3

[109] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011. 2

[110] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. 1.4

[111] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. 3.3.1, 4.3, A.5, A.7

[112] Dimitra Triantafyllou and Nikos A Aspragathos. A vision system for the unfolding of highly non-rigid objects on a table by one manipulator. In *International Conference on Intelligent Robotics and Applications*, pages 509–519. Springer, 2011. 2

[113] Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pages 1439–1456, 2020. 2

[114] Greg Ver Steeg. Non-parametric entropy estimation toolbox (npeet). 2000. B.3.1

[115] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008. B.1.3

[116] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. 1.4

[117] Ruohan Wang, Carlo Ciliberto, Pierluigi Amadori, and Yiannis Demiris. Random expert distillation: Imitation learning via expert policy support estimation. *arXiv preprint arXiv:1905.06750*, 2019. 2

[118] Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P Burgess, and Alexander Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019. 2

[119] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, NJ, 1996. 5.2.2

[120] Bryan Willimon, Stan Birchfield, and Ian Walker. Model for unfolding laundry using interactive perception. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4871–4876. IEEE, 2011. 2

[121] Wilson Wu, Yilin adn Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to manipulate deformable objects without demonstrations. *Robotics Science and Systems (RSS)*, 2020. 2, 5.3.3, 5.4.2, C.2.5

[122] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015. (document), 2, B.2.1

[123] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 2

[124] Yu Xiang, Christopher Xie, Arsalan Mousavian, and Dieter Fox. Learning rgb-d feature embeddings for unseen object instance segmentation. In *Conference on Robot Learning (CoRL)*, 2020. 3.2.2

[125] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *Conference on Robot Learning (CoRL)*, 2020. A.7

[126] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *Conference on Robot Learning*, pages 1369–1378, 2020. 3.2.2

[127] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. Unseen object instance segmentation for robotic environments. In *arXiv:2007.08073*, 2020. 3.2.2

[128] M. Yan, Y. Zhu, N. Jin, and J. Bohg. Self-supervised learning of state estimation for manipulating deformable linear objects. *IEEE Robotics and Automation Letters*, pages 1–1, 2020. ISSN 2377-3774. doi: 10.1109/LRA.2020.2969931. 2

[129] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019. 2

[130] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018. 2

[131] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. 2019. 1.1

[132] Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010. 4.1, B.1.1

[133] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. (document), 2, 4.3, B.2.1

[134] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction.

In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 28–31. IEEE, 2004. 3.2.2, A.4

[135] Zoran Zivkovic and Ferdinand Van Der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7): 773–780, 2006. 3.2.2, A.4