LEARNING FROM IMPERFECT DEMONSTRATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Zhangjie Cao
December 2022

Abstract

Imitation learning is one of the most promising robot learning paradigms, which attempts to learn robot policies from demonstrations. Standard imitation learning algorithms assume that the demonstrations are provided by optimal experts who are capable of perfectly performing the task on the robot of interest (i.e., the target environment). However, under these assumptions, imitation learning usually requires large amounts of demonstrations. This is limiting in many environments, where collecting optimal demonstrations is difficult due to various reasons such as difficulty of controlling robots with high degrees of freedom or limited interactions with the environment.

In practice, we often have access to large amounts of imperfect demonstrations, which are possibly not optimal or are produced by different agents with different morphologies or dynamics. Such imperfect demonstrations contain valuable information that can be helpful for learning the optimal policy. However, directly imitating these imperfect demonstrations will lead to learning a suboptimal policy. Instead of direct imitation learning, we propose developing new algorithms to utilize these imperfect demonstrations towards learning an optimal robot policy. In this thesis, we categorize imperfect demonstrations into: i) suboptimal demonstrations, ii) cross-domain demonstrations, and iii) infeasible demonstrations. Suboptimal demonstrations often contain non-optimal sequence of states and actions. For example when reaching an object, the robot might take a longer path towards the goal. Cross-domain demonstrations are collected from agents with different morphologies or dynamics, but such demonstrations can still have correspondence to behaviors of the target agent. Finally, infeasible demonstrations are drawn from other agents that might not have any correspondence

to the target agent.

Prior works in learning from imperfect demonstrations only focus on one of these categories of imperfect demonstrations. In this thesis, we comprehensively address the problem of learning from imperfect demonstrations: We formalize the different categories of imperfect demonstrations and introduce a set of robot learning algorithms that tackle each category when learning from these demonstrations. We will further discuss under what assumptions each of our methods should be used with imperfect demonstrations. We conduct experiments in a number of robotics manipulation tasks in simulation and real to demonstrate the developed algorithms.

Acknowledgements

First and foremost, I would like to thank my advisor Dorsa Sadigh. Though I had no experience in robotics when I joined Dorsa's lab, she guided me to explore the field with her broad knowledge and great passion and addressed all of my questions with patience. Under her supervision, I quickly adapted myself into the robotics research and started to conduct research and solve new problem in robotics. I would never be who I am now without her support and it is my honor to be her Ph.D. student.

Many of my works in graduate school have been in close collaboration with other advisors. Yanan Sui not only gives academic advice in our collaboration but also provided helpful suggestions for finding jobs. Erdem Biyik has always been a role model with his dedication and time management skills. I also want to thank Juan Carlos Niebles, Stefano Ermon, Aditya Grover, Amir Zamir and Leonidas Guibas for their supervision in the collaboration. I would also like to thank my internship supervisors and collaborators at Waymo: Yin Zhou and Drago Anguelov for guiding me on the 3D point cloud perception of autonomous driving vehicles. It has also been an honor for me to collaborate and be co-authors with great mentors from industry: Adrien Gaidon, Guy Rosman, and Allan Raventos. Also, I appreciate the guidance from Chelsea Finn, Karen Liu, Monroe Kennedy and Guy Rosman on my thesis.

I would also like to thank all my amazing co-authors who have been one of the main sources of learning and incredibly fun to work with. I have several works collaborating with Zihan Wang, who is already a Ph.D. student in University of Washington. Great ideas emerged when chatting with him and it is a great pleasure to work with him, who is always energetic and hard-working. I also had the privilege to work with Woodrow Z Wang, Minae Kwon, Yilun Hao, Mengxi Li, Songyuan Zhang, Tong Xiao, Nancy

Wang and Yuchen Cui.

My first research experience was at Tsinghua University during my undergraduate studies with Mingsheng Long in 2015. He introduced me into the artificial intelligence research and helped me to start my first project on artificial intelligence research. Under his careful supervision, I completed several great works in my undergraduate years, only with which could I apply for the Ph.D. program. His continuous support to this date has been invaluable and changed my life. I also want to thank Qixing Huang. Working in his lab helped me to adapt to research and life in US and prepared me for the Ph.D. program in US.

Although the life in pademics is difficult these years, I never feel lonely with my friends Lantao Yu, Tianhe Yu, Yilin Zhu, Boxiao Pan and Hongyu Ren. We have fun playing video games, board games and watching shows since going out is limited in the past two years.

Finally, I would like to thank my family for all their love and support. My parents Canhua Cao and Liping Zhang have always believed and boosted me when I was frustrated. They brought up me in a way that made this thesis possible: they always taught me to always do things with passion and responsibility once you determined to do them. Also, they told me to not always put myself under pressure and relax sometimes. I am thankful to all members of my family. They gave me a happy childhood and taught me with the helpful life philosophy, which always guides me in my growth.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, machine learning techniques have been integrated into robot learning and enable researchers to develop robots to achieve various tasks. To learn a robotics policy, one of the most common and useful paradigms is imitation learning. Imitation learning aims to learn a policy by imitating the behavior in the pre-collected demonstrations.

However, standard imitation learning algorithms often assume access to optimal expert demonstrations collected directly on the target robot [2, 54, 20, 9]. These algorithms usually require large amounts of demonstrations while it is expensive to collect a large number of optimal expert demonstrations on the target robot due to factors such as human bounded rationality [43] or difficulty of controlling robots with different or high degrees of freedom [28]. Thus, this assumption limits the practical use of imitation learning. As an alternative choice, we usually have access to a plethora of imperfect demonstrations — demonstrations that can range from random noise or failures to expert or even optimal demonstrations, and demonstrations that are collected from other agents with different dynamics, different embodiments, body schema, or joint or rigid body structures. For example, we may collect demonstrations from people with different experience and as shown in [29], non-experts do not follow the optimal trajectories and show suboptimal behavior.

Standard imitation learning copies the behavior of the imperfect demonstrations, which learns an imperfect policy. The approach in this thesis aims to enable robots to

learn from such imperfect demonstrations.

To learn from imperfect demonstrations, we need to first understand what is the composition of imperfect demonstrations. How do different components of imperfect demonstrations influence imitation learning? We need to clearly discriminate which component contains correct trajectories to learn from and which component contains suboptimal or unlearnable trajectories that should be discarded. Based on the above understanding, we need to develop algorithms to tackle each component of imperfect demonstrations. In this thesis, we attempt to answer the above questions.

1.1 Thesis Approach

To learn a policy for a new robot, imperfect demonstrations exist both in off-the-shelf datasets and in the process of collecting new demonstrations. Leveraging these data relieves the stress of insufficient optimal demonstrations and also enables learning a better policy. Our approach divides the imperfect demonstrations into three components: (1) suboptimal demonstrations showing suboptimal behavior but are collected on the target robot; (2) cross-domain demonstrations which are collected on other agents but still have correspondence to the target robot; (3) infeasible demonstrations which are collected on other agents but cannot be achieved by the target robot (thus without correspondence). We show examples and analyze the challenges of each component of imperfect demonstrations and then develop algorithms to tackle each component in the corresponding chapters.

Though recent works trying to address learning from imperfect demonstrations [47, 7, 26], they usually focus on only one component in the imperfect demonstrations. In this thesis, we aim to comprehensively analyze and solve the problem of learning from all kinds of imperfect demonstrations, which is crucial to the practical use of imitation learning in real-world applications.

1.2 Contributions

The thesis makes the following contributions.

The goal of this thesis is to develop learning algorithms to learn from imperfect demonstrations including suboptimal demonstrations, cross-domain demonstrations, and infeasible demonstrations.

Confidence Re-weighting for Improving Learning from Suboptimal Demonstrations

In chapter 3, we develop confidence measurements to assess the performance of each demonstration trajectory. When conducting imitation learning, we re-weight the demonstrations with the confidence measurement and enable the target robot to learn from optimal or near-optimal demonstrations and ignore suboptimal or even noisy demonstrations. We design three methods to learn confidence measurement. The first method learns confidence based on the rectified reward value. The second method, Confidence-Aware Imitation Learning, learns confidence with a bi-level optimization framework using the ranking data over demonstrations. The third method, adversarial confidence transfer, leverages confidence annotations from other agents and transfers the confidence function from other agents to the target robot. The three methods learn the confidence measurement under different types of annotations.

Learning from Cross-Domain Demonstrations via Correspondence Learning

In chapter 4, we build correspondence between different agents to enable the target robot to learn from cross-domain demonstrations from other agents. We introduce weak supervision as the abstraction of states and actions to learn correspondence and reduce the annotation efforts. Experimental results show that with the correspondence built by weakly-supervised correspondence learning, the target robot can directly learn from the optimal demonstrations from other agents.

Learning Feasibility to Re-Weight Infeasible Demonstrations

Not all demonstrations from other agents have correspondence to the target robot. Some of the demonstrations cannot be achieved by the target robot, which is coined infeasible demonstrations. In chapter 5, we develop a feasibility measurement to evaluate how each demonstration trajectory is likely to be feasible in the target environment. We re-weight the demonstrations with the feasibility measurement and enforce the robots to learn more from demonstrations that are more likely to be feasible. Specifically, in the first method, we develop a feasibility measurement with

the inverse dynamics function. In the second method, we design a feasibility-Markov decision process to release the assumption of the inverse dynamics function and learn the feasibility measurement.

To verify the performance of our approaches, for all three kinds of demonstrations, we conduct experiments on MuJoCo, simulated robot arms, and a real robot arm to show that our approaches learn the optimal policy under different kinds of imperfect demonstrations.

1.3 Thesis Organization

In chapter 2, we introduce the basic problem setting for imitation learning, which gives an introductory overview of imitation learning to readers without a robotics background and defines the variables and notations that will be used in the following chapters. We also give an overview of the three components of the imperfect demonstrations to help readers have a basic idea of imperfect demonstrations before going deep into each component.

In chapter 3, we explain how suboptimal demonstrations influence imitation learning and propose a solution that learns a confidence measurement to assess how likely the trajectory is optimal (Section 3.1). To achieve this solution, we develop a confidence measurement based on the expected return of each trajectory (Section 3.2). Due to the influence of the initial state on the expected return, we design a rectifying function to rectify the return to the confidence measurement. Since the reward function is difficult to obtain for many environments, we propose a confidence-aware imitation learning framework with a bi-level optimization algorithm and resort to the easier-to-access ranking data as the annotation (Section 3.3). We also show the convergence bound for the algorithm. Finally, we consider a scenario where no annotation is available for the target robot and we need to leverage the confidence annotation from other agents. We develop an adversarial confidence transfer algorithm to transfer the confidence function across agents (Section 3.4).

In chapter 4, we first define the correspondence between robots, which is the key to learning from cross-domain demonstrations (Section 4.1). With the correspondence

definition, we introduce the definition and the practical value of cross-domain demonstrations. Then we briefly introduce the Dynamics Cycle-Consistency (DCC) [52], which is a correspondence learning work on which our method is built. Noticing that unsupervised correspondence learning methods like DCC still suffers from a performance gap to the supervised correspondence learning methods, we propose weak supervision as the similarity over abstraction of states and actions, which are easy to annotate but significantly improve the correspondence learning performance (Section 4.4). We also introduce multi-step dynamics cycle-consistency to enforce the long-term alignment between agents.

In chapter 5, we focus on infeasible demonstrations, which exist in the demonstrations from other agents but have no correspondence to the target robot (Section 5.1). This means that the target robot cannot follow these demonstrations. Thus, we propose to use a feasibility measurement to re-weight demonstrations to filter out those far from feasible demonstrations. We utilize the inverse dynamics function to reproduce trajectories and define the feasibility based on how likely each demonstration trajectory can be reproduced by the target robot (Section 5.2). However, since the inverse dynamics function is not always available for a robot environment, we remove this assumption and develop a feasibility-MDP to learn the feasibility measurement, which only requires the target environment without the inverse dynamics function.

In chapter 6, we summarize the contribution of the thesis and discuss the limitations and open challenges not resolved in the thesis. Additional proofs, implementation details, and experimental results are presented in the Appendix.

Chapter 2

Background

In this chapter, we introduce the basic problem setting of learning from imperfect demonstrations and the background of standard imitation learning algorithms. Note that in this chapter, we define the general notations that will be used by all of our algorithms in Chapter 3, 4 and 5. For specific notations used in each algorithm, we define them in the corresponding section respectively.

2.1 Problem Setting

We model the robot of interest with its environment as a standard Markov decision process (MDP): $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \rho_0, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability, which indicates the probability distribution of the next state given a state and an action. ρ_0 is the initial state distribution, which means that Each episode in the system starts with an initial state $s_0 \in \mathcal{S}$ drawn from ρ_0. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which reflects how good the agent performs at each step. γ is the discount factor.

A policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines a probability distribution over the space of actions in a given state. An optimal policy π^* maximizes the expected return $\eta_\pi = \mathbb{E}_{\xi \sim \pi}[G_t(\xi)] = \mathbb{E}_{s_0 \sim \rho_0, \pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)\right]$, where t indicates the time step.

We formalize the problem to learn the optimal policy π^* for the MDP $\mathcal{M}$ as an imitation learning problem: We are given a set of demonstration trajectories

$\Xi = \{\xi_1, \ldots, \xi_D\}$ collected by a demonstrator d following policy π_d. Here each trajectory is a sequence of state-action pairs $\xi = \{s_0, a_1, s_1, a_2, \ldots, a_N, s_N\}$, we aim to learn a policy π^* that best matches the demonstration trajectories. Similarly, we can define the expected return of a trajectory as $\eta_\xi = \sum_{(s_t,a_t)\in\xi} \gamma^t \mathcal{R}(s_t, a_t)$. Note that we focus on the offline imitation setting as employed in [20, 48], where a set of demonstrations are provided ahead of time instead of gradually incrementing our dataset as in online imitation learning [25].

2.2 Imitation Learning from Expert Demonstrations

In the standard imitation learning paradigm, the demonstrations are assumed to be drawn from the expert policy $\pi_d = \pi^*$, *i.e.,* , a policy that optimizes the expected return of the MDP $\mathcal{M}$ [37, 1, 21, 13]. With the expert demonstrations, standard imitation learning learns a policy π parameterized by θ_π to imitate the behavior in the demonstrations. As introduced later, we will build our approaches based on the standard imitation learning algorithms. Thus, in this section, we will briefly review the standard imitation learning algorithms we will use.

Behavior cloning (BC) is the most basic algorithm of imitation learning, which models imitation learning as a supervised learning problem [36]. At a given state, BC directly minimizes the difference between the action predicted by the policy and the ground truth action in the demonstrations. The loss is shown as follows:

$$\mathcal{L}_{\mathrm{BC}}(\theta_\pi) = \mathbb{E}_{(s,a)\in\Xi} \|\pi(s) - a\|. \tag{2.1}$$

Here we use the L2-loss as the loss to penalize the difference.

Generative adversarial imitation learning (GAIL) is another imitation learning algorithm following a different logic from BC [20]. Instead of penalizing the action difference at each state, it minimizes the divergence of the occupancy measure of state-action pairs of the policy and the demonstrations. Specifically, GAIL consists of a discriminator D parameterized by θ_D and the learned policy π. GAIL learns D a discriminator to discriminate whether a state-action pair is from the policy or the

demonstrations while enforcing the policy to generate state-action pairs to confuse the discriminator. Such as adversarial game makes the policy generate the same state-action distribution as the demonstrations. The loss for the discriminator is:

$$\mathcal{L}_{\text{GAIL}}(\theta_D) = \mathbb{E}_{(s,a)\in\pi}[-\log D(s,a)] + \mathbb{E}_{(s,a)\in\Xi}[-\log(1 - D(s,a))]. \tag{2.2}$$

In addition, the loss for the policy can be written as:

$$\mathcal{L}_{\text{GAIL}}(\theta_\pi) = \mathbb{E}_{(s,a)\in\Xi}[\log D(s,a)] - \lambda H(\pi), \tag{2.3}$$

where H refers to the entropy of the policy π and λ is the trade-off. We may use typical policy optimization methods such as TRPO [38] to update the policy using $-\mathcal{L}_{\text{GAIL}}(\theta_\pi)$ as the reward function.

Adversarial Inverse Reinforcement Learning (AIRL) follows GAIL and also minimizes the divergence of the occupancy measure. However, in addition to learning the optimal policy, AIRL can recover the reward function that is robust to changes in dynamics from the discriminator parameters. Therefore, AIRL updates the discriminator similar to GAIL but when updating the policy, AIRL first recovers the reward function as:

$$\mathcal{R}(s,a) = \log D(s,a) - \log(1 - D(s,a)). \tag{2.4}$$

Using the recovered reward, AIRL updates the policy using any policy optimization methods such as TRPO [38].

In this thesis, we relax this assumption so that the demonstrations may contain non-expert demonstrations drawn from policies other than π^* or demonstrations collected from other environments. We categorize the demonstrations into suboptimal demonstrations, cross-domain demonstrations, and infeasible demonstrations and will introduce the challenges and solutions to dealing with these demonstrations in the next three chapters respectively.

Chapter 3

Confidence Learning from Suboptimal Demonstrations

3.1 Suboptimal Demonstrations

When collecting demonstrations on robots, instead of purely collecting optimal demonstrations, we usually have access to a mixture of demonstrations with varying optimality ranging from random trajectories to optimal demonstrations. This can be caused by various factors such as bounded human rationality [43] or difficulty of controlling robots with different or high degrees of freedom [28].

The suboptimal demonstrations introduce a new set of challenges. First, one needs to select useful demonstrations beyond the optimal ones. We are interested in settings where we do not have sufficient expert demonstrations in the mixture so we have to rely on learning from sub-optimal demonstrations that can still be successful at parts of the task. Second, we need to be able to filter the negative effects of useless or even malicious demonstrations, *e.g.*, demonstrations that implicitly fail the tasks.

To address the above challenges, we propose to use a measure of *confidence* to indicate the likelihood that a demonstration is optimal. The confidence $w(s, a) \in [0, 1]$ is defined on each state-action pair. The confidence score can provide a fine-grained characterization of each demonstration's optimality. For example, it can differentiate

between near-optimal demonstrations and adversarial ones. By reweighting demonstrations with a confidence score, we can simultaneously learn from useful but sub-optimal demonstrations while avoiding the negative effects of malicious ones. So our problem reduces to learning an accurate confidence measure for demonstrations. Previous work learns the confidence from manually annotated demonstrations [47], which are difficult to obtain and might contain bias—For example, a conservative and careful demonstrator may assign lower confidence compared to an optimistic demonstrator to the same demonstration. In this thesis, we propose two approaches to learning confidence measurement and further learning the optimal policy from suboptimal demonstrations. The first approach leverages the reward value as the confidence measurement while the second approach learns the confidence from some evaluation data, especially the ranking data.

3.2 Reward-based Confidence Measurement

We would like to define a confidence score that not only measures the instance reward received by the state transition but also the influence of a given state transition on the future state transitions, e.g., whether the given state transition leads to higher expected return trajectories. Thus, we use the expected return η_ξ received by each trajectory ξ as the confidence, which can estimate how optimal a trajectory is.

3.2.1 Effects of Initial States on the Expected Return

However, the expected return suffers from some problems. Is a trajectory with a higher expected return more useful for the policy learning of the target agent? If the trajectories are starting from the same initial state, this claim is correct, because a higher expected return means we have a more optimal path from this initial state. However, when the trajectories start from different initial states, the claim may not be correct. Imagine the case shown in Fig. 3.1, where the car navigates to the goal from different initial states. If the car is close to the goal, e.g., one step from the goal, the navigation seems easier while if the car is far away from the goal, the navigation

is more difficult. The optimal trajectory for the close initial location receives a much higher reward than the optimal trajectory for the far one and thus is assigned with much higher confidence if we only use the expected return as our measure of confidence. However, both trajectories (starting close or far) are important for the target agent since they give the agent guidance on how to navigate to the goal from different initial locations. In fact, we should assign similar confidence to both trajectories. Therefore, naïve confidence scores based on the expected return may negatively influence the diversity of the demonstrations.

The Confidence Score. To address this problem, we rectify the expected return to make it conditioned on the initial states. Our key idea is that if demonstration trajectories are from the same initial states, we should learn from the one with a higher expected return. Based on this idea, we define a function $f_{\text{rec}} : \mathcal{S} \to \mathbb{R}$. The input is an initial state s_0 and the output is the highest expected return of a demonstration starting from s_0. We then firstly define the confidence of a trajectory ξ as:

$$w(\xi) = \exp\left(-\frac{(\eta_\xi - f_{\text{rec}}(s_0))^2}{2\sigma^2}\right). \quad (3.1)$$

We compute the distance between the expected return of each trajectory $\xi = \{s_0, s_1, \ldots, s_N\}$ and the best expected return the demonstration can achieve by starting from s_0, and this can determine the optimality score assigned to a trajectory. However, the distance $\eta_\xi - f_{\text{rec}}(s_0)$ is smaller than 0 but the lower bound depends on the reward function, which may have highly different scales from the feasibility. So we normalize this score in the range of $[0, 1]$. We use a Gaussian function for normalization instead of the

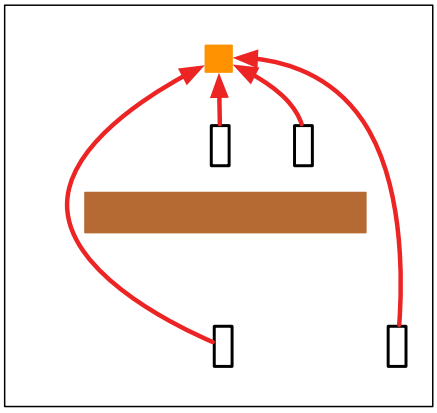


Figure 3.1: The optimal trajectories from different initial states with the same dynamics under the navigation task to reach the orange goal as fast as possible. So the car receives a positive reward for reaching the goal but is punished with a negative reward for every time step. The expected returns received by the optimal trajectories are different for different initial states.

widely-used min-max normalization [49],
which can more effectively filter out extremely low return trajectories. Here, we only define the confidence score for each trajectory and we will introduce later that we assign the same confidence score for each state-action pair in the same trajectory.

Learning the Rectifying Function. The key to the designed confidence measurement is estimating f_{rec} – the highest expected return for an initial state. Given that we only have access to a small number of trajectories, where the initial states and the expected returns are known, we need a set of assumptions for generating training data to learn f_{rec}. For example, if the initial states of two trajectories ξ_1 and ξ_2 are very close, the policy learned from one trajectory can also perform on the other and achieve a similar expected return. So we may assume that f_{rec} of the two close initial states should be the same or at least similar. A realistic assumption for f_{rec} is the neighborhood property: $\exists \delta > 0, \forall \xi = \{s_0, s_1, \ldots, s_N\}$, we set the highest expected return for s_0 as:

$$f_{\text{rec}}(s_0) = \max_{\xi' \in \Xi, |s'_0 - s_0| < \delta, w_f(\xi) > 0} \eta_{\xi'}, \tag{3.2}$$

where s'_0 is the initial state of ξ' and Ξ is the set of all demonstrations. We set the highest expected return for an initial state as the highest expected return of neighboring initial states of feasible trajectories ($w_f(\xi) > 0$), because the rectify function will be influenced by infeasible trajectories with high expected returns otherwise. Using this rectifying function f_{rec}, we can compute the rectified confidence, which induces more diverse optimal demonstrations.

We emphasize that we do not assume access to the reward function, and only assume access to the expected return of a trajectory similar to prior work that leverages a given confidence measure for each demonstration [48].

3.2.2 Algorithm

Using the defined notions of confidence, we now can compute the final score of how useful each state transition is for learning a policy for the target agent. We then

re-weigh each state transition in the demonstration by the score $w(\xi)$. The weighting-based work for learning from imperfect demonstrations often directly incorporates the weight in the imitation loss [48]. However, such re-weighing can suffer from numerical issues, when the weight of some state transitions is too large. Instead, for more stable training, we define a distribution p_w over state transitions based on the score $w(\xi)$. For every state-action pair in a demonstration from the dataset, we assign a weight w_i (for the i-th state-action pair) equal to the weight assigned to the trajectory $w(\xi)$, i.e., $\forall i \quad w_i = w(\xi)$. We compute a probability distribution over the state transitions as $p_{w_i} = \frac{w_i}{\sum_i w_i}$. Using the sampling distribution p_w, we can embed our method into any imitation learning algorithm to enable it to learn from imperfect demonstrations of various dynamics.

As an example, we derive the loss for generative adversarial imitation learning (GAIL) [20]. We modify the loss in Eqn (3.3) as follows:

$$\mathcal{L}_{\text{GAIL}}(\theta_D) = \mathbb{E}_{(s,a)\in\pi}[-\log D(s,a)] + \mathbb{E}_{(s,a)\in p_w}[-\log(1 - D(s,a))]. \tag{3.3}$$

. The loss to update the policy in Eqn (2.3) does not change.

Note that given a trajectory, we assign equal scores to each state transition within the trajectory, which fails to emphasize the most important transitions. However, this is not a problem as the key transitions repeatedly appear in high-score trajectories, and every time they appear, their sampling probability p_w accumulates. This ensures the key transitions are considered.

3.3 Confidence-Aware Imitation Learning

However, the above confidence value relies on the reward value, which requires a well-defined reward function to annotate. Such a reward function does not exist in many environments. Thus, in our method, we aim to relax the strong assumption of the reward function. We propose a general framework to learn from demonstrations with varying optimality that jointly learns the confidence score and a well-performing policy. Our approach, Confidence-Aware Imitation Learning (CAIL) learns a well-performing

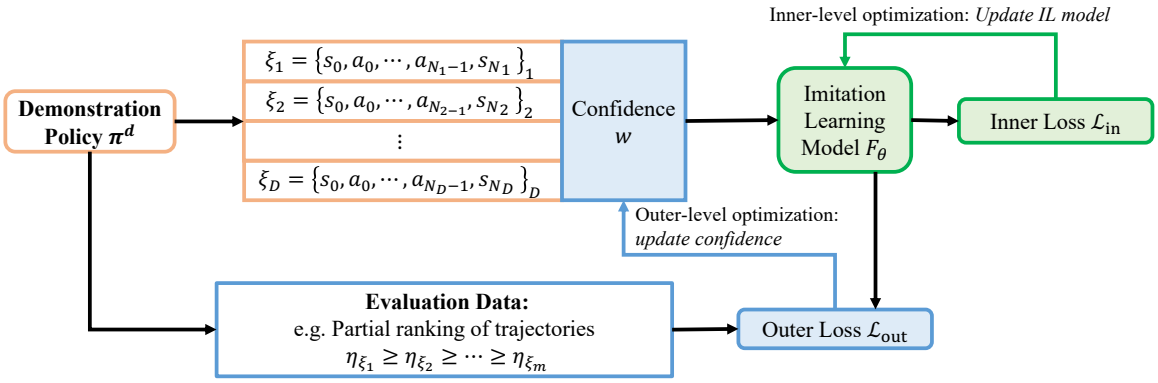


Figure 3.2: Confidence-Aware Imitation Learning. The demonstrations are shown in the orange box drawn from the demonstration policy: $\xi_1, \ldots, \xi_D \sim \pi_d$. The confidence learning component and the outer loss are shown in blue. The confidence w reweights the distribution of state-action pairs in the demonstration set, and then the imitation learning model F_θ learns a well-performing policy and new parameters θ with the confidence-reweighted distribution using the inner loss (imitation loss) shown in green. Next iteration, the updated F_θ generates new trajectories that are then evaluated by the outer loss and potentially other evaluation data (*e.g.* partial ranking of trajectories) to update confidence.

policy from confidence-reweighted demonstrations while using an outer loss to track the performance of our model and to learn the confidence. We only rely on the ability to evaluate the performance of imitation learning, which is much easier than accessing the reward function.

Given the demonstration set Ξ, we need to assess our *confidence* in each demonstration. To achieve learning confidence over this mixture of demonstrations, we rely on the ability to evaluate the performance of imitation learning. This can be achieved by using an evaluation loss trained on *evaluation data*, Ξ_E (as shown in Fig. 3.2). In our implementation, we rely on a small number of rankings between trajectories as our evaluation data: $\Xi_E = \eta_{\xi_1} \geq \cdots \geq \eta_{\xi_m}$. To summarize, our framework takes a set of demonstrations with varying optimality Ξ as well as a limited amount of evaluation data Ξ_E along with an evaluation loss to find a well-performing policy. Note that unlike prior work [47], we do not assume that optimal demonstrations always exist in the demonstration set, and CAIL can still extract useful information from Ξ while avoiding the negative effects of non-optimal demonstrations.

In our framework, we adopt an imitation learning algorithm with a model F_θ parameterized by θ and a corresponding imitation learning loss $\mathcal{L}_{\text{in}}$, which we refer to as inner loss (as shown in Figure 3.2). We assign each state-action pair a confidence value indicating the likelihood of the state-action pair appearing in the well-performing policy. The confidence can be defined as a function mapping from a state-action pair to a scalar value $w : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We aim to find the optimal confidence assignments w^* to reweight state-action pairs within the demonstrations. We then conduct imitation learning from the reweighted demonstrations using the inner imitation loss $\mathcal{L}_{\text{in}}$ to learn a well-performing policy. Here, we first define the optimal confidence w^* and describe how to learn it automatically.

Defining the Optimal Confidence. We define the distribution of state-action pairs visited by a policy π based on the occupancy measure $\rho_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: $\rho_\pi(s, a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi)$, which can be explained as the un-normalized distribution of state transitions that an agent encounters when navigating the environment with the policy π. We can normalize the occupancy measure to form the state-action distribution: $p_\pi(s, a) = \frac{\rho_\pi(s,a)}{\sum_{s,a} \rho_\pi(s,a)}$. Recall that π_d is the policy that the demonstrations are derived from, which can potentially be a mixture of different expert, suboptimal, or even malicious policies. We reweight the state-action distribution of the demonstrations to derive a new state-action distribution, which corresponds to another policy π_{new}: $p_{\pi_{\text{new}}}(s, a) = w(s, a)p_{\pi_d}(s, a)$. Our goal is to find the optimal confidence w^* that ensures the derived policy π_{new} maximizes the expected return:

$$w^*(s, a) = \arg\max_w \eta_{\pi_{\text{new}}}. \tag{3.4}$$

With such $w^*(s, a)$, we can conduct imitation learning from the reweighted demonstrations to maximize the expected return with the provided demonstrations.

Learning the Confidence. We will learn an estimate of the confidence score w without access to any annotations of the ground-truth values based on optimizing two loss functions: The inner loss and the outer loss. The inner loss $\mathcal{L}_{\text{in}}$ is accompanied with the imitation learning algorithm encouraging imitation, while the outer loss $\mathcal{L}_{\text{out}}$ captures the quality of imitation learning, and thus optimizing it finds the confidence

value that maximizes the performance of the imitation learning algorithm.

Specifically, we first learn the imitation learning model parameters θ^* that minimize the inner loss:

$$\theta^*(w) = \arg\min_{\theta} \mathbb{E}_{(s,a)\sim w(s,a)p_{\pi_d(s,a)}} \mathcal{L}_{\text{in}}(s, a; \theta, w) \tag{3.5}$$

We note that the inner loss $\mathcal{L}_{\text{in}}(s, a; \theta, w$ refers to settings where (s, a) is sampled from the distribution $w(s, a)p_{\pi^d(s,a)}$, and hence implicitly depends on w. Thus we need to find the optimal w^*, which can be estimated by minimizing an outer loss $\mathcal{L}_{\text{out}}$:

$$w^*_{\text{out}} = \arg\min_{w} \mathcal{L}_{\text{out}}(\theta^*(w)). \tag{3.6}$$

This evaluates the performance of the underlying imitation learning algorithm with respect to the reward with limited evaluation data Ξ_E (*e.g.* limited rankings if we select a ranking loss as our choice of $\mathcal{L}_{\text{out}}$; which we will discuss in detail in Sec. 3.3.3).

3.3.1 Optimization of Outer and Inner Loss

We design a bi-level optimization process consisting of an inner-level optimization and an outer-level optimization to simultaneously update the confidence w and the model parameters θ. Within the outer-level optimization, we first pseudo-update the imitation learning parameters to build a connection between w and the optimized parameters θ' with the current w. We then update w to make the induced θ' minimize the outer loss $\mathcal{L}_{\text{out}}$ in Eqn. (3.6). The inner-level optimization is to find the imitation learning model parameters that minimize inner loss $\mathcal{L}_{\text{in}}$ with respect to the confidence w. We introduce the details of the optimization below. We use τ to denote the number of iterations. Note that the losses in this section are all computed based on the expectation over states and actions.

Outer-Level Optimization: Updating w. Let w_τ be the confidence at time τ. Using w_τ, we first pseudo-update the imitation learning parameters θ using gradient descent. Here pseudo-update means that the update aims to compute the gradients of w but does not change the value of θ. Let $\theta'_0 = \theta_\tau$ be the current imitation learning

model parameters, and we update θ' as:

$$\theta'_{t+1} = \theta'_t - \mu \nabla_{\theta'} \mathcal{L}_{\text{in}}(s, a; \theta'_t, w_\tau), \tag{3.7}$$

where μ is the learning rate, t is the pseudo-updating time step for θ'. We will update θ' with respect to the fixed w_τ after convergence of Eqn. (3.7). After updating θ', we now update w using gradient descent with the outer loss $\mathcal{L}_{\text{out}}$ from Eqn. (3.6):

$$w_{\tau+1} = \beta_\tau - \alpha \nabla_w \mathcal{L}_{\text{out}}(\theta'), \tag{3.8}$$

where α is the learning rate for updating w. Intuitively, updating w as in Eqn. (3.8) aims to find the fastest update direction of θ' for decreasing the outer loss $\mathcal{L}_{\text{out}}$. Though we compute gradients of gradients for w here, w is only a one-dimension scalar for each state-action pair and within each iteration of training, we only sample a mini-batch of thousands of state pairs for the update. Thus, within each iteration, the total dimension of w is small, and computing the gradient of the gradient is not costly.

Inner-Level Optimization: Updating θ. With the updated $w_{\tau+1}$, we now will update θ using gradient descent, where we denote the initialization as $\theta_0 = \theta_\tau$.

$$\theta_{t+1} = \theta_t - \mu \nabla_\theta \mathcal{L}_{\text{in}}(s, a; \theta_t, w_{\tau+1}). \tag{3.9}$$

After convergence, we set $\theta_{\tau+1} = \theta$. With the two updates introduced above (outer and inner optimization), we finish one update iteration by setting w_τ to $w_{\tau+1}$ using the converged value from Eqn. (3.8) and θ_τ to $\theta_{\tau+1}$ using the converged value from Eqn. (3.9).

In each iteration of the above optimization—in the steps of pseudo-updating and the steps of updating the imitation learning model—multiple gradient steps are required for convergence, meaning that there is a nested loop of gradient descent algorithms. The nested loop costs quadratic time and is inefficient, especially for deep networks. To further accelerate the optimization, we propose an approximation, which only updates θ once in the pseudo-updating and the updating steps. Therefore, the

new updating rule can be formalized as follows:

$$\begin{aligned}\theta'_{\tau+1} &= \theta_\tau - \mu\nabla_\theta\mathcal{L}_{\text{in}}(s,a;\theta_\tau,w_\tau),\\ w_{\tau+1} &= w_\tau - \alpha\nabla_w\mathcal{L}_{\text{out}}(\theta'_{\tau+1}),\\ \theta_{\tau+1} &= \theta_\tau - \mu\nabla_\theta\mathcal{L}_{\text{in}}(s,a;\theta_\tau,w_{\tau+1}).\end{aligned} \tag{3.10}$$

3.3.2 Theoretical Results

We analyze the convergence of the proposed bi-level optimization algorithm for the CAIL framework and derive the following theorems. We provide detailed proofs of these theorems in the Appendix.

Theorem 1. *(Convergence) Suppose the outer loss $\mathcal{L}_{out}$ is Lipschitz-smooth with constant L, the inequality*

$$\nabla_\theta\mathcal{L}_{out}(\theta_{\tau+1})^\top\nabla_\theta\mathcal{L}_{in}(\theta_\tau,w_{\tau+1}) \geq C||\nabla_\theta\mathcal{L}_{in}(\theta_\tau,w_{\tau+1})||^2 \tag{3.11}$$

holds for a constant $C \geq 0$ in every step τ,[1] *and the learning rate satisfies $\mu \leq \frac{2C}{L}$, then the outer loss decreases along with each iteration: $\mathcal{L}_{out}(\theta_{\tau+1}) \leq \mathcal{L}_{out}(\theta_\tau)$, and the equality holds if $\nabla_w\mathcal{L}_{out}(\theta_\tau) = 0$ or $\theta_{\tau+1} = \theta_\tau$.*

Remark 1. *The inequality in the assumption of Theorem 1 (Eqn. 3.11) indicates that the directions of the gradients of $\mathcal{L}_{out}$ and $\mathcal{L}_{in}$ with respect to θ should be close. Intuitively only when the two gradient directions align, we can decrease the evaluation loss $\mathcal{L}_{out}$ by updating θ with $\mathcal{L}_{in}$.*

Theorem 1 ensures that the confidence and the imitation learning parameters monotonically decrease the outer loss. When the gradient of the outer loss with respect to w is zero, w converges to the optimal confidence that minimizes the outer loss, *i.e.,* , w^* in Eqn. (3.4). With the optimal confidence, we can learn a well-performing policy from more useful demonstrations by reweighting them. Thus, the learned imitation model induces lower outer loss (has higher quality) than the

[1] We remove (s,a) in $\mathcal{L}_{\text{in}}$ for notation simplicity.

imitation learning model learned from the original demonstrations in the dataset without reweighting.

Theorem 2. *(Convergence Rate) Under the assumptions in Theorem 1, let*

$$g(\theta, \beta) = \theta - \mu \nabla_\theta \mathcal{L}_{in}(s, a; \theta, \beta) \tag{3.12}$$

We assume that $\mathcal{L}_{out}(g(\theta, \beta))$ is Lipschitz-smooth w.r.t. β with constant L_1, $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ have σ-bounded gradients, and the norm of $\nabla_\beta \nabla_\theta \mathcal{L}_{in}(\theta; \beta)$ is bounded by σ_1. L is the Lipschitz-smooth constant for $\mathcal{L}_{out}$ w.r.t. $g(\theta, \beta)$ as shown in Theorem 1. Consider the total training steps as T, we set $\alpha = \frac{C_1}{\sqrt{T}}$, for some constant C_1 where $0 < C_1 \leq \frac{2}{L_1}$ and $\mu = \frac{C_2}{T}$ for some constant C_2. Then:

$$\min_{1 \leq \tau \leq T} \mathbb{E}[||\nabla_\beta \mathcal{L}_{out}(\theta_\tau)||^2] \leq O\left(\frac{1}{\sqrt{T}}\right). \tag{3.13}$$

Remark 2. *The assumptions of Theorem 2 are Lipschitz-smoothness and bounded first-order and second-order gradients of $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$, which are satisfied for typical $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ such as the cross-entropy loss of AIRL and the ranking loss in our implementation of CAIL in Section 3.3.3.*

With the bound on the convergence rate, the gradient of the outer loss with respect to β is gradually getting close to 0, which means that β gradually converges to the optimal confidence β^* that minimizes the outer loss if $\mathcal{L}_{\text{out}}$ is convex with respect to β.

3.3.3 An Implementation of CAIL

To implement CAIL, we need to adopt an imitation learning algorithm whose imitation loss will be the inner loss. We also need to design an outer loss on the imitation learning algorithm to evaluate the quality of imitation given some evaluation data $\mathcal{D}_E$ (*e.g.* partial ranking annotations).

Based on the above considerations, as an instance of the implementation of CAIL, we use Adversarial Inverse Reinforcement Learning (AIRL) [13] as our imitation learning model. We use the imitation loss of AIRL as the inner loss and a ranking loss

(based on a partial ranking of trajectories) as the outer loss. AIRL and the ranking loss are compatible since AIRL can induce the reward function from the discriminator within the model, and the ranking loss can penalize the mismatches of the trajectory rankings computed by the induced reward function and the ground-truth rankings from the evaluation data $\mathcal{D}_E$. Furthermore, the implementation only requires the ranking of a subset of demonstrations $\{\xi_i\}_{i=1}^m \subset \Xi$, *i.e.,* , $\mathcal{D}_E = \eta_{\xi_1} \geq \eta_{\xi_2} \geq \cdots \geq \eta_{\xi_m}$, which is much easier to access than the exact confidence value annotations [5, 33] since confidence not only reflects the rankings of different demonstrations but also how much one demonstration is better than the other.

AIRL consists of a policy π serving as a generator parameterized by θ_π as the policy, and a discriminator parameterized by θ_D. The generator and the discriminator are trained in an adversarial manner as in [15] to match the occupancy measures of the policy and the demonstrations. We write the loss $\mathcal{L}_{\text{in}}$ as:

$$\mathcal{L}_{\text{in}}(\theta_D, w) = \mathbb{E}_{(s,a)\sim w(s,a)p_{\pi_d(s,a)}}[-\log D(s,a)] + \mathbb{E}_{(s,a)\sim\pi}[-\log(1 - D(s,a))], \quad (3.14)$$

$$\mathcal{L}_{\text{in}}(\theta_\pi) = \mathbb{E}_{(s,a)\sim\pi}[\log D(s,a) - \log(1 - D(s,a))], \quad (3.15)$$

where $\mathcal{L}_{\text{in}}(\theta_D, w)$ is the inner loss for the discriminator, $\mathcal{L}_{\text{in}}(\theta_\pi)$ is the inner loss for the generator. The discriminator D is learned by minimizing the loss $\mathcal{L}_{\text{in}}(\theta_D, w)$, which aims to discriminates the state-action pair (s, a) drawn from π and the state-action pair (s, a) drawn from $w(s, a)p_{\pi_d(s,a)}$. The generator parameter θ_π is trained to minimize the loss $\mathcal{L}_{\text{in}}(\theta_\pi)$ with policy optimization methods. This enables the learned policy π to generate state-action pairs that are similar to the state transitions in the demonstrations.

For the outer loss, AIRL approximates the reward function by the discriminator parameters, *i.e.,* $\mathcal{R}'_{\theta_D}$. We compute $\eta'_{\xi_i} = \sum_{t=0}^{N} \gamma^t \mathcal{R}'_{\theta_D}(s_t, a_t)$ as the expected return of a trajectory using the reward $\mathcal{R}'_{\theta_D}$. Then we penalize the mismatches of the rankings derived by η'_{ξ_i} and the ground-truth rankings:

$$\mathcal{L}_{\text{out}}(\theta_D) = \sum_i \sum_{j>i} \text{RK}\left[\eta'_{\xi_i}, \eta'_{\xi_j}; \mathbb{I}[\eta_{\xi_i} > \eta_{\xi_j}]\right], \quad (3.16)$$

where $\mathbb{I}[\eta_{\xi_i} > \eta_{\xi_j}]$ is 1 if $\eta_{\xi_i} > \eta_{\xi_j}$ and otherwise is -1. RK is defined as a revised version of the widely-used margin ranking loss with margin as 0:

$$\mathrm{RK}\left[\eta'_{\xi_i}; \eta'_{\xi_j}, \eta_{\xi_i}, \eta_{\xi_j}\right] = \begin{cases} \max(0, -\mathbb{I}[\eta_{\xi_i} > \eta_{\xi_j}](\eta'_{\xi_i} - \eta'_{\xi_j})), & |\eta'_{\xi_i} - \eta'_{\xi_j}| > \epsilon \\ \max(0, \frac{1}{4\epsilon}(\mathbb{I}[\eta_{\xi_i} > \eta_{\xi_j}](\eta'_{\xi_i} - \eta'_{\xi_j}) - \epsilon)^2), & |\eta'_{\xi_i} - \eta'_{\xi_j}| \leq \epsilon \end{cases} \tag{3.17}$$

We revised the original margin ranking loss within a ϵ range around the point of $(\eta'(\xi_i) - \eta'(\xi_j)) = 0$ to make it Lipschitz smooth. If we adopt a small enough ϵ, the functionality of the revised marginal ranking loss is close to the original one. In all the experiments, we use $\epsilon = 10^{-5}$.

3.3.4 Experiments

In this section, we conduct experiments on the implementation of CAIL in Sec. 3.3.3. We verify the efficacy of the CAIL in simulated and real-world environments. We report the results on various compositions of demonstrations with varying optimality. **The code is available on our website**[2]

We conduct experiments in four environments including two MuJoCo environments (Reacher and Ant) [45] in OpenAI Gym [6], one Franka Panda Arm[3] simulation environment, and one real robot environment with a UR5e robot arm[4].

Reacher. In the Reacher environment, the agent is an arm with two links and one joint, and the end effector of the arm is supposed to reach a final location. For each step, the agent is penalized for the energy cost and the distance to the target.

We collect 200 trajectories in total for training, where each trajectory has 50 interaction steps. 5% of the trajectories are annotated with rankings. We collect 5 trajectories for testing. We run the experiment for 5 runs and compute the mean and the standard deviation of the expected return. The average time for each run is 1,291s.

Ant. In the Ant environment, the agent is an ant with four legs and each leg has

[2]https://sites.google.com/view/cail
[3]https://www.franka.de
[4]https://www.universal-robots.com/products/ur5-robot

two links and two joints. Its goal is to move forward in the x-axis direction as fast as possible. For each step, the agent is rewarded for moving fast in the x-axis direction without falling, while it is penalized for the energy cost. If the ant fails to stand, the trajectory will be terminated.

We collect 200 trajectories in total for training, where each trajectory has at most 1000 interaction steps. 5% of the trajectories are annotated with rankings. We collect 5 trajectories for testing. We run the experiment for 5 runs and compute the mean and the standard deviation of the expected return. The average time for each run is 17,140s.

Simulated Robot Arm. In this environment, there is a Franka Panda Arm that is supposed to pick up a bottle, avoid the obstacle, and put the bottle on a target platform. For each step, the agent is penalized for the energy cost and the distance to the target. If the agent drops the bottle or hits the target, it will receive a large negative reward and the trajectory will be terminated. If the agent succeeds to make the bottle stand on the target, the trajectory will be terminated too, so that the arm will no longer receive penalization. The reachable region of the arm is $[0.20, 0.80]$ in x-axis, and $[-0.35, 0.35]$ in y-axis. The initial position of the bottle is sampled in $[0.68, 0.72] \times [-0.05, 0]$ and the initial position of the target is sampled in $[0.28, 0.32] \times [-0.32, -0.28]$. The action space is the velocity of the end-effector, and the maximum velocity is 1 in each direction.

We collect 200 trajectories in total for training, where each trajectory has at most 2000 interaction steps. 5% of the trajectories are annotated with rankings. We collect 5 trajectories for testing. We run the experiment for 5 runs and compute the mean and the standard deviation of the expected return. The average time for each run is 44,731s.

Real Robot Arm. In this environment, we use a real UR5e robot arm in a similar setting as the simulation environment.

We collect 200 trajectories in total for training, where each trajectory has at most 2000 interaction steps. 5% of the trajectories are annotated with rankings. We collect 5 trajectories for testing. We run the experiment for 5 runs and compute the mean and the standard deviation of the expected return. The average time for each run is

141,699s.

Source of Demonstrations

For MuJoCo environments, following the demonstration collecting method in [47], we train a reinforcement learning algorithm and select four intermediate policies as policies with varying optimality and the converged policy as the optimal policy, so that the demonstrations range from worse-than-random ones to near-optimal ones. We draw 20% of demonstrations from each policy. For the RL algorithm, we use SAC [18] for the Reacher environment and PPO [40] for the Ant environment. For the Franka Panda Arm simulation and the real robot environment with UR5e, we hand-craft demonstrations with optimality varying continuously from near-optimal ones to unsuccessful ones to approximate the demonstration collecting process from demonstrators with different levels of expertise.

Reward Design

To evaluate the proposed CAIL and other methods, we use the expected return for all the environments, which is the discounted cumulative reward of a trajectory. For the Reacher and the Ant environments, we use the reward function in their original implementation in Gym[5]. For the Simulated Robot Arm and the Real Robot Arm environments, we define a reward as follows: Assume that the action of the robot arm (the velocity of the end-effector) is a, the distance between the bottle and the target is d, the distance between the bottle's initial position and the target is d_{init}, then at each step, the robot will receive a reward of $-\frac{0.02a}{d_{\text{init}}^2} - 0.05d$. If the robot drops the bottle or the obstacle is moved, the robot will receive a reward of -2000 and the trajectory will be terminated. In the robot arm environments (both simulated and real), we also use the success rate as another metric to evaluate the rate at which the robot arm successfully moves the bottle to the goal area without colliding with the obstacle.

[5]https://github.com/openai/gym

Implementation Details

Baselines. We compare CAIL with the most relevant works in our problem setting including the state-of-the-art standard imitation learning algorithms: GAIL [21], AIRL [13], imitation learning from suboptimal demonstration methods including two confidence-based methods, 2IWIL and IC-GAIL [47], and three ranking-based methods, T-REX [7], D-REX [8], and SSRR [11]. GAIL and AIRL learn directly from the mixture of optimal and non-optimal demonstrations. T-REX needs demonstrations paired with rankings, so we provide the same number of rankings as our approach. For D-REX and SSRR, we further generate rankings by disturbing demonstrations as done in their papers. For 2IWIL and IC-GAIL—that need a subset of demonstrations labeled with confidence—we label the subset of ranked demonstrations with evenly-spaced confidence, *i.e.,* , the highest expected return as confidence 1, and the lowest expected return as 0. This is a reasonable approximation of the confidence score with no prior knowledge available. For a fair comparison, we re-implement 2IWIL with AIRL as its backbone imitation learning method. For the RL algorithm in T-REX, D-REX, and SSRR, we also use PPO. DPS [33] requires interactively collecting demonstrations and the approach in Cao *et al.* [10] requires the ground truth reward of demonstrations, which are both not implementable under the assumptions in our setting, so we do not include them.

We implement CAIL based on a PPO-based AIRL[6]. The actor and the critic are neural networks with two hidden layers with size 64 and Tanh as the activation function, and the discriminator is a neural network with two hidden layers with size 100 and ReLU as the activation function. We use ADAM to update the imitation learning model and the Stochastic Gradient Descent method (SGD) to update the confidence. We implement our method in the PyTorch framework [34]. We train each algorithm 10 times with different random seeds and record how the expected return and the standard deviation vary during training. While testing the return, we run the algorithm for 100

[6]We use the AIRL implementation in this repository: `https://github.com/toshikwa/gail-airl-ppo.pytorch`. The repository uses a slightly different implementation from the original paper, which uses $\log D(s,a)$ as the reward to update the policy instead of $\mathcal{R}(s,a)$ in Eqn. (2.4). This achieves a more stable performance than the original AIRL in the repository

episodes. While implementing Eqn. (11), we normalize β so that their mean value is 1, i.e. for the first part of Eqn. (11), we use $\sum_{(s,a)\in\Xi}\left(-\frac{n\beta(s,a)}{\sum_{(s,a)\in\Xi}\beta(s,a)}\right)\log(D(s,a))$, where n is the number of state-action pairs in Ξ. All the experiments in all environments are run on one Intel(R) Xeon(R) Gold 6244 CPU @ 3.60GHz with 10G memory.

Results

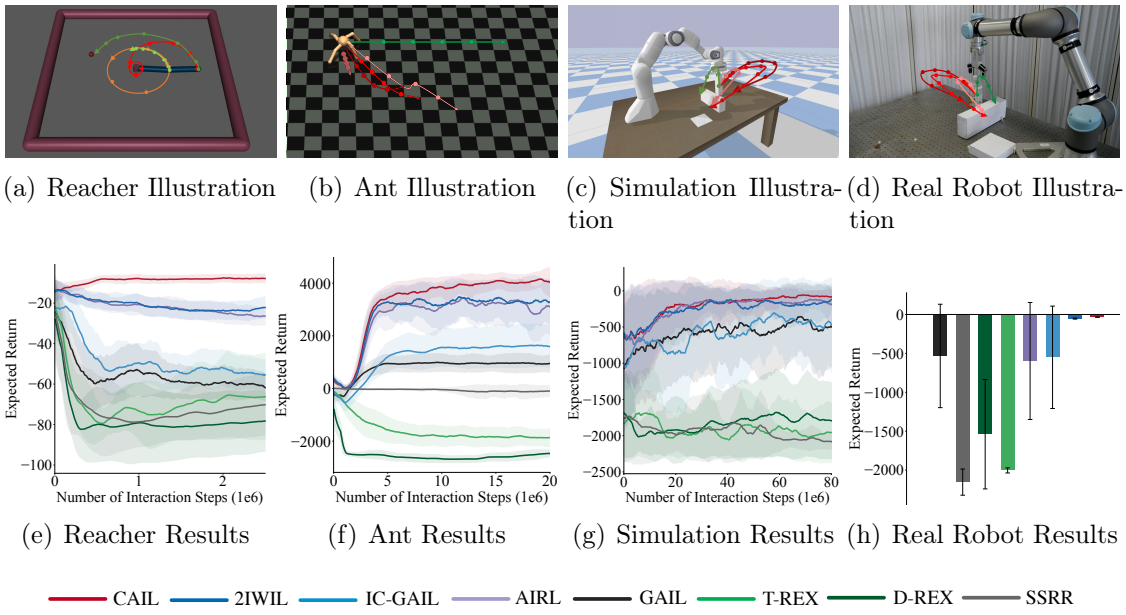
Reacher and Ant. In the Reacher, the end effector of the arm is supposed to reach a final location. Figure 3.3(a) shows the optimal trajectories of the joint and the end effector in green, which illustrates the policy reaching the location with the minimum energy cost, and the trajectories with lower optimality in red and orange, where the agent just spins around the center and wastes energy without reaching the target. We collect 200 trajectories in total, where each trajectory has 50 interaction steps.

In Ant, the agent has four legs, each with two links and two joints. Its goal is to move in the x-axis direction as fast as possible. Figure 3.3(b) illustrates the demonstrated trajectories, where green shows the optimal one, and red shows suboptimal trajectories (darker colors show lower optimality). In optimal demonstrations, the agent moves quickly along the x-axis, while in suboptimal ones, it moves slowly in other directions. We collect trajectories with 200,000 interaction steps in total.

As shown in Figure 3.3(e) and 3.3(f), CAIL achieves the highest expected return compared to the other methods and experiences fast convergence. For Reacher, the p-value[7] between CAIL and the closest baseline method, T-REX, is 5.4054×10^{-6} (statistically significant). For Ant, the p-value between CAIL and the closest baseline method, 2IWIL, is 0.1405. CAIL outperforms standard imitation learning methods, GAIL and AIRL, because CAIL selects more useful demonstrations, and avoids the negative influence of harmful demonstrations. We observe that 2IWIL and IC-GAIL do not perform well because neighboring demonstrations in a given ranking are not guaranteed to have the same distance in terms of confidence score and thus the evenly-spaced confidence values derived from rankings are likely not accurate. All the ranking-based methods do not perform well. For T-REX, the potential reason can be

[7]All p-values are computed by the student's t-test and the null hypothesis is the performance of CAIL is equal to or smaller than the baseline methods.

that the rankings of a subset of demonstrations are not enough to learn a generalizable reward function covering states. For D-REX and SSRR, the automatically generated rankings can be incorrect since we also have unsuccessful demonstrations— which can at times be worse than random actions—and perturbing such demonstrations is not guaranteed to produce demonstrations that imply rankings.

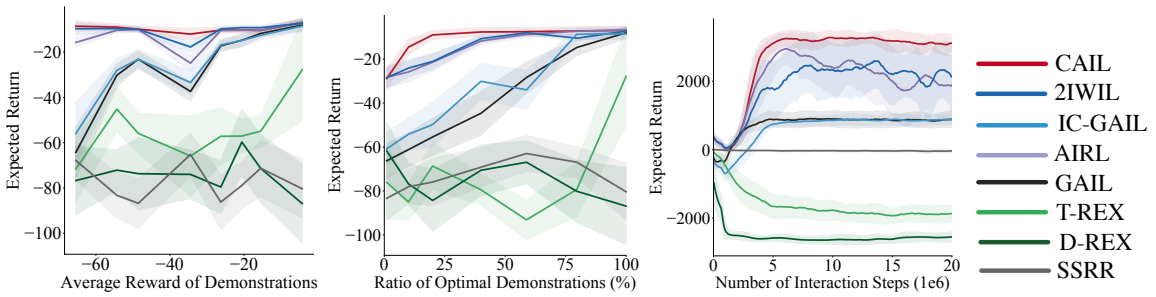


(a) Reacher Illustration (b) Ant Illustration (c) Simulation Illustration (d) Real Robot Illustration

(e) Reacher Results (f) Ant Results (g) Simulation Results (h) Real Robot Results

Figure 3.3: (a) Reacher, (b) Ant, (c) Simulated Panda Robot Arm, (d) Real UR5e Robot Arm. In (a-d), the green trajectories indicate the optimal demonstrations, while the red and orange trajectories indicate demonstrations with varying optimality. (e-g) The expected return with respect to the number of interaction steps. (h) The expected return of the converged policies for UR5e Robot Arm.

Robot Arm. We further conduct experiments in more realistic environments: a simulated Franka Panda Arm and a real UR5e robot arm. As shown in Figure 3.3(c) and 3.3(d), we design a task to let the robot arm pick up a bottle, avoid the obstacle, and put the bottle on a target platform. In the optimal demonstrations in green, the arm takes the shortest path to avoid the obstacle, and puts the bottle on the target, while in suboptimal ones in red (where, similar to before, the brightness of the trajectories indicates their optimality), the arm detours, does not reach the target, and even at times collides with the obstacle. The suboptimal demonstrations represent



(a) Varying Optimality (b) Varying Ratio (c) Pure Suboptimal

Figure 3.4: (a-b) The Expected Return with respect to different optimality of demonstrations in the Reacher environment, where the different optimality are created by varying the optimality of non-optimal demonstrations and varying the ratio of optimal demonstrations. (c) Results for learning from only non-optimal demonstrations in the Ant environment.

a wide range of optimality from near-optimal ones (small detours) to adversarial ones (colliding). We vary the initial position of the robot end-effector and the goal position within an initial area and goal area respectively. For both simulated and real robot environments, we collect trajectories with 200,000 interaction steps in total.

As shown in Figure 3.3(g) and 3.3(h), CAIL outperforms other methods in expected return in both the simulated and real robot environments. For the simulated robot arm environment, the p-value between CAIL and the closest baseline, 2IWIL, is 0.0974. For the real robot environment, the p-value between CAIL and the closest baseline, AIRL, is 0.0209 (statistically significant). In particular, in the real robot environment, CAIL achieves a low standard deviation while other methods especially AIRL, IC-GAIL, D-REX, and GAIL suffer from an unstable performance. The results demonstrate that CAIL can work stably in the real robot environment. We report the success rate—rate that the robot successfully reaches the target within the time limit without colliding with the obstacle—and videos of sample policy rollouts in the supplementary materials.

Demonstrations with Different Optimality. We show the performance of different methods with demonstrations at different levels of optimality in the Reacher environment. We fix 20% of the total demonstrations to be optimal and make the remaining 80% demonstration drawn from the same suboptimal policy. We vary the

optimality of this policy to investigate the performance change with respect to different optimalities. Another way to obtain different optimality is to vary the ratio of optimal demonstrations. We show the results of both varying optimality in Figure 3.4(a) and 3.4(b) respectively. We observe that CAIL consistently outperforms or performs comparably to other methods with demonstrations at different optimality. Also, CAIL performs more stably while the baselines suffer from a performance drop at specific optimality levels.

Learning from Only Non-optimal Demonstrations. We verify that CAIL can also learn from solely non-optimal demonstrations without relying on any optimal demonstrations. We remove the optimal demonstrations in the Ant environment and use the remaining demonstrations to conduct imitation learning. As shown in Figure 3.4(c), CAIL still achieves the best performance among all the methods, which demonstrates that even with all demonstrations being non-optimal, CAIL still can learn useful knowledge from those demonstrations with higher expected return and induce a better policy. The highest p-value between CAIL and the closest baseline (2IWIL) is 0.0067, which indicates the performance gain is statistically significant. We observe that the performance of AIRL first increases and then decreases. This is because even though the demonstrations are suboptimal, there are potentially optimal state-action transitions leading to the initial high performance. However, at this early training stage, the AIRL model does not converge yet and the model parameters can still change rapidly. After training a sufficient number of steps, the AIRL model observes both useful and less useful transitions and converges to the average return of all the demonstrations.

Numerical Comparison. We provide the numerical comparison of CAIL and the baseline methods in Table 3.1. The results correspond to the results in Fig. 2 in the main text. We can observe that CAIL outperforms all the baseline methods in all the environments and the margin between CAIL and the best-performing policy is much closer than the margin between baseline methods and the best-performing policy.

Success Rate. We report the success rate among 100 trials of different methods in Table 3.2. We observe that for both simulated and real robot environments, CAIL achieves the highest success rate. Though 2IWIL also achieves a high success rate;

Table 3.1: The converged expected return of all the methods in Mujoco Reacher and Ant, Simulated Franka Panda Robot Arm, and the Real UR5e Robot Arm environments. We provide numerical results for a clearer comparison.

Method	Reacher	Ant	Simulated Robot	Real Robot
CAIL	**-7.82**±1.52	**3825.64**±940.23	**-62.95**±50.64	**-34.33**±1.24
2IWIL	-23.06±3.80	3473.85±271.70	-120.62±122.79	-52.45±7.18
IC-GAIL	-55.36±5.05	1525.67±747.88	-349.51±342.60	-550.24±657.84
AIRL	-25.92±2.34	3016.13±1028.89	-236.95±230.50	-597.82±752.15
GAIL	-60.86±3.30	998.23±387.83	-527.60±452.38	-532.85±664.42
T-REX	-66.37±21.30	-1867.93±318.34	-1933.94±380.83	-2003.67±32.77
D-REX	-78.10±14.92	-2467.78±135.18	-1817.24±481.67	-1538.10±703.27
SSRR	-70.04±14.74	-105.35±210.84	-2077.62±58.76	-2154.21±168.09
Oracle	-4.31	4787.23	-35.36	-31.06

Table 3.2: Success rate (%) among 100 trials of all the methods in the simulated and real robot environments.

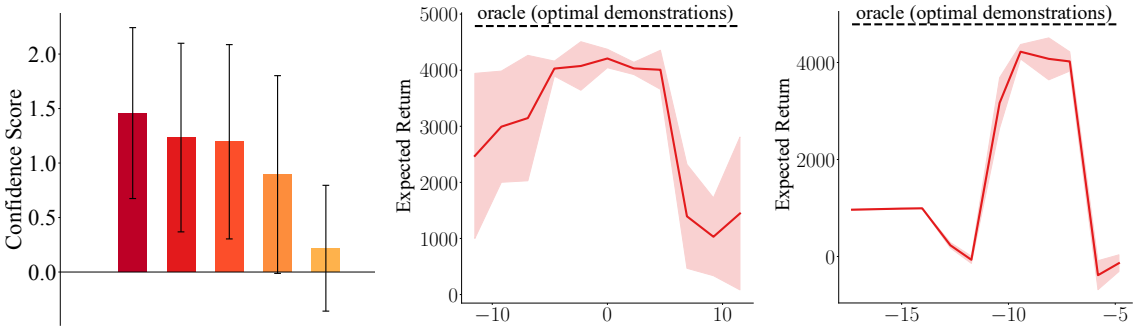
Method	CAIL	2IWIL	IC-GAIL	AIRL	GAIL	T-REX	D-REX	SSRR
Simulated Robot	100	100	81	87	31	0	0	0
Real Robot	100	83	7	33	20	0	0	0

however, it induces trajectories with longer detours and thus has a lower expected return.

Table 3.3: The performance with respect to the size of the ranking dataset.

Label Ratio	1%	2%	5%	10%	20%	50%	100%
2IWIL	-33.5 ± 4.9	-34.4 ± 3.2	-23.3 ± 4.1	-27.7 ± 6.7	-24.5 ± 3.0	-30.0 ± 2.7	-25.2 ± 6.9
IC-GAIL	-56.4 ± 10.1	-53.7 ± 4.0	-61.0 ± 5.0	-54.1 ± 6.0	-58.8 ± 3.4	-44.6 ± 8.3	-57.1 ± 3.7
T-REX	-83.7 ± 18.6	-85.8 ± 15.3	-82.3 ± 10.2	-73.2 ± 21.6	-91.8 ± 15.4	-38.6 ± 35.8	-27.2 ± 37.2
CAIL	$\mathbf{-8.0} \pm 2.4$	$\mathbf{-8.7} \pm 3.6$	$\mathbf{-7.3} \pm 2.0$	$\mathbf{-8.1} \pm 2.9$	$\mathbf{-7.1} \pm 1.7$	$\mathbf{-7.5} \pm 2.3$	$\mathbf{-7.8} \pm 3.0$

Ablating the size of ranking dataset We provide results of CAIL and the compared methods including 2IWIL, IC-GAIL, and T-REX with varying levels of supervision. We do not include GAIL, AIRL, D-REX, and SSRR in this ablation since they do not require any supervision. We conduct experiments in the Reacher environment and vary the ratio of demonstrations labeled with ranking. The agents are provided with



(a) Visualization of Confidence (b) $\log(\alpha)$ (c) $\log(\mu)$

Figure 3.5: (a) The visualization of confidence for demonstrations drawn from policies with different optimality. There are 5 policies with different optimality, where the darker color means the policy has a higher expected return. (b-c) The expected return with varying hyper-parameters α and μ.

200 trajectories, and the ratios of labeled demonstrations are 1%, 2%, 5%, 10%, 20%, 50%, 100%. The average trajectory rewards and the standard deviations are shown in Table 3.3. CAIL outperforms all the other methods with a large gap in all settings, even in the setting with only 1% labeled demonstrations, i.e., only two trajectories are labeled, which is the minimum label we can have for ranking. 2IWIL and IC-GAIL, however, do not perform well, and there is no clear increase in performance as the label ratio increases. This is because what they need is labeled confidence, which is a much stronger type of supervision than ranking. The confidence cannot be accurately recovered when only given rankings. T-REX does not perform well either, but it is getting better as the ratio of labels increases. This experimentally proves that T-REX needs much more data than CAIL to learn a reward function and CAIL can use the demonstrations more efficiently.

Visualization of the Confidence In our experiments, we have 5 sets of demonstrations collected from 5 different policies, where each set of demonstrations has different average returns. So we can learn different average confidence values for each different set of demonstrations. The larger the average return, the larger the average confidence. In our framework, we learn confidence for each state-action pair. We visualize the un-normalized confidence learned by CAIL of these 5 set of demonstrations in

Fig. 3.5(a), where the darker color means the demonstrations have higher expected returns. We observe that the darker color bar has higher confidence, which indicates that CAIL-learned confidence matches the optimality of the demonstrations.

Hyper-parameter Sensitivity. We investigate the sensitivity of hyper-parameters including the two learning rates α and μ. We aim to demonstrate two points: (1) The proposed approach can work stably with the hyper-parameters falling into a specific range; (2) If the hyper-parameters are too large or too small, the performance can drop, which means that tuning the two hyper-parameters are necessary for the performance of our algorithm. We conduct experiments in the Ant environment. As shown in Figure 3.5(b) and 3.5(c), the proposed approach work stably with α in the range $[10^{-3}, 100.0]$, and with μ in the range $[3 \times 10^{-5}, 3 \times 10^{-4}]$. When α and μ are too large or too small, the performance drops. The observations demonstrate the two points introduced above.

3.4 Adversarial Confidence Transfer

To derive the confidence value, in Section 3.2, we use the reward value of each demonstration trajectory as the annotation while in Sec. 3.3, we use the ranking of trajectories. However, for complex environments, large-scale annotations are required to cover the whole state-action space, which is labor-intensive and expensive to obtain. Furthermore, the annotations can often contain a significant amount of noise when collected through crowdsourcing platforms.

In this section, instead of using manually annotated confidence over demonstrations, we leverage confidence annotations in a different but related environment—the *source* environment. As shown in Fig. 3.6, the source environment may have a different state-action space from the *target* environment—the environment we are operating in. The source environment and the target environment should have a correspondence mapping between state-action pairs [52].

We assume that collecting offline demonstrations along with their confidence annotations is easy in the source environment. As an example, the source and target environments can be a simulated robot and a real robot respectively, or two

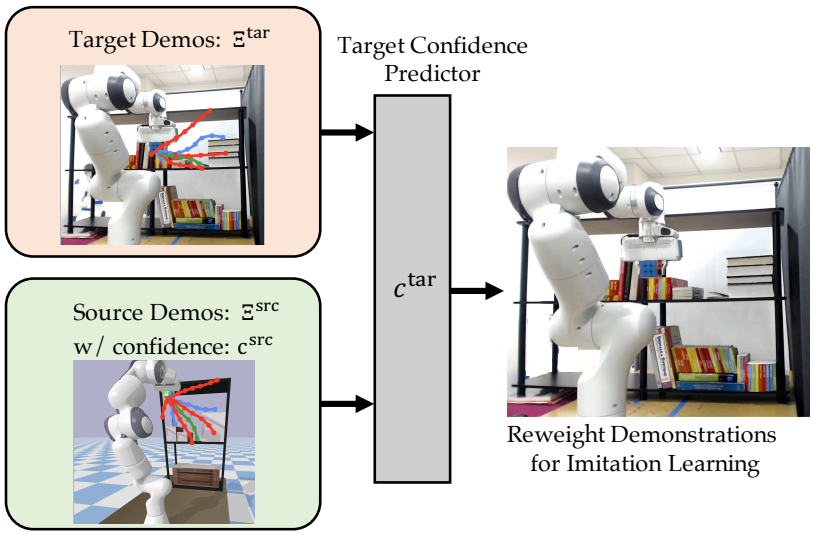


Figure 3.6: Overview of the adversarial confidence transfer.

different robot arms with different degrees of freedom. Our key idea is to leverage the correspondence between state-action pairs in the source and target environments to learn a confidence measure without the restrictive assumption of access to confidence annotations of target demonstrations.

This problem setting introduces a new challenge: The source and target environments have different state-action spaces. That means the dimensions of the states and actions and even the semantic meaning of each dimension can be different between the two environments, so we cannot directly apply the source confidence predictor to the target state-action pairs. We instead need to transfer the source confidence predictor to the target environment.

Specifically, the source agent is modeled as a Markov decision process (MDP), $M^{\text{src}} = \langle \mathcal{S}^{\text{src}}, \mathcal{A}^{\text{src}}, p^{\text{src}}, \mathcal{R}^{\text{src}}, \rho_0^{\text{src}}, \gamma^{\text{src}} \rangle$. The source MDP M^{src} and the target MDP M may differ in any component of the MDP. Our goal is to find the optimal policy for the target agent with the help of the source agent.

Specifically, in this problem setting, apart from a set of imperfect demonstrations Ξ for the target agent, we are also given a set of imperfect demonstrations $\Xi^{\text{src}} = \{\xi_1^{\text{src}}, \xi_2^{\text{src}}, \dots\}$ for the source agent along with a measure of confidence c^{src} in the source environment. Here the confidence function $w^{\text{src}} : \mathcal{S}^{\text{src}} \times \mathcal{A}^{\text{src}} \to \mathbb{R}$ evaluates

how useful the input state-action pairs are similar to the confidence in [47] and if they are available or easy to access. Our goal is to find a well-performing policy π for the target environment. Note that we go beyond standard imitation learning by allowing these demonstrations in both environments to range from useless or even harmful demonstrations to nearly optimal or fully optimal ones. At the high level, our approach is to learn a confidence function $w : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which assigns a value to how confident we are in the optimality of the action a_t in a particular state s_t. We can then reweight the target state-action pairs and conduct standard imitation learning on the reweighted state-action pairs to learn a policy. We would like to model w as a neural network and learn w by leveraging the knowledge of the confidence function w^{src} from the source environment.

Assumptions. To transfer the confidence predictor from the source to the target environment, we assume that the source and target environments are *correspondent*, which means that there exists a relational mapping between the state-action pairs of the source agent and the target agent [52]. Specifically, there exists a relation on states: $\Phi : \mathcal{S}^{\text{src}} \times \mathcal{S}$. There also exists a mapping from source state-action pairs to the target action space, $H_1 : \mathcal{S}^{\text{src}} \times \mathcal{A}^{\text{src}} \to \mathcal{A}$, and another mapping from target state-action pairs to the source action space, $H_2 : \mathcal{S} \times \mathcal{A} \to \mathcal{A}^{\text{src}}$. The mappings Φ, H_1, and H_2 satisfy the following: if $(s^{\text{src}}, s) \in \Phi$, then $\forall a^{\text{src}} \in \mathcal{A}^{\text{src}}, (s^{\text{src}}_{\text{next}}, s_{H_1}) \in \Phi$ and $\forall a \in \mathcal{A}, (s_{\text{next}}, s^{\text{src}}_{H_2}) \in \Phi$. Here, $s^{\text{src}}_{\text{next}}$ is the successor state from s^{src} when taking the action a^{src}, and s_{H_1} is the successor state from state s when taking the action $H_1(s^{\text{src}}, a^{\text{src}})$, and s_{next} and $s^{\text{src}}_{H_2}$ are defined similarly. The assumption builds a connection between the source and target agents. Without this assumption, the two environments may have no relation to each other, and there will not be any reason for knowledge transfer across them. This assumption is often satisfied in many real-world applications. Specifically, we are interested in two common settings: (1) The source environment is a simulation of a real robot that would define the target environment. For example, the source environment can be a simulated robotic arm picking an apple from a bowl in simulation, while the target environment can be a real robotic arm picking an apple from a bowl. (2) The source and target environments are different robots performing the same task in a similar context. For example, the source

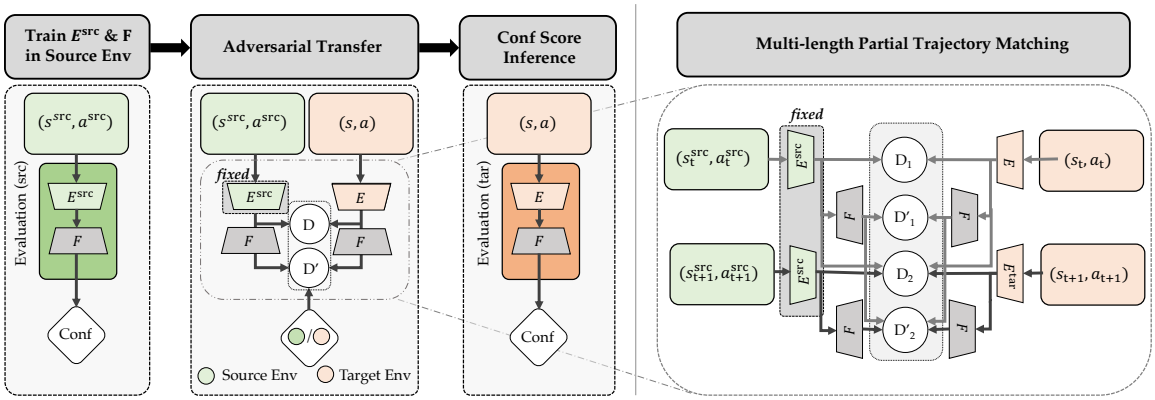


Figure 3.7: (left) The overview of the learning process consisting of three stages: In the first stage, we train the source encoder E^{src} and the shared decoder with source confidence-labeled demonstrations $s^{\text{src}}, a^{\text{src}}$. In the second stage, we align the latent distributions of source and target state-action pairs with a multi-length partial trajectory matching, where we explain the second stage in detail in the right figure. In the third stage, we use the learned E chained with F as the target confidence predictor. (right) Multi-length partial trajectory matching: we align the feature-level distributions (output of encoders E^{src} and E) of different lengths' partial trajectories with different discriminators $D_1, D_2, \ldots, D_n$ and align the confidence-level distributions (output of the shared decoder F) of different lengths' partial trajectories with different discriminators $D'_1, D'_2, \ldots, D'_n$.

environment can be a simpler 3-DoF robot arm reaching the center of a bowl without colliding with obstacles, while the target robot can be a 7-DoF robot performing the same task. We note that the correspondence assumption is less restrictive compared to assumptions in related prior work such as assuming the same dynamics across the environments [48, 8].

3.4.1 Adversarial Confidence Transfer

We aim to leverage the confidence annotation in the source environment and transfer the confidence knowledge to the target environment. Our key insight is to map the source and target state-action pairs to a common latent space $\mathcal{Z}$, and enforce similarity between the latent features (z^{src} and z) for corresponding state-action pairs. Here, $(s^{\text{src}}, a^{\text{src}})$ and (s, a) satisfy $(s^{\text{src}}, s) \in \Phi$ and $(s^{\text{src}}_{\text{next}}, s_{\text{next}}) \in \Phi$. We then build a shared decoder to predict the confidence from the latent features.

Since the source and target environments have different state-action spaces, we

learn two different encoders $E^{\text{src}} : \mathcal{S}^{\text{src}} \times \mathcal{A}^{\text{src}} \to \mathcal{Z}$ and $E : \mathcal{S} \times \mathcal{A} \to \mathcal{Z}$ to map the source and target state-action pairs into a common latent space $\mathcal{Z}$. We then learn a shared decoder $F : \mathcal{Z} \to \mathbb{R}$ to predict the confidence from latent features with confidence-labeled demonstrations in the source environment. The target confidence predictor can be computed by $F \circ E$. Now, the challenge is how to learn the encoders to ensure that the common latent space satisfies the above requirements.

Multi-length Partial Trajectory Matching

To map correspondent state-action pairs to a similar latent feature, we develop a distribution matching objective to align the latent distribution of source and target state-action pairs. We can directly align the distribution of state-action latent features, i.e., matching $E^{\text{src}}(s_t^{\text{src}}, a_t^{\text{src}})$ and $E(s_t, a_t)$. However, this does not capture the temporal dependency between state-action pairs over a trajectory. To address this issue, we incorporate a prior that captures this temporal dependency when learning the encoders. Imagine a setting where a simulated robot and a real robot are both making a burger. The state-action pairs corresponding to placing the ingredients (e.g., the patty, lettuce, and tomatoes) are always after placing the bottom bun. If we only match the state-action pairs one at a time between the simulated and the real robot, placing the bottom bun may match placing any of the ingredients or even the top bun, but if we also match the consecutive steps of the state-action pairs, placing the bottom bun in both environments will more likely capture the information that the bottom bun comes before the rest of the ingredients. So, as shown in Fig. 3.7 (right), we propose a multi-length partial trajectory alignment that emphasizes learning the temporal relationship between state-action pairs. Specifically, we match the latent feature distribution of length k, $(k = 1, 2, \ldots, K)$ partial trajectories, which preserves the temporal relationship between consecutive states and actions and makes the latent features of correspondent state-action pairs more likely to be aligned.

Feature-Level and Confidence-Level Matching

We can now develop our learning algorithm that aligns the latent distribution of state-action pairs. We first train the source encoder E^{src} and the shared decoder F with source confidence-labeled state-action pairs using a regression loss:

$$\mathcal{L}_{\text{reg}} = \mathbb{E}_{(s^{\text{src}},a^{\text{src}})\sim\xi^{\text{src}},\xi^{\text{src}}\sim\Xi^{\text{src}}} \left[\|F(E^{\text{src}}(s^{\text{src}},a^{\text{src}})) - c^{\text{src}}(s^{\text{src}},a^{\text{src}})\|\right]. \tag{3.18}$$

After learning E^{src} and F, we fix the parameters of E^{src}, and F and use the latent feature space of E^{src} as the common feature space. We then only need to make the target encoder E encode the target state-action pairs into this shared feature space and match the source latent feature distribution.

We learn the latent feature distribution alignment using a generative adversarial network. Since we only want the partial trajectories of the same length to be matched, for each length of partial trajectories k, we adopt a discriminator D_k and a loss $\mathcal{L}^k_{\text{fea}}$ to match the latent distribution of length-k partial trajectories. Specifically, the discriminator D_k aims to discriminate the latent features of source and target length-k partial trajectories, while the target encoder E is trained to confuse the discriminator. The loss for the discriminator D_k can be derived as a binary classification loss:

$$\begin{aligned}\mathcal{L}^k_{\text{fea}} = &- \mathbb{E}_{(s_1,a_1,\ldots,s_k,a_k)\sim\xi,(\xi,)\sim\Xi^{\text{src}}} \left[\log\left(D_k(\text{Cat}\left(E^{\text{src}}(s_1,a_1),\ldots,E^{\text{src}}(s_k,a_k)\right))\right]\\ &- \mathbb{E}_{(s_1,a_1,\cdots,s_k,a_k)\sim\xi,\xi\sim\Xi} \left[\log\left(1 - D_k(\text{Cat}\left(E(s_1,a_1),\ldots,E(s_k,a_k)\right))\right].\end{aligned} \tag{3.19}$$

We concatenate the encoded features of multiple consecutive state-action pairs as the latent feature for the length-k partial trajectories. The discriminator D_k is trained to minimize the discriminator loss $\mathcal{L}^k_{\text{fea}}$ and the target encoder E is trained to maximize the discriminator loss.

However, the above losses only match the latent feature distributions of state-action pairs, which is confidence-agnostic. So there may still exist mismatches of state-action pairs with different confidence values. As shown in Fig. 3.7 (right), to address the issue, we introduce a confidence-level distribution matching to match the predicted confidence of source and target state-action pairs, which further guides the

latent features to be matched in a confidence-aware way. Specifically, we use another discriminator D'_k for matching the length-k confidence sequence of length-k partial trajectories using a similar binary classification loss:

$$\begin{aligned}\mathcal{L}^k_{\text{con}} = &- \mathbb{E}_{(s_1,a_1,\cdots,s_k,a_k)\sim\xi,(\xi,)\sim\Xi^{\text{src}}}\left[\log\left(D'_k(\text{Cat}\left(F(E^{\text{src}}(s_1,a_1)),\ldots,F(E^{\text{src}}(s_k,a_k))\right)\right)\right] \\ &- \mathbb{E}_{(s_1,a_1,\ldots,s_k,a_k)\sim\xi,\xi\sim\Xi}\left[\log\left(1-D'_k(\text{Cat}\left(F(E(s_1,a_1)),\ldots,F(E(s_k,a_k))\right)\right)\right].\end{aligned}$$

Similarly, D'_k is trained to minimize the discriminator loss $\mathcal{L}^k_{\text{con}}$, and E is trained to maximize the discriminator loss. To optimize the whole network, we first train the source encoder and the shared decoder with the source confidence-labeled data as shown in Eqn. (3.20) (the first stage in Fig. 3.7 (left)).

$$\min_{E^{\text{src}},F} \mathcal{L}_{\text{reg}} \tag{3.20}$$

We then iteratively train the feature-level and confidence-level discriminators and the target encoder as shown in Eqn. (3.21) and Eqn. (3.22)

$$\min_{D_k,D'_k} \mathcal{L}^k_{\text{fea}} + \mathcal{L}^k_{\text{con}}(k\in[1,K]) \tag{3.21}$$

$$\max_{E} \sum_{k=1}^{K}\left(\mathcal{L}^k_{\text{fea}} + \lambda\mathcal{L}^k_{\text{con}}\right), \tag{3.22}$$

where λ is the trade-off between $\mathcal{L}_{\text{fea}}$ and $\mathcal{L}_{\text{con}}$ (the second stage in Fig. 3.7 (left)). After convergence, we chain the target encoder E and the confidence decoder F to form the target confidence predictor as shown under **Conf. Score Inference** in Fig. 3.7.

We can then predict the confidence of each state-action pair (s,a) in the target environment as $F(E(s,a))$. With the confidence of each state-action pair, when conducting imitation learning on the target demonstrations, we use the confidence to re-weight each state-action pair in the imitation loss as in [48]. Then the imitation learning policy can learn more from useful demonstrations with higher confidence.

Relation to Existing Works. Our method relaxes assumptions in prior works [48, 10]

and targets a more realistic problem. However, we argue that our method is compatible with prior works. The common latent feature space in our method can be regarded as a new space of state-action pairs for the source and target environments, where the correspondence between state-action pairs is preserved, especially w.r.t. confidence. With the common latent feature space, we can easily extend prior works to the setting of different state spaces by conducting the algorithm, that is originally performed on states, on the latent feature. Our contribution is not only providing a new approach to learning from imperfect demonstrations but also expanding the usage of algorithms from prior work by introducing a new space to operate in.

3.4.2 Experiments

We conduct experiments on two MuJoCo environments, a simulated and a real Franka Panda arm, and have released the code here. We compare our method (**Ours**) with baselines **GAIL** [20], Dynamics Cycle-Consistency (**DCC**) [52], and variants of our method by gradually adding modules: starting from GAIL without confidence weighting, we first add feature-level matching as **Ours-Feature**. We then add the confidence-level matching as **Ours-Confidence**. Adding multi-length partial trajectory matching to Ours-Confidence derives **Ours**. We show the results of using the ground-truth confidence to reweight the target demonstrations (**Oracle**).

Our approach uses the source and target demonstrations and the source confidence function as the input. To implement the baseline methods we make the following choices: For GAIL [20], we directly conduct imitation learning from the demonstrations in the target environment without any confidence weights. For DCC [52], the original paper uses trajectories sampled from a random policy to learn the translation mappings. In our setting, we use both the demonstrations and the random trajectories to learn the translation mapping.

To generate the ground-truth confidence score for the source demonstrations, following the common practice in prior works [48, 10], we normalize the expected return of all the demonstrations to $[0, 1]$ by min-max normalization and assign all the state-action pairs in each trajectory with the confidence of the trajectory.

Mujoco Simulation

Demonstration Generation. For the Reacher environment, we compute the median value between the expected returns of a random policy and the optimal policy and find a partially-trained policy to match the value. Then we have three policies: random policy, 50% partially-trained policy, and optimal policy. We generate 94, 5, and 1 trajectories from the random policy, the 50% partially-trained policy, and the optimal policy respectively, which are 100 trajectories in total.

For the Ant environment, we select five values with an equal interval between the expected returns of a random policy and the optimal policy and find three partially-trained policies to match the three values. Then, we have five policies: random policy, 25% partially-trained policy, 50% partially-trained policy, 75% partially-trained policy, and optimal policy. We generate 48, 49, 97, 5, and 1 trajectories from the random policy, the 25% partially-trained policy, the 50% partially-trained policy, and the 75% partially-trained policy, and the optimal policy respectively.

Simulated Robot Arm

Reward Design. We use x_{center} to denote the center of the circle on the table surface, and x_{t} to denote the position of the end-effector at time step t. Based on this goal, we develop a reward function consisting of three parts: the first part is the negative L2 distance between the x_{t} and x_{center}; the second part penalizes collisions with -1000 reward when the end-effector collides with the wall or does not reach the table within the time limit T_h; and the third part is a gain of positive reward when the end-effector reaches the table surface within the circle area and the closer to the center, the higher the reward. The reward for the simulated robot arm environment is defined as $R_t = -\|x_t - x_{\text{center}}\|_2 - 1000 \times (\mathbb{I}\left[\text{collide}\right] |\mathbb{I}\left[t \geq T_h\right]) + 400 \times \exp(-\frac{\|x_{\text{reach}} - x_{\text{center}}\|_2}{0.1}) \times \mathbb{I}\left[\text{reach}\right]$. The initial position of the joint that controls the rotation of the hand can vary freely in $[0, 2\pi]$, and the initial position of all the other joints of the arm can vary ± 0.3. We create an initial position area within these regions to allow some variance for the task while not exceeding the joint limits of the robot.

Demonstration Generation. We create out-of-time trajectories by injecting noisy

actions with some probability and we vary the probability to create different kinds of out-of-time trajectories. We vary the suboptimal trajectories by changing the distance from the goal point on the table to the circle center.

Composition of Demonstrations. In both the source and target environments, we use the same composition of trajectories. In the OR setting, we collect 10 optimal trajectories and 200 out-of-time trajectories. In the OSC setting, we collect 40 optimal trajectories, 80 suboptimal trajectories, and 150 collision failure trajectories. The composition is designed to contain more suboptimal demonstrations and failure cases to make the imitation learning problem more challenging.

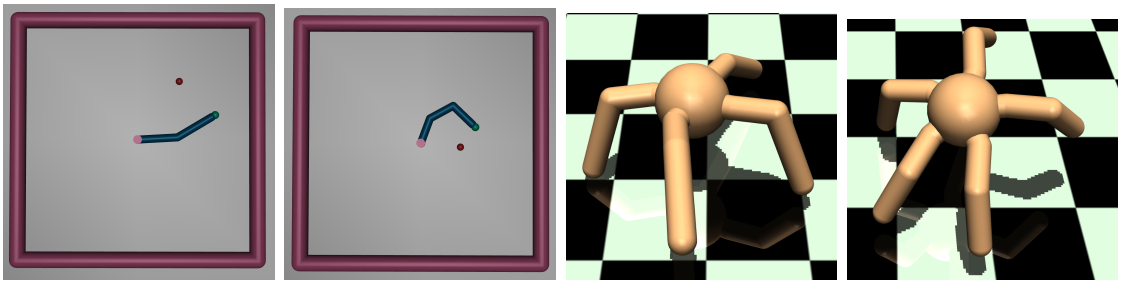
Sim-to-Real Environment

In the Sim-to-Real environment, the network and the imitation learning algorithm are the same as in the simulated robot arm environment.

Composition of Demonstrations. For the simulated robot arm, we collect human demonstrations by controlling the end-effector by the mouse. For the real robot arm, we collect human demonstrations by moving the robot arm by hand. For both the simulated and real robot arms, the demonstrations are composed of 5 trajectories for placing the cube on the left, middle, and right each, 5 trajectories colliding with the left, up, and bottom boundary of the upper shelf each, and 5 trajectories colliding with the books on the right side (we cannot reach the right boundary due to the joint limit).

To implement our approach in the simulated robot arm and the real robot arm, we use a three-layer fully-connected network for E^{src} and E respectively, a one-layer fully-connected network for F and a three-layer fully-connected network for D_k and D'_k $(k = 1, \cdots, K)$ respectively. We use behavior cloning [2] as the imitation learning algorithm to learn the final policy from the reweighted demonstrations.

For the Mujoco environments, we evaluate the average return of 100 rollouts. For both the simulated robot and sim-to-real environments, we evaluate the average return of 500 rollouts. In all the experiments, we compute the results over 10 runs and report the mean and the standard deviation. We compute the p-values using the student's t-test.



(a) 1-joint (b) 2-joint (c) 4-leg (d) 5-leg

Figure 3.8: Illustration of source and target MuJoCo environments. The left two figures are Reacher and the right two are Ant.

Results

MuJoCo Environment. We create 4 different MuJoCo environments (see Fig. 3.8): 1-joint and 2-joint reacher, 4-leg and 5-leg ant. 1-joint reacher and 4-leg ant are the source environments, and the 2-joint reacher and 5-leg ant are the target environments. The task for the reacher is to reach the red point from its initial configuration. The task for the ant is to move towards the right horizontally as fast as possible. We train the optimal policy using the RL algorithm TRPO [39]. For the Reacher environment, we select the random policy, a partially-trained policy, and the optimal policy to collect demonstrations with mixed confidence. For the Ant environment, we select the random policy, three partially-trained policies, and the optimal policy to collect demonstrations with mixed confidence.

The expected return for these experiments is shown in Table 3.4. **Ours** outperforms **GAIL** and **DCC**, and is comparable with the **Oracle** in both environments. Also, **Ours** outperforms **Ours-Confidence**, which demonstrates the efficacy of multi-step partial trajectory matching. **Ours-Confidence** outperforms **Ours-Feature** and **Ours-Feature** outperforms **GAIL**, which demonstrate the efficacy of confidence-level and feature-level matching respectively. For 2-joint reacher and 5-leg ant, the highest p-values comparing with baselines (between **Ours** and **DCC**) are 0.147 and 0.031 (statistically significant) respectively.

Figure 3.9(a) and 3.9(c) show the learning curves of the algorithms. We observe that all methods have similar convergence rates. Figure 3.9(b) and 3.9(d) show a

	Reacher	Ant
GAIL	-47.13±5.67	507.85±338.80
DCC	-41.11±5.35	1218.59±231.74
Ours-Feature	-43.97±4.68	1059.02±280.95
Ours-Confidence	-41.82±6.80	1263.04±343.02
Ours	**-39.53**±4.32	**1556.43**±159.12
Oracle	-28.73±3.97	1622.55±179.43

Table 3.4: The expected return for MuJoCo environments.

rollout generated by each of the policies. The rollout from the oracle policy reaches the goal in the most effective manner following the shortest trajectory length. **Ours** is the next closest to the **Oracle**, while the **GAIL** baseline fails at reaching the goal position at times.

Simulated Robot. As shown in Fig. 3.10, we create a task to move the end-effector of a 5 DoF Panda Franka Robot arm toward the center of a plate without colliding with the boundaries. The reward function consists of the negative L2 distance to the center; collisions with the wall or failing to reach the table within the time limit; and positive reward when the end-effector reaches the plate.

The source robot arm has 7 DoF and the target has 5 DoF, where the joints marked by red in Fig. 3.10 are disabled. We show different kinds of trajectories: (green–optimal trajectory reaching the center of the plate), (blue–suboptimal trajectories reaching the plate but not the center), (red–failure cases colliding with the wall), (violet–another type of failure case failing to reach the plate within the time limit). We create two settings: (1) (OT) consists of the optimal (green) and out-of-time (violet) trajectories; and (2) (OSC) consists of optimal (green), suboptimal (blue), and collision (red) trajectories. In the OT setting, we use the joint and end-effector position as the state, which serves as an easy setup since the end-effector position is shared between the 7-DoF and the 5-DoF. In the OSC setting, we use the joint position, velocity, and torque as the state, which is difficult to align. For both settings, we use the 3D end-effector position difference as the action space.

The expected return and the success rate are shown in Table 3.5. **Ours** outperforms **GAIL** and **DCC** with a large margin, which indicates that the learned target

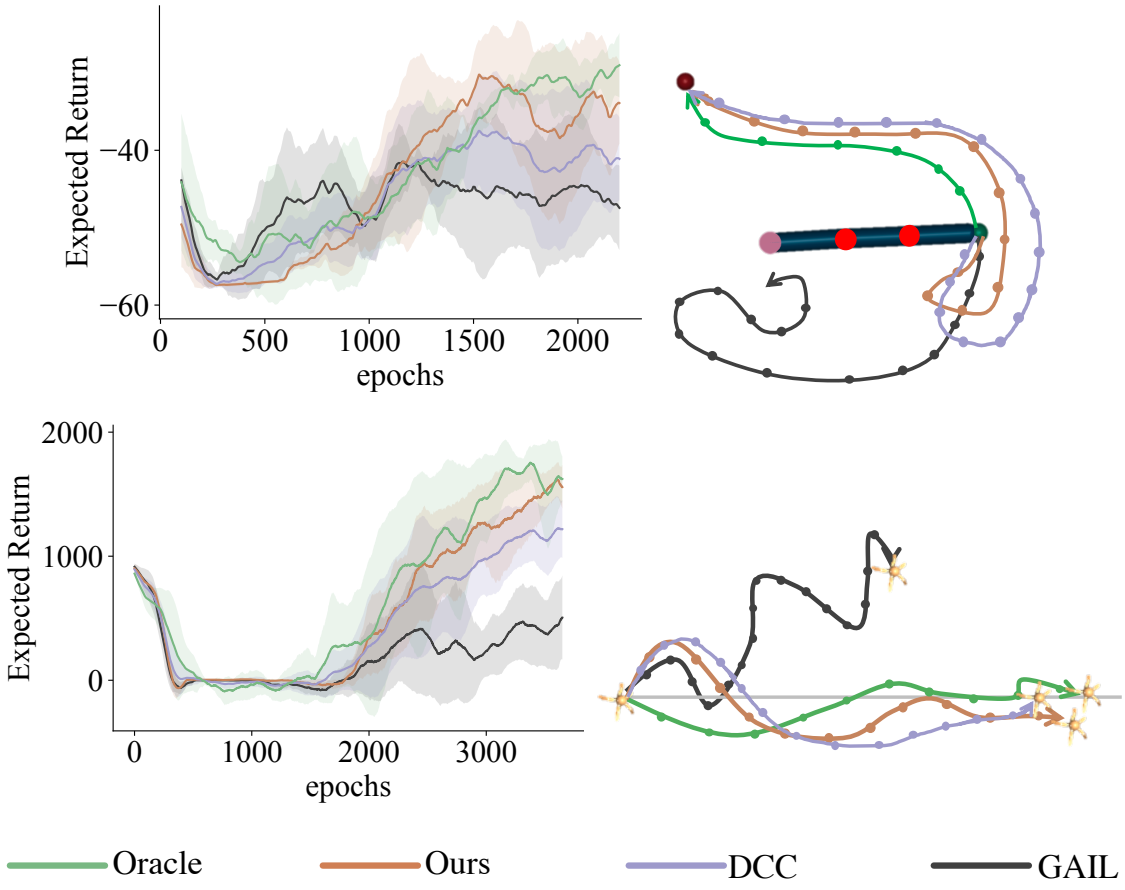


Figure 3.9: The top row and the bottom row are the expected return and sample trajectories for 2-joint reacher and 5-legged ant respectively. The light grey line in the Ant environment is the positive x-axis, which is the direction the ant is supposed to move toward.

confidence predictor can assign higher confidence to useful demonstrations than **DCC**, and **GAIL**. The order of the performance from high to low is **Ours**, **Ours-Confidence**, **Ours-Feature**, and **GAIL**, which demonstrates the efficacy of our multi-length partial trajectory matching, confidence-level matching, and feature-level matching respectively. The highest p-values when comparing with baselines (between **Ours** and **DCC**) are 5.64×10^{-12} for the success rate and 1.25×10^{-13} for the expected return in the OR setting, and 0.006 for the success rate and 0.031 for the expected return in the OSC setting, demonstrating statistical significance.

Sim-to-Real Environment. In the sim-to-real environment, we use a simulated and

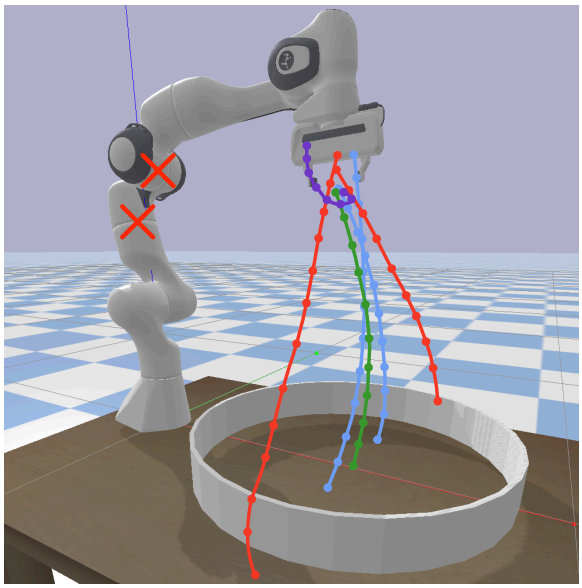


Figure 3.10: Illustration of different trajectories and the 5 DoF robot arm.

a real Franka Panda Arm as the source and the target respectively (both with 7-DoF). The task is to move the cube to the upper layer of the shelf (see Fig. 3.11). On the shelf, there is a large stack of books on the right and a small stack on the left. The middle area is empty. We assign positive rewards to the success of placing the cube on the shelf, with rewards 200, 300, and 100 for placing it on the left, the middle, and the right respectively. We penalize time by giving a -1 reward for each time step. The average number of steps for all demonstrations is about 300. If the arm fails to put the cube within 1000 steps, it receives a reward 0. We use the joint position, velocity, and end effector position as the state space and joint forces as the action space.

Fig. 3.11(c) and 3.11(d) show **Ours** outperforms baselines **DCC** and **GAIL**. The performance from high to low is: **Ours**, **Ours-Confidence**, **Ours-Feature**, and **GAIL**, which demonstrates that multi-length partial trajectory matching, feature-level, and confidence-level matching are all necessary to learn a common latent space and an accurate target confidence predictor for the real robot. The p-value between **Ours** and the highest baseline, **DCC**, is 0.0004 for the expected return and 0.0052 for the success rate, demonstrating statistical significance.

Varying Composition of Demonstrations. We conduct experiments by varying

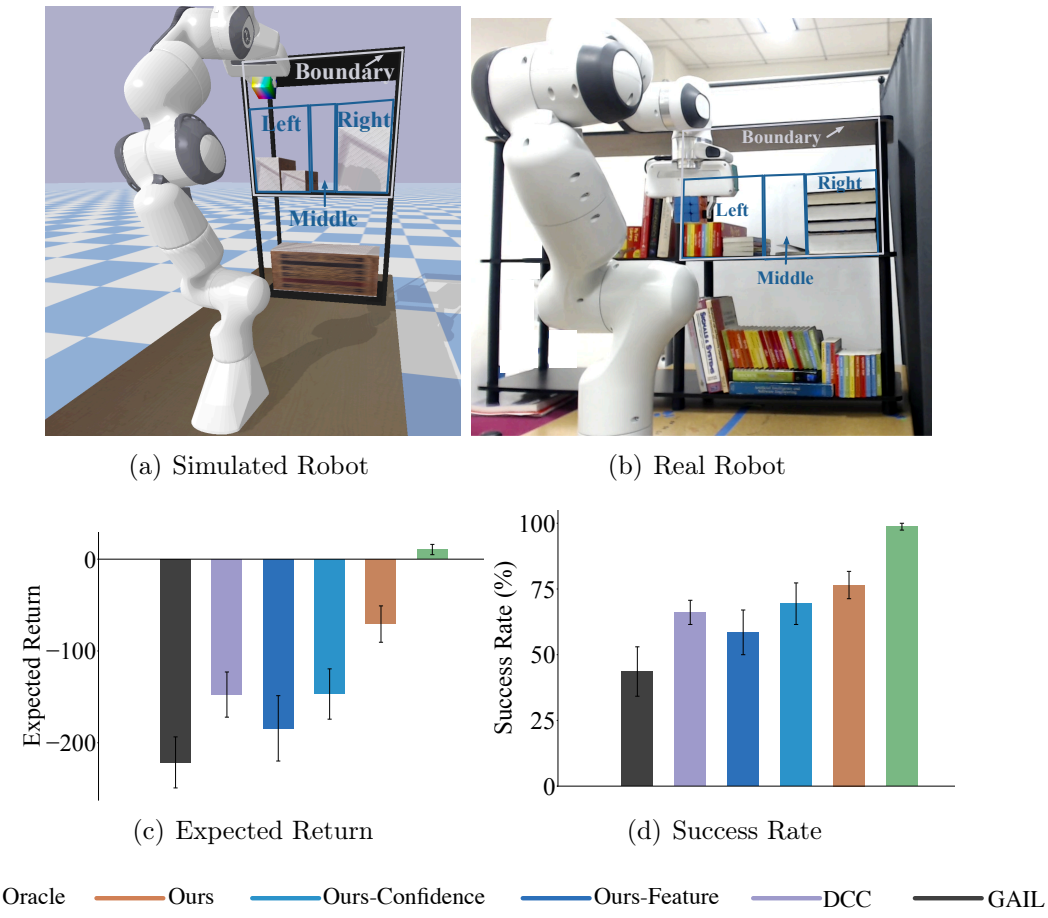
Table 3.5: The expected return and the success rate among 500 trials (%) of GAIL, DCC, Ours-Single, Ours w/o $\mathcal{L}_{\text{con}}$, Ours, and Oracle for the two setups OR and OSC in the simulated robot experiments.

Method	**OT**		**OSC**	
	Return	Success	Return	Success
GAIL	-902.2±152.2	42.8±11.9	-1065.8±105.8	21.2±8.6
DCC	-495.6±46.5	76.4±3.4	-723.7±121.4	49.5±15.2
Ours-Feature	-1198.4±80.4	31.2±7.5	-868.1±265.7	37.6±18.3
Ours-Confidence	-781.9±12.7	59.6±0.8	-821.2±121.4	42.0±11.9
Ours	**-154.8**±12.8	**92.8**±1.0	**-702.2**±104.4	**52.4**±7.9
Oracle	-55.6±2.3	100.0±0.0	-613.9±159.4	59.6±13.2

Table 3.6: Expected return with respect to varying compositions of the source demonstrations for Reacher and Ant. The numbers in the demonstration column indicate the number of the demonstrations collected from random to optimal policies (3 policies for Reacher and 5 policies for Ant).

Reacher		**Ant**	
Demonstration	Expected Return	Demonstration	Expected Return
1-5-94	**-36.69**±5.65	48-49-97-5-1	**1525.03**±176.91
1-49-50	-37.38±10.11	48-73-73-5-1	1489.85±268.22
1-94-5	-37.96±6.61	48-97-49-5-1	1436.78±176.91
47-48-5	-37.61±5.50	72-73-49-5-1	1351.44±115.59
94-1-5	-39.49±2.56	97-48-49-5-1	1345.96±299.08

the composition of source demonstrations but fixing the target demonstrations in the two Mujoco environments to test the robustness of the confidence predictor under different demonstration mixtures. We do not change the target demonstrations since that will also influence the imitation learning performance and we cannot test the efficacy of the confidence predictor. In the *Demonstration* column in Table 3.6, we show the number of demonstrations that are collected from different policies from random to optimal in Reacher (3 policies) and Ant (5 policies). From top to bottom, the optimality of source and target demonstrations deviates more and more. We observe that the performance drops with a larger deviation but does not drop too much even on the last row. This demonstrates that the proposed approach can work



(a) Simulated Robot (b) Real Robot

(c) Expected Return (d) Success Rate

Figure 3.11: (a-b) Illustration of the simulated robot and the real robot arm environments. Different colors indicate different areas to place the objects. (c-d) Expected return and success rate.

stably even with a confidence distribution shift between demonstrations.

3.5 Chapter Summary

In this chapter, we introduce the challenges of learning from suboptimal demonstrations and propose several algorithms to tackle suboptimal demonstrations. The key to addressing this problem is developing a confidence measurement to down-weight suboptimal demonstrations and learn more from closer to optimal demonstrations. To

learn the confidence measurement, we first resort to the expected return and design a confidence measurement based on the rectified expected return.

We then propose a general learning framework, Confidence-Aware Imitation Learning, for imitation learning from demonstrations with varying optimality. We adopt standard imitation learning algorithms with their corresponding imitation loss (inner loss) and leverage an outer loss to evaluate the quality of the imitation learning model. In this way, we can simultaneously learn a confidence score over the demonstrations using the outer loss and learn the policy by optimizing the inner loss over the confidence-reweighted distribution of demonstrations. This framework is applicable to any imitation learning model with compatible choices of inner and outer losses. We also provide theoretical guarantees on the convergence of CAIL and show that the learned policy outperforms baselines on various simulated and real-world environments under demonstrations with varying optimality.

Besides leveraging the annotations in the target agent, we further propose an algorithm to learn the confidence predictor for the target agent by leveraging confidence labels and demonstrations in a different but correspondent environment.

Our experimental results show that all of our methods can learn an optimal or near-optimal policy from suboptimal demonstrations.

Chapter 4

Correspondence Learning from Cross-Domain Demonstrations

4.1 Cross-Domain Demonstrations

Humans are born with the ability to develop new skills by mimicking the behavior of others who may have different embodiments [31]. For example, prior cognitive science work suggests that 1- or 2-year-old children can infer the intentions of adults and re-enact their behavior with their own body even with a large difference in body structures [32, 30]. We refer to the ability to infer the mapping between the state, and action pairs of agents with different dynamics or embodiment as *correspondence learning*. Cross-domain demonstrations refer to demonstrations that are collected on other robots but have correspondence to the target agent. Thus, in applications with limited demonstrations, we could learn the correspondence between agents to enable the target agent to learn from these cross-domain demonstrations, which resolves the issue of insufficient demonstrations.

To learn the correspondence between agents, several prior works leverage paired trajectories to learn invariant representations across agents [17, 41, 24, 52], where the representation only preserves the information that is relevant to the downstream tasks. However, collecting and annotating paired trajectories require experts with substantial domain knowledge and is usually expensive to access at a large scale.

Due to the difficulties of collecting paired data, several works propose learning the correspondence between environments as a translation map between the agents using unpaired trajectories [53, 3]. The key insight of these works is adopting a regularization term over the translation model, where cycle-consistency is the most commonly used regularization [44, 22, 23, 51]. However, with no supervision, the quality of the learned correspondence model is usually not as good as models learned with strong supervision over paired data [50, 42].

In section, we propose *Weakly Supervised Correspondence Learning* (WeaSCL) to find a trade-off between strong supervision of strictly paired data and regularization over unpaired data. *Our key insight is to leverage weak supervision that is useful for learning correspondence and also is easy to access in real-world applications.* We propose two types of weak supervision: i) temporal ordering in states and actions, and ii) paired abstractions over data.

The temporal ordering, which originates from the nature of sequential decisions, indicates the temporal dependency of the consecutive states and actions. Such ordering has been used in other domains to detect the discontinuities [4]. Leveraging temporal dependency as a measure of weak supervision enables us to avoid compounding errors of translation maps that can be accumulated over long horizons.

We define *paired abstractions* by a similarity metric over some abstraction of states or state-action pairs of the agents. For example, the location of a mobile robot, the pose of an end-effector, or the confidence of a behavior can potentially be suitable abstractions over data. When learning correspondence between two agents, one can consider a pair of these abstractions as opposed to paired data. The paired abstractions are easier to obtain and annotate than strictly paired data, as annotators would have an easier time comparing similarity over simpler abstractions. For example, in Fig. 4.1, it would be difficult to align the full states including the joint angles of the trajectories of a four- and five-legged Ant. On the other hand, it is much easier and more informative to decide if an abstraction of the state, e.g., the location of the Ant agents on the 2D plane is aligned. We collect such paired abstractions and learn a similarity function over this data. We then incorporate this similarity function in the loss function imposing a constraint on the translation maps.

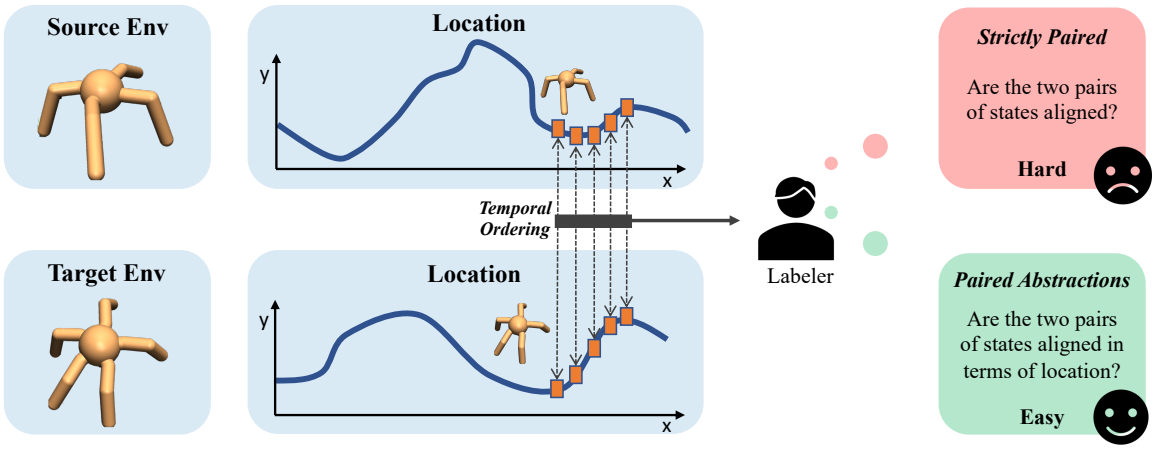


Figure 4.1: An example of the paired abstractions. Given two trajectories of four- and five-legged ant robots, it is difficult to decide whether two full states that include joint angles of each agent are aligned, while it is easy to align simpler abstractions over these states such as whether the ants have the same spatial location.

4.2 Correspondence Learning

We focus on learning correspondence between two agents. However, we note that one can extend this to multiple agents by building correspondence between pairs of agents. Since correspondence learning is not limited to the imitation learning problem, *we use a new set of notations and definitions in this chapter*. We model each agent as a deterministic Markov Decision Process (MDP): $\mathcal{M}^1 = (\mathcal{S}^1, \mathcal{A}^1, \mathcal{T}^1, \mathcal{R}^1, p_0^1, \gamma)$ and $\mathcal{M}^2 = (\mathcal{S}^2, \mathcal{A}^2, \mathcal{T}^2, \mathcal{R}^2, p_0^2, \gamma)$. Similar to [51], we define a correspondence from $\mathcal{M}^1$ to $\mathcal{M}^2$ as follows: Let $\Phi : \mathcal{S}^1 \to \mathcal{S}^2$ be a state map, and $H^1 : \mathcal{S}^1 \times \mathcal{A}^1 \to \mathcal{A}^2$ and $H^2 : \mathcal{S}^2 \times \mathcal{A}^2 \to \mathcal{A}^1$ be two action maps, where the state map and the action maps satisfy the following requirements: $\forall s^1 \in \mathcal{S}^1$, if $s^2 = \Phi(s^1)$, then $\forall a^1 \in \mathcal{A}^1, \Phi(\mathcal{T}^1(s^1, a^1)) = \mathcal{T}^2(s^2, H^1(s^1, a^1))$ and $\forall a^2 \in \mathcal{A}^2, \Phi(\mathcal{T}^1(s^1, H^2(s^2, a^2))) = \mathcal{T}^2(s^2, a^2)$. Intuitively, the requirements mean that the successor states of the two aligned states should be aligned if taking aligned actions.

Using this correspondence definition, we are now ready to introduce our problem statement. We assume access to three pieces of information: a set of trajectories (sequence of state, action pairs) $\Xi^1 = \{\xi^1\}$ for $\mathcal{M}^1$, a set of trajectories $\Xi^2 = \{\xi^2\}$ for $\mathcal{M}^2$, and one or multiple sets of paired abstractions over the states or the state-action

pairs. Specifically, we have K^s sets of paired abstractions over states: $Y_1^s, Y_2^s, \ldots, Y_{K^s}^s$ and K^a sets of paired abstractions over state-action pairs. Each Y_k^s is a set of pairs of states and similarity labels over abstractions of states: $Y_k^s = \{(s^1, s^2, v^s)\}$, where $v^s \in [0, 1]$ reflects the similarity of one choice of abstraction, e.g., the pose of an end-effector, over the state s^1 and s^2. Note that the data tuples (s^1, s^2, v^s) are given by annotators, where the annotators decide which abstraction to take and how to annotate similarity. Our algorithm does not have access to the choice of abstraction and similarity but aims to learn a similarity function $\Phi_k^{\text{weak}} : \mathcal{S}^1 \times \mathcal{S}^2 \to [0, 1]$ mapping the raw pairs of states to a similarity value based on the given data tuples. Similarly, each $Y_k^a = \{((s^1, a^1), (s^2, a^2), v^a)\}$ and $v^a \in [0, 1]$ reflects the similarity of a choice of abstraction over (s^1, a^1) and (s^2, a^2), and we aim to learn a similarity function $H_k^{\text{weak}} : \mathcal{S}^1 \times \mathcal{A}^1 \times \mathcal{S}^2 \times \mathcal{A}^2 \to [0, 1]$ mapping the raw pairs of state-action pairs to the similarity value. Our goal in correspondence learning is to learn the state map Φ and the action map H^1 and H^2 with Ξ^1, Ξ^2, and the similarity functions learned from the paired abstraction data $Y_1^s, \ldots, Y_{K^s}^s$ and $Y_1^a, \ldots, Y_{K^a}^a$.

We would like to emphasize that the paired abstractions only consider a loose alignment between the states and actions of the two MDPs. For example, going back to Fig. 4.1, using paired abstractions, we only consider the pairing of locations of the two Ant agents as opposed to the strict alignment of the full states including the joint angles or velocities. Such loose pairing of the states—pairing of abstractions over states—simply can be assessed by visual observations, and collecting such data along with annotations is much easier, and can serve as cheap supervision.

4.3 Background on Dynamics Cycle-Consistency

To address our correspondence learning problem, we would like to first introduce some background on dynamic cycle-consistency (DCC) [51]. DCC first uses adversarial learning to ensure that the states mapped by Φ fall into the domain of $\mathcal{M}^2$. Specifically,

one can learn Φ with a discriminator D^s by the following adversarial objective:

$$\begin{aligned}&\min_{\Phi}\max_{D^s}\mathcal{L}^s_{\text{adv}}(\Phi, D^s) = \\ &\mathbb{E}_{s^2\sim\Xi^2}[D^s(s^2)] + \mathbb{E}_{s^1\sim\Xi^1}[1 - D^s(\Phi(s^1))].\end{aligned} \tag{4.1}$$

In addition, DCC ensures that the actions mapped by H^1 and H^2 also match the actions in the domain of $\mathcal{M}^2$ and $\mathcal{M}^1$ using discriminators D^{a^1} and D^{a^2} respectively:

$$\begin{aligned}&\min_{H^1,H^2}\max_{D^{a^1},D^{a^2}}\mathcal{L}^a_{\text{adv}}(H^1, H^2, D^{a^1}, D^{a^2}) = \\ &\quad\mathbb{E}_{a^2\sim\Xi^2}[D^{a^2}(a^2)] + \mathbb{E}_{(s^1,a^1)\sim\Xi^1}[1 - D^{a^2}(H^1(s_1, a^1))] \\ &+ \mathbb{E}_{a^1\sim\Xi^1}[D^{a^1}(a^1)] + \mathbb{E}_{(s^2,a^2)\sim\Xi^2}[1 - D^{a^1}(H^2(s_2, a^2))].\end{aligned} \tag{4.2}$$

Finally, one can add a domain cycle-consistency objective on the state-action maps H^1 and H^2:

$$\begin{aligned}&\min_{H^1,H^2}\mathcal{L}_{\text{dom\_con}}(H^1, H^2) = \\ &\quad\mathbb{E}_{(s^1,a^1)\in\Xi^1}\left[\|H^2\left(\Phi(s^1), H^1(s^1, a^1)\right) - a^1\|\right] \\ &+ \mathbb{E}_{(s^2,a^2)\in\Xi^2}\left[\|H^1\left(\Phi(s^2), H^2(s^2, a^2)\right) - a^2\|\right].\end{aligned} \tag{4.3}$$

This equation ensures that the two action maps are consistent with each other and the translated action should be able to be translated back.

The adversarial training as proposed so far suffers from the mode collapse problem [16], where multiple states for one agent can potentially be mapped to one state in the other. In addition, the domain cycle-consistency cannot solve the problem when the two maps H^1 and H^2 make consistent mistakes. For example, we can map (s^1, a^1) to an incorrect action, e.g., $\bar{a}^2$, by H^1 and map it back to a^1 by H^2. Here, both maps make mistakes but the domain consistency is still preserved. To address this issue, DCC introduces the dynamics cycle-consistency objective:

$$\begin{aligned}&\min_{\Phi,H^1}\mathcal{L}_{\text{dyn\_con}}(\Phi, H^1) = \\ &\quad\mathbb{E}_{(s^1_t,a^1_t,s^1_{t+1})\sim\Xi^1}\left[\left\|\Phi(s^1_{t+1}) - \mathcal{T}^2\left(\Phi(s^1_t), H^1(s^1_t, a^1_t)\right)\right\|\right].\end{aligned} \tag{4.4}$$

Here, the transition function $\mathcal{T}^2$ for $\mathcal{M}^2$ is not always known and can be non-differentiable. So one can empirically learn a transition function $\hat{\mathcal{T}}^2$ using the following objective:

$$\min_{\hat{\mathcal{T}}^2} \mathcal{L}_{\text{forward}}(\hat{\mathcal{T}}^2) = \mathbb{E}_{(s_t^2, a_t^2, s_{t+1}^2)\sim\Xi^2} \left[\left\| s_{t+1}^2 - \hat{\mathcal{T}}^2(s_t^2, a_t^2) \right\| \right]. \tag{4.5}$$

Combining all the losses introduced so far, the final optimization objective is:

$$\begin{aligned} \mathcal{L}_{\text{DCC}} =& \lambda_0 \mathcal{L}_{\text{dyn\_con}}(\Phi, H^1) + \lambda_1 \mathcal{L}_{\text{dom\_con}}(H^1, H^2) \\ &+ \lambda_2 \mathcal{L}^a_{\text{adv}}(H^1, H^2, D^{a^1}, D^{a^2}) + \lambda_3 \mathcal{L}^s_{\text{adv}}(\Phi, D^s), \end{aligned} \tag{4.6}$$

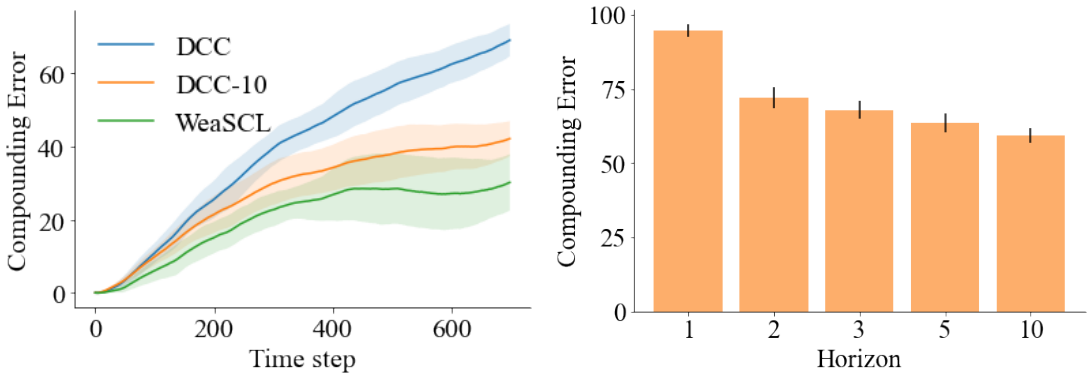
where λ_0, λ_1, λ_2 and λ_3 are hyperparameters trading off between the different losses. DCC firstly trains the forward dynamics $\hat{\mathcal{T}}^2$ and then trains the translation model with $\mathcal{L}_{\text{DCC}}$.

Limitations of DCC. Here, we discuss two core shortcomings of DCC—compounding error and misalignment—which can lead to errors in the translation model.

The compounding error problem refers to the fact that the single-step errors from the state and action maps can accumulate over a sequence. We empirically demonstrate the existence of compounding errors by selecting a segment of a trajectory with horizon T: $\xi^1 = \{s_0^1, a_0^1, \ldots, s_T^1\}$ in Ξ^1. We use two methods to derive the translated state at time step T: (1) $s_T^2 = \Phi(s_T^1)$; (2) $\hat{s}_T^2 = \mathcal{T}^2\left(\cdots \mathcal{T}^2\left(\Phi(s_0^1), H_1(s_0^1, a_0^1)\right), \ldots, H_1(s_T^1, a_T^1)\right)$. The second method continuously uses the translated action to generate the next state to follow the transition process in ξ^1. Comparing the states reached by the first and second approaches, we can empirically check whether there exists compounding errors if we consecutively use the translated actions. We experiment in the Mujoco HalfCheetah environment to build a correspondence between the two-legged and three-legged robots. As shown in Fig. 4.2(a), the distance of s_T^2 and $\hat{s}_T^2$ for DCC gets larger over time, which suggests the existence of compounding errors in the action maps. We hypothesize that this is due to the fact that dynamics cycle-consistency is only ensured for one time step and leads to a small error in that step but cannot bound the error over a long horizon.

Dynamic cycle-consistency still suffers from misalignment issues. For example, assume we are given two trajectories ξ_A^1 and ξ_B^1 for the agent following $\mathcal{M}^1$ and two

trajectories ξ_A^2 and ξ_B^2 for the agent following $\mathcal{M}^2$, where the four trajectories have the same number of time steps. Let's assume the ground-truth translation should translate ξ_A^1 to ξ_A^2 and ξ_B^1 to ξ_B^2. However, if one only enforces dynamics cycle-consistency, it is possible to learn a map that translates the states and actions at each step from ξ_A^1 to ξ_A^2 and from ξ_B^1 to ξ_B^2, or translates from ξ_A^1 to ξ_B^2 and from ξ_B^1 to ξ_A^2, where both maps have zero errors in terms of dynamics cycle-consistency. So the misalignment issue can occur without strong supervision of paired data. However, strictly paired data is often difficult to collect, and we thus aim for some intermediate supervision such as learning similarities between paired abstractions over states, which are much easier to annotate.



(a) Compounding Error (b) Different Horizon

Figure 4.2: (a) The translation error at each time step. (b) The compounding error with respect to different final horizons.

4.4 Weakly-Supervised Correspondence Learning

Based on the above discussion, we propose weakly supervised correspondence learning (WeaSCL), which addresses the above issues with two types of weak supervision: the temporal ordering of information and paired abstraction data.

Multi-Step Dynamics Cycle-Consistency. As we discussed in Sec. 4.3, even a small error for the state map and the action maps at each step will cause a large deviation in a long horizon because DCC only enforces one-step consistency and the

error can accumulate across time steps given no constraint. To address this problem, we use the weak supervision of consecutive states and actions to enforce the dynamics cycle-consistency over multiple steps. The new loss can be formulated as follows:

$$\min_{\Phi,H^1} \mathcal{L}_{\text{m\_dyn\_con}}(\Phi, H^1) = \mathbb{E}_{(s_t^1,a_t^1,s_{t+1}^1,\cdots,s_{t+T}^1)\sim\Xi^1} \sum_{\tau=1}^{T} \left[\left\|\Phi(s_{t+\tau}^1) - \hat{\mathcal{T}}^2\left(\cdots\hat{\mathcal{T}}^2\left(\Phi(s_t^1),\hat{a}_t^2\right)\cdots\hat{a}_{t+\tau-1}^2\right)\right\|\right], \tag{4.7}$$

where $\hat{a}_t^2 = H_1(s_t^1, a_t^1)$ is the translated action at time t and T is the final horizon to enforce dynamics cycle-consistency. With this new loss, as shown in Fig. 4.2(a), with a final horizon 10, the compounding error is substantially reduced.

Now the question is how long of the final horizon we should enforce the dynamics cycle-consistency. If the final horizon is small, the compounding error problem could still exist. If the final horizon is too large, computing the loss at each step can lead to high computation costs. We conduct an experiment on the performance of translation with respect to the final horizon in the HalfCheetah environment. We create two agents $\mathcal{M}^1$ *with three legs* and $\mathcal{M}^2$ *with two legs*. The two agents different in the body structure and to achieve the same trajectory like moving forward for one unit, $\mathcal{M}^1$ needs to coordinate the 3 legs while $\mathcal{M}^2$ needs to coordinate the 2 legs. So they may take different actions to achieve the same trajectory.

We translate the states of $\mathcal{M}^1$ to $\mathcal{M}^2$ with Φ and take the optimal action based on the optimal policy of $\mathcal{M}^2$. We then translate the action back to $\mathcal{M}^1$ with H^2. In Fig. 4.2(b), we observe that the performance of translation increases with a longer horizon at first but saturates from horizon 5 onwards. The observation indicates that we can treat the final horizon as a hyperparameter and tune it by gradually increasing the horizon until when the performance saturates.

Learning Correspondence by Weak Supervision. To address the misalignment issue, we adopt weak supervision from paired abstractions over states or state-action pairs, where a similarity metric is defined on the abstractions, e.g., the location, end-effector pose, or confidence over a state or state-action pair as opposed to the full information. The key difference between strictly paired data and paired abstractions

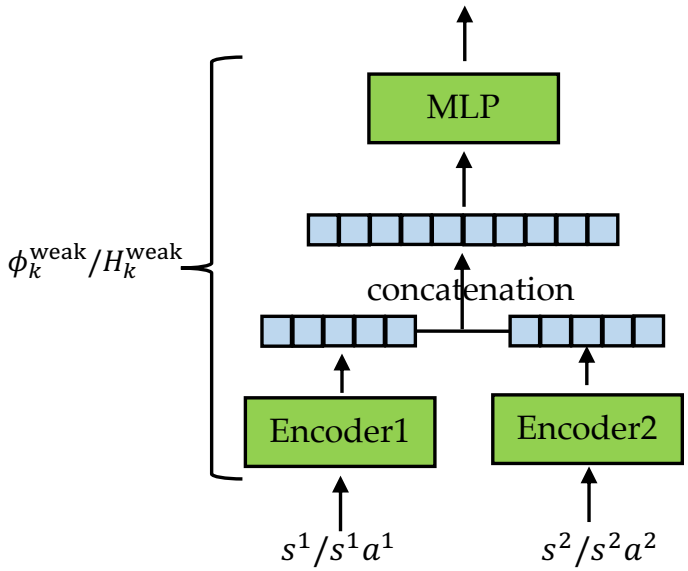


Figure 4.3: The architecture of the similarity function. The states or state-action pairs from the two agents are first mapped by their individual encoders to a shared hidden space. The hidden feature is concatenated and mapped to the similarity value with a multi-layer perceptron.

is that strictly paired data need to comprehensively assess all the aspects of the two states or state-action pairs, which is difficult to collect. On the other hand, paired abstractions only consider similarities over the abstraction of the state, which is thus easier to annotate.

We first learn a similarity function from each set of paired abstraction data, which is modeled as a neural network with a pair of states or state-action pairs as input and outputs a similarity value in $[0, 1]$. The architecture is shown in Fig. 4.3. We first map the input states from both agents to the same hidden space by their individual encoders and concatenate the two hidden features from the two input states. Then we use a fully-connected network to map the concatenated feature to the scalar similarity value. The losses for all the similarity functions are

$$\begin{aligned}
&\min_{\Phi_k^{\text{weak}}} \mathcal{L}_k^s(\Phi_k^{\text{weak}}) = \mathbb{E}_{(s^1,s^2,v^s)\sim Y_k^s} \ell(\Phi_k^{\text{weak}}(s^1, s^2), v^s) \\
&\min_{H_k^{\text{weak}}} \mathcal{L}_k^a(H_k^{\text{weak}}) = \mathbb{E}_{((s^1,a^1),(s^2,a^2),v^a)\sim Y_k^s} \ell(H_k^{\text{weak}}(s^1, a^1, s^2, a^2), v^a),
\end{aligned} \tag{4.8}$$

where ℓ takes the binary cross entropy loss to minimize the difference between the predicted and the ground-truth similarity. Then, we impose the learned similarity function as a constraint on the state map and the action maps:

$$\begin{aligned}\min_{\Phi} \mathcal{L}_s^{\text{weak}}(\Phi) &= \sum_{k=1}^{K^s} \mathbb{E}_{s^1 \in \Xi^1} \left[-\Phi_k^{\text{weak}}(s^1, \Phi(s^1)) \right] \\ \min_{H^1} \mathcal{L}_a^{\text{weak}}(H^1) &= \sum_{k=1}^{K^a} \mathbb{E}_{(s^1,a^1) \in \Xi^1} \left[-H_k^{\text{weak}}(s^1, a^1, \Phi(s^1), H^1(s^1, a^1)) \right].\end{aligned} \tag{4.9}$$

We minimize the negative similarity to ensure the states and the translated states are similar as well as the state-action pairs and the translated state-action pairs stay similar. With the above constraint, the misalignment of the learned translation model will be substantially reduced. Also, as shown in Fig. 4.2(a), paired abstractions can reduce the compounding error by reducing the translation error at each step.

Overall Loss and Algorithm. Integrating all the losses, we derive the final learning objective of our model as follows:

$$\begin{aligned}\mathcal{L}_{\text{all}} =& \lambda_0 \mathcal{L}_{\text{m\_dyn\_con}}(\Phi, H^1) + \lambda_1 \mathcal{L}_{\text{dom\_con}}(H^1, H^2) \\ &+ \lambda_2 \mathcal{L}_{\text{adv}}^a(H^1, H^2, D_{a^1}, D_{a^2}) + \lambda_3 \mathcal{L}_{\text{adv}}^s(\Phi, D_s) \\ &+ \lambda_4 (\mathcal{L}_s^{\text{weak}}(\Phi) + \mathcal{L}_a^{\text{weak}}(H^1))\end{aligned} \tag{4.10}$$

where λ_4 is the trade-off parameter for the weakly supervised loss.

Jointly optimizing all the loss functions in Eqn. (4.10) can cause the training to be unstable [52]. Thus, we first learn the forward model $\hat{\mathcal{T}}^2$ and the similarity functions $\Phi_1^{\text{weak}} - \Phi_{K^s}^{\text{weak}}$ and $H_1^{\text{weak}} - H_{K^a}^{\text{weak}}$. After the training of these networks converges, we fix their parameters. We do not fine-tune these parameters during correspondence learning since the parameters are already learned to differentiate similar or dissimilar pairs and we can use them to decide whether the mapped state/state-action pair is similar to the original state/state-action pair. Then, we iteratively train the networks related to the state map: Φ and D^s, and the networks related to the action maps: H^1, H^2, D^{a^1} and D^{a^2}. When we train Φ and D^s, we fix the parameters of H^1, H^2, D^{a^1}, and D^{a^2}, and vice versa. Such an iterative training paradigm avoids the state map

and the action maps converging to unstable solutions. When training Φ and D^s or training the action maps H^1 and H^2, and the discriminators D^{a^1} and D^{a^2}, we follow the training paradigm of adversarial networks [16].

4.5 Experiments

In our experiments, we aim to demonstrate the efficacy of WeaSCL in different correspondence learning settings including cross-morphology, cross-physics, and cross-modality, and demonstrate that WeaSCL works well with different types of paired abstractions in different environments.

We use **WeaSCL-T** to refer to our approach, where T corresponds to the final horizon at which we enforce dynamics cycle-consistency. We compare WeaSCL-T with baseline methods: **DCC** [52] and **CC**, which removes the dynamics cycle-consistency in DCC, and several variants of WeaSCL: **DCC-T** and **WeaSCL-1**, where DCC-T only adopts multi-step dynamics cycle-consistency without using paired abstractions while WeaSCL-1 uses the paired abstractions but only uses single-step dynamics cycle-consistency.

We show additional experimental details and more visualization results on our website.

4.5.1 Cross-Morphology Demonstrations

Table 4.1: Morphology parameters and dimension of state and action spaces in the HalfCheetah, Swimmer, and Ant.

Environment	Agent $\mathcal{M}^2$			Agent $\mathcal{M}^1$		
	Morphology	State	Action	Morphology	State	Action
HalfCheetah	2 legs	18	6	3 legs	24	9
Swimmer	3 links	10	2	4 links	12	3
Ant	4 legs	113	8	5 legs	135	10

Mujoco Environments. We conduct our experiments in Mujoco HalfCheetah,

Swimmer, and Ant environments under a **cross-morphology** setting, where we create different agents by varying the morphology. The morphology and the dimension of state space and action space are shown in Table 4.1. We measure the similarity of states using the x-axis location as the abstraction of the state. Since both state spaces and action spaces are different, we train both the state map Φ and action maps H^1 and H^2.

The goal of correspondence learning is to learn a translation model to leverage the optimal policy for the agent $\mathcal{M}^1$ to make decisions in the environment of agent $\mathcal{M}^2$. We evaluate the performance of the translation model by the performance of the translated policy in $\mathcal{M}^2$.

We train the similarity function for 100 epochs, the forward dynamics model for 10 epochs, and the translation model for 30 epochs. We use Adam optimizer with a learning rate 0.001 for all the model training. For the trade-offs parameters, we use $\lambda_0 = 15$, $\lambda_1, \lambda_2 = 1$ and $\lambda_3 = 10$. The trajectory pairs used for all three environments are 1,000 each.

The results are shown in Table 4.2. For both DCC and our methods, using a horizon of 5 for dynamics cycle-consistency achieves a much better performance than a horizon of 1, which demonstrates the efficacy of multi-step dynamics cycle-consistency. WeaSCL-5 and WeaSCL-1 outperform DCC-5 and DCC-1 respectively, which demonstrates the efficacy of paired abstractions.

Simulated Robots. As shown in Fig. 4.5, we create two dynamics in the simulated Panda Robot: the original 7-DoF robot arm, and a 5-DoF arm that fixes the third and fourth joints of the 7-DoF arm (shown by red crosses). We define the paired abstractions based on the end-effector position in the state (green arrows) or the joint force in the action (purple arrows). We test two settings of paired abstractions: (1) only using the end-effector position (Y^s); (2) using both the end-effector position and the joint force (Y^s and Y^a). Our goal is to translate the policy from 5-DoF to 7-DoF.

We show our results in Table 4.3. We observe that WeaSCL-5 outperforms the baselines, DCC-1 and CC. WeaSCL-5 also outperforms the variants: WeaSCL-1 and DCC-5, which demonstrates the efficacy of both kinds of weak supervision. We also note that WeaSCL-5 with Y^s and Y^a outperforms WeaSCL-5 with Y^s, which

Table 4.2: The performance of the translated policy under different morphologies in Mujoco environments.

Method	HalfCheetah	Swimmer	Ant
CC	-104.39±92.72	30.00±2.19	297.52±87.48
DCC-1	658.66±23.13	53.40±11.39	447.50±470.19
DCC-2	1005.52±44.12	64.92±5.43	669.94±72.54
DCC-3	1166.90± 50.67	71.70±3.53	762.43±1.92
DCC-5	1250.55± 51.66	65.19± 2.16	928.22± 1.96
DCC-10	1249.15±434.78	52.18±3.61	942.03± 2.61
WeaSCL-1	1284.61±109.47	69.59±13.88	969.28±1.03
WeaSCL-5	**1455.08**±63.59	**86.14**±2.46	**971.08**±2.10
Oracle	4380.75±97.30	126.19±2.42	991.56±1.98

demonstrates that WeaSCLcan handle similarities over multiple abstractions elegantly and having access to similarities over multiple types of abstractions improves the performance.

Table 4.3: The performance of the translated policy under different morphologies in the simulated robot environment.

CC	-315.09±115.74
DCC-1	-255.33±160.19
DCC-5	-233.47±103.19
WeaSCL-1 (Y^s)	-225.28±100.39
WeaSCL-1 (Y^s and Y^a)	-219.88±121.11
WeaSCL-5 (Y^s)	-78.43±22.06
WeaSCL-5 (Y^s and Y^a)	**-73.07** ±42.19
Oracle	-20.68±21.30

4.5.2 Cross-Physics Demonstrations

We conduct the experiments in Mujoco Hopper and Walker2d environments under a **cross-physics** setting, where we create different agents by varying the physical factors

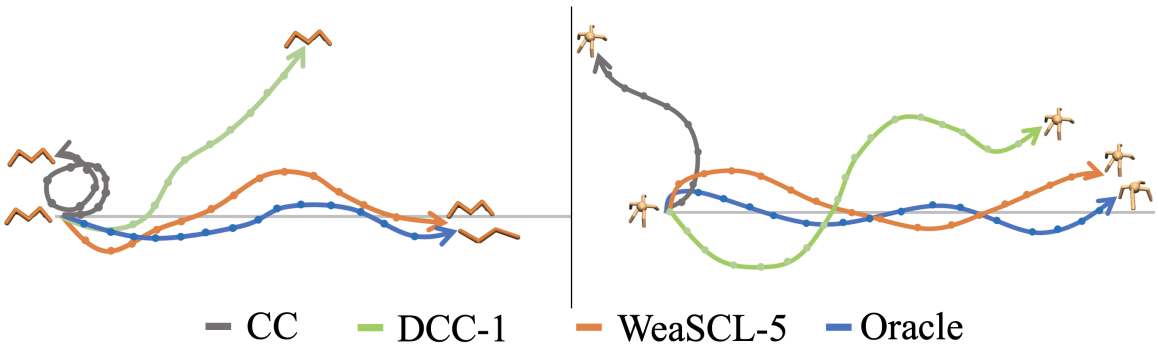


Figure 4.4: Sample trajectories for the 4-link swimmer (left) and 5-legged ant (right). The grey line is the positive x-axis, which direction the robot is supposed to move toward. The oracle is only available in $\mathcal{M}^2$ (3-link swimmer and 4-legged ant).

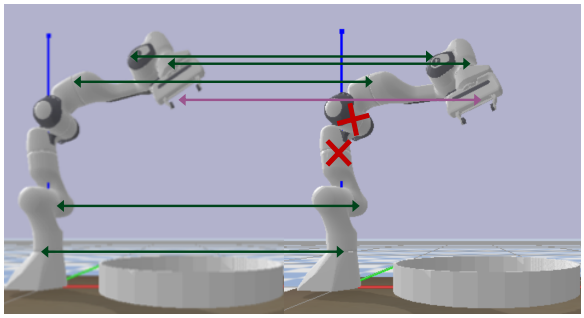


Figure 4.5: Demonstrating the two robot arms with different degrees of freedom and the paired abstractions of end-effector positions and joint forces.

in the environment. We vary the gravitational constant in the Hopper environment and vary the friction of feet in the Walker2d environment. The exact value of the gravitational constant and the friction of the agent $\mathcal{M}^1$ and $\mathcal{M}^2$ are in Table 4.4. Note that only changing the physical parameters does not change the state and action spaces but changes the transition, where taking the same action at a state will transition to potentially different states, so we only learn the action maps to align the agents. Our goal is to translate a policy across environments with different physical parameters.

We use confidence as the abstraction to define similarity, where confidence lies in $[0, 1]$ indicating how good a state, action pair is with respect to the reward function.

Table 4.4: Physical parameters in the Hopper and Walker2d.

Environment	Agent $\mathcal{M}^2$	Agent $\mathcal{M}^1$	
		Setup1	Setup2
Hopper (Gravitational Constant)	9.8	0.5	5.0
Walker2d (Friction)	0.9	9.9	19.9

For example, if a state, action pair always appears in optimal trajectories, we regard it as optimal and assign confidence 1 to it.

Here, we need trajectories with different confidence values for Ξ^1 and Ξ^2. For each environment and physical parameter, we train 7 policies with different rewards, which range from the random policy to the optimal policy. We then collect 10 trajectories from each policy as the trajectory set. We compute the reward for each trajectory and normalize the reward into $[0, 1]$ by min-max normalization, where the normalized reward is used as the confidence for each trajectory. For each state-action pair in a trajectory, we use the trajectory confidence value as the confidence used for abstraction. Then for all the state-action pairs in all trajectories, we randomly sample 1000 pairs of state-action pairs with varying similarity as the dataset to learn the similarity function.

We show the results of our method and baselines in Table 4.5. For DCC and our method, we report the results of using the dynamics cycle-consistency for $1 - 5$ steps, since the performance does not increase or even decrease for more than 5 steps. We observe that our method with a proper number of steps for dynamics cycle-consistency achieves the best reward in all the tasks. Note that in most of the tasks, only two steps of dynamics cycle-consistency are sufficient to achieve the best performance, which demonstrates that the proposed approach is computationally efficient.

4.5.3 Cross-Modality Demonstrations

We also conduct experiments on the real robot under a **cross-modality** setting, where we translate across the visual observations and the joint states of a real Franka Panda robot arm. Our goal is to predict the state of the robot (joint configurations) from

Table 4.5: The performance of the transferred or translated policy under different physics.

Method	Gravity 0.5	Gravity 5.0	Friction 9.9	Friction 19.9
Direct	269.59±2.45	335.41±7.89	290.84±10.12	280.49±20.54
CC	61.19±39.91	83.26±155.79	178.93±219.81	236.15±72.39
DR	295.64±4.87	376.31±9.41	297.32±9.42	310.18±22.24
DCC-1	26.48±45.17	6.03±4.32	305.28±7.01	375.22±101.77
DCC-2	271.59±50.02	190.08±186.22	369.07±48.11	588.70±201.02
DCC-3	234.46±217.32	229.69±243.62	302.15±7.11	540.31±143.63
DCC-4	256.91±55.10	195.03±148.78	307.50±3.04	799.97±138.63
DCC-5	276.73 ±120.39	231.76±161.27	305.11±4.62	598.56±219.53
WeaSCL-1	208.14±189.65	143.72±180.01	321.04±14.40	587.73±117.49
WeaSCL-2	**325.80**±57.06	279.33±107.24	**499.52**±48.99	**1052.62**±224.62
WeaSCL-3	137.49±132.33	**387.12** ±186.80	301.30±4.18	693.94±245.45
WeaSCL-4	129.05±84.62	283.38 ±184.15	308.94±2.39	674.47±122.79
WeaSCL-5	130.09±75.48	272.69±91.31	306.67±6.30	550.24±202.03
Oracle	1952.99±32.41	3060.55±21.72	3604.38±52.59	1632.18±22.86

the visual observation of the robot. Our abstraction here is the end-effector pose of the robot in these two domains (ground-truth state and visual observations) and we collect 100 similarity pairs to learn the similarity function. Note that the actions are the same and we just need to learn the state map Φ, which takes the RGB images as inputs and outputs the joint state of the robot. The visual observations are captured by an external RGB camera with a third-person point of view. We collect random trajectories of visual observations and joint configurations (not paired) on the robot through teleoperation to train the state map Φ.

As shown in Figure 4.6, WeaSCL-5 achieves the lowest estimation error compared to baselines CC and DCC-1 and also the variants DCC-5 and WeaSCL-1, which demonstrates the efficacy of our approach in real robot applications.

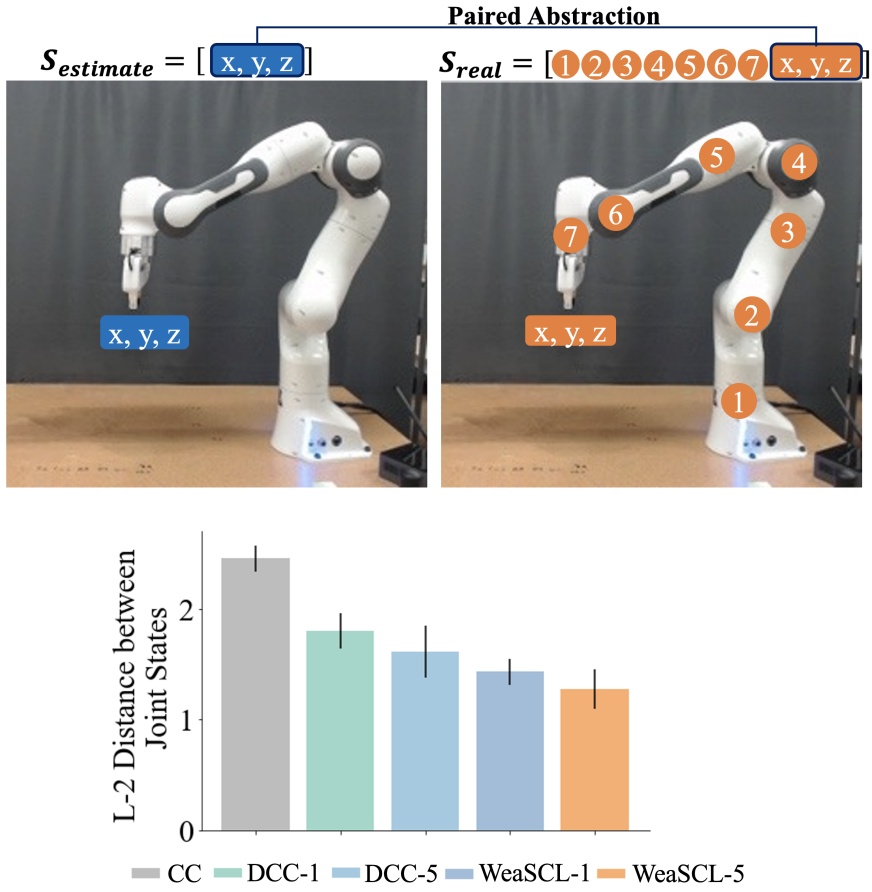


Figure 4.6: Top: An illustration of the real robot arm environments. Bottom: The norm of joint differences on the robot.

4.6 Chapter Summary

In this chapter, to address the problem of learning from cross-domain demonstrations, we aim to build correspondence between different agents. We first define correspondence learning and introduce the background on dynamics cycle-consistency, the state-of-the-art unsupervised correspondence learning method. To better trade-off between the accuracy of correspondence learning and the annotation effort, we propose a weakly supervised correspondence learning approach (WeaSCL) that leverages weak supervision in the form of temporal ordering and paired abstraction data. This eases the need for expensive paired data and enables more accurate correspondence learning.

Experiment results show that WeaSCL outperforms the state-of-the-art correspondence learning methods based on unpaired data.

Chapter 5

Feasibility Learning from Infeasible Demonstrations

5.1 Infeasible Demonstrations

In the last chapter, we learn an optimal policy for the target agent by leveraging cross-domain demonstrations from other agents. We enable the target agent to learn from cross-domain demonstrations by building correspondence between agents. However, not all the demonstrations in other agents have correspondence to the target agent. Due to the different morphology and dynamics between the other agents and the target agent, some trajectories from other agents cannot be achieved by the target agent and thus we could not build correspondence between these trajectories to the target agent. We call these trajectories infeasible demonstrations. which naturally exist in the demonstrations collected from other agents.

Imagine a setting, where a set of demonstrations are collected on a 7 Degrees of Freedom (DoF) robot arm shown in Fig. 5.1 to place a book on the empty area of the shelf (on the left) without colliding with the books that are already placed on the right side of the shelf. Later, we might decide to buy a different arm with 3 DoF (e.g., only the joints circled in green as shown in the figure are used). We would like to learn a policy for this 3 DoF robot arm that can achieve the same task—placing the book on the empty region of the shelf—using the originally collected demonstrations

on the 7 DoF arm. In general, our goal is to enable using and reusing of data collected on robots with different dynamics or embodiments to tackle the problem of lack of in-domain data in robotics. However, the red trajectories that move around the stack of books are not feasible for the 3 DoF robot arm. Imitating such trajectories may cause the 3 DoF robot arm to maximally follow these trajectories and even collide with the existing stack of books.

Therefore, it is crucial to identify and avoid trajectories that are far from feasible for the imitator, and instead learn more from useful demonstrations, e.g., the blue trajectories that go over the shelf that are still feasible for a robot with 3 DoF. To avoid the influence of useless or harmful demonstrations from agents with different dynamics, we rely on a *feasibility score*, which measures how feasible a trajectory is for the imitator, and selects trajectories with high feasibility to imitate. For example, the blue trajectories should have higher feasibility than the red trajectories in Fig. 5.1.

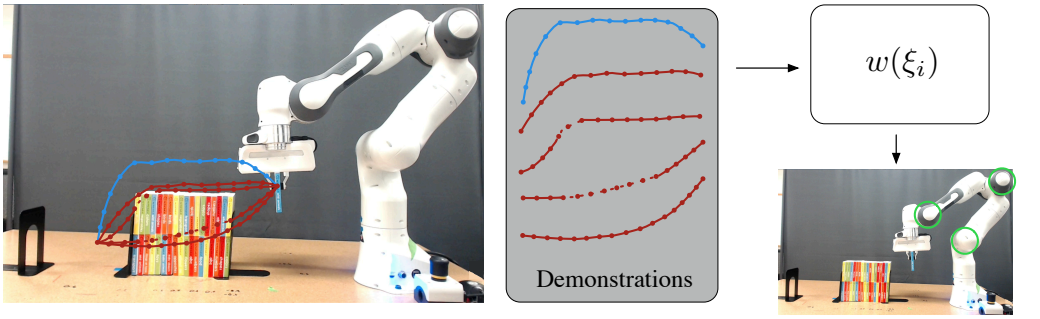


Figure 5.1: An example of imitating demonstrators with feasibility. The left image shows that a set of demonstrations (blue and red trajectories) are available for the 7 DoF robot arm. We aim to learn a policy for the 3 DoF robot (joints are circled in green) by learning from the demonstrations of the 7 DoF robot (blue is feasible and red is infeasible). We learn a feasibility score to reweight each demonstration to conduct imitation learning.

5.1.1 Problem Setting

In this problem setting, instead of learning from demonstrations collected on a single demonstrator, we aim to learn the target robot policy from demonstrations collected from N demonstrators with various dynamics. We formalize each demonstrator $j, (1 \leq$

$j \leq N$) as a standard Markov decision process (MDP):$\mathcal{M}_j^d = \langle \mathcal{S}, \mathcal{A}_j^d, \mathcal{T}_j^d, \mathcal{R}, \rho_0, \gamma \rangle$. $\mathcal{A}_j^d$ is the action space and $\mathcal{T}_j^d : \mathcal{S} \times \mathcal{A}_j^d \times \mathcal{S} \to [0, 1]$ is the transition probabilities for each demonstrator respectively.

Note that we assume that all the demonstrators and the target robot share the same state space and initial state distribution but may have different transition functions and action spaces. This is satisfied in many real-world applications. For example, we may use a camera to capture the robot, and the camera images embed both the 3-dof and 7-dof representation of the robots, along with their environment.

ρ_0 is the shared initial state distribution for all MDPs. We also make the assumption that the reward function is based on state transitions and is shared between the demonstrators and the target robot, which is a common assumption used in prior work [26, 10], and is usually satisfied since the demonstrators and the target robot conduct the same task in the same context.

Different from our problem setting in Section 2.1, now we are given a set of demonstrations collected from different demonstrators $\Xi^j = \{\xi_1^j, \ldots, \xi_D^j\}_{j \in \{1 \ldots N\}}$ where each trajectory is a sequence of states $\xi = \{s_0, s_1, \ldots, s_N\}$. We assume that *the optimal policy can be learned by imitating the useful demonstrations*, which means that if we remove all the demonstrations that are far from feasible, the remaining useful demonstrations provide sufficient knowledge to learn an optimal policy. This is a general assumption adopted by prior imitation learning works [2, 20, 26, 10]. The violation of this assumption, as shown in prior works, leads to learning a suboptimal policy. Note that *we discard actions from the demonstrations* instead of imitating the state-action trajectories because different action spaces between the demonstrators and the imitator make it impossible to imitate the actions.

Recall that there are two core challenges for imitation learning from demonstrations of other agents: (1) How to imitate useful demonstrations with different dynamics, (2) How to avoid harmful demonstrations misleading the imitator. In chapter 4, we address the first challenge by learning about the correspondence between agents. In this chapter, we aim to address the second challenge by learning a feasibility score that measures how likely it is for a demonstration to be feasible for the imitator.

5.2 Feasibility Based on Inverse Dynamics Function

Feasibility measures how likely it is for each state transition in the demonstration to be generated by the target MDP. Formally, a state transition (s_t, s_{t+1}) is *feasible* if and only if there exists an action $a_t \in \mathcal{A}$ in the target MDP $\mathcal{M}$ satisfying $\mathcal{T}(s_t, a_t, s_{t+1}) > 0$.

However, we cannot simply apply this definition to our demonstrations to verify feasibility. First, the transition probability function is usually unknown or difficult to learn in most realistic robotics tasks, and thus we cannot directly use the function to check for feasible actions. If we do not use the transition probability function, but assume access to a simulator, where we can sample the next state given a state and action – as is assumed by prior works [20, 48, 14, 35] – we would still need to search over a large space of possible actions to verify the existence of action a_t for the state transition (s_t, s_{t+1}) to be successful in the target MDP.

We instead propose computing the most probable action using an inverse dynamics model of the target MDP $f_{\text{id}} : \mathcal{S} \times \mathcal{S} \to \mathcal{A}$, which takes a state transition pair (s_t, s_{t+1}) that could be generated by either the target MDP or the demonstrator MDP as an input and outputs all the actions a_t that satisfy $\mathcal{T}(s_t, a_t, s_{t+1}) > 0$. For every ξ from the set of demonstrations provided by any demonstrator MDP, we initialize the agent at the state s_0, which is the initial state of ξ. We then make the agent take an action a'_t generated by f_{id} at each time step to generate a new state trajectory $\xi' = \{s'_0, s'_1, s'_2 \ldots, s'_N\}$ that is achievable by the target MDP:

$$\begin{aligned} &s'_0 = s_0, \quad a'_t = f_{\text{id}}(s'_{t-1}, s_t), t \geq 1 \\ &s'_t \sim \mathcal{T}(s'_{t-1}, a'_t, s'_t), t \geq 1 \end{aligned} \tag{5.1}$$

Learning Inverse Dynamics. If we assume access to an accurate inverse dynamics model f_{id} – meaning that f_{id} gives the correct action for a feasible state transition and outputs "None" for an infeasible state transition, we can detect the infeasible trajectories based on the "None" output. However, such a binary feasibility metric cannot differentiate between nearly feasible and infeasible trajectories, where the former is useful for learning and the latter does not have much learning value. So instead we aim to learn a f_{id} that outputs the correct action if the state transition is

feasible while still outputting the closest action when the state transition is infeasible. We model f_{id} as a neural network and train it on a randomly sampled set of trajectories $\Xi_f = \{\xi_f\}$ from the target MDP. We emphasize that similar to prior imitation learning works [20, 48, 14, 35], we assume access to a simulator for the target MDP, where we can sample feasible but random trajectories. We learn f_{id} using a regression loss:

$$\min_{f_{\text{id}}} \mathbb{E}_{(s_t,a,s_{t+1})\sim\xi_f,\xi_f\sim\Xi_f} L_{1;\text{smooth}}(f_{\text{id}}(s_t, s_{t+1}), a). \tag{5.2}$$

Computing the Feasibility Score. Here, the trained f_{id} will output similar actions if the input state transition is similar to a state transition seen in the training data. With the learned f_{id}, for every trajectory ξ – even if it is infeasible – we can compute a corresponding trajectory ξ'. In addition, we can compute a distance metric between the two trajectories ξ and ξ', $F(\xi, \xi')$, where lower distances correspond to higher feasibility scores. We compute the feasibility score by normalizing this distance within a range $[d_{\min}, d_{\max}]$:

$$w_f(\xi) = \begin{cases} 1 & F(\xi,\xi') < d_{\min} \\ 1 - \frac{F(\xi,\xi') - d_{\min}}{d_{\max} - d_{\min}} & d_{\min} \le F(\xi,\xi') \le d_{\max} \\ 0 & F(\xi,\xi') > d_{\max} \end{cases} \tag{5.3}$$

Here, ξ' is a corresponding trajectory to ξ generated by the inverse dynamics f_{id}. Further, we can choose $F(\xi, \xi')$ to be any sequence metric between the two trajectories. In our experiments, we computed the mean of the l_2 distance between each pair of states in ξ and ξ'. The proposed continuous feasibility score can measure different degrees of feasibility and preserve nearly feasible trajectories that can be useful for learning a policy for the target agent.

Discussion of Feasibility Score. Using our feasibility measurement, the more infeasible transitions contained in ξ are, the deviation between ξ' and ξ becomes larger, and hence it would be less likely that the trajectory is drawn from the target MDP. A possible drawback of our feasibility score is compounding errors from our inverse dynamics model f_{id}. To address this problem, we normalize the distance

between the trajectories $F(\xi, \xi')$ using $d_{\min}$ and $d_{\max}$. Here, $d_{\min}$ models the lowest compounding error that could be achieved by a feasible trajectory, while $d_{\max}$ is the highest compounding error. Thus, our feasibility measure can assign non-zero feasibility to any feasible trajectories even with compounding errors from the inverse dynamics.

To set two threshold values, we design the method as follows. First, $d_{\min}$ sets the minimum distance between ξ and ξ', and any trajectory with distance smaller than $d_{\min}$ will be assigned feasibility of 1. We set $d_{\min}$ by taking the minimum in the sampled feasible trajectories Ξ_f: $d_{\min} = \min_{\xi_f \in \Xi_f} F(\xi_f, \xi'_f)$.

Second, $d_{\max}$ sets the maximum distance between ξ and ξ' that can be tolerated. There are two clear approaches for computing $d_{\max}$: the maximum distance in the demonstration trajectories $\max_{\xi \in \Xi} F(\xi, \xi')$ and the maximum distance in the sampled feasible trajectories $\max_{\xi_f \in \Xi_f} F(\xi_f, \xi'_f)$. Here, ξ' and ξ'_f are computed using the inverse dynamics as in Eqn. (1) in the main text. The former can be too large and can assign positive feasibility to harmful and completely infeasible trajectories at times. On the other hand, the latter may filter nearly feasible trajectories, which are useful for imitation learning. Instead of these two extreme measures, we propose estimating $d_{\max}$ from the environment. For every trajectory ξ_f in Ξ_f, we derive a trajectory $\xi''_f = \{s''_0, s''_1, \cdots, s''_N\}$:

$$\begin{aligned} & s''_0 = s_0, \quad a''_t = f_{\text{id}}(s''_{t-1}, s_t), t \geq 1 \\ & s''^{\text{sample}}_t \sim \mathcal{T}(s''_{t-1}, a''_t, s''_t), t \geq 1 \\ & s''_t = s''^{\text{sample}}_t + \Delta s. \end{aligned} \tag{5.4}$$

We perturb the state at each step with a Δs, where $|\Delta s| \leq \delta$, and then we set $d_{\max}$ as the max distance $\max_{\xi_f \in \Xi_f} F(\xi_f, \xi''_f)$. We note that ξ'_f is the corresponding trajectory for ξ while ξ''_f is a trajectory with perturbation δs over the states of ξ. This modeling allows us to know that a distance smaller than $d_{\max}$ means the trajectory is within δ state averaged distance from a feasible trajectory. We can control δ in different environments to decide if a trajectory should be assigned positive or zero feasibility weights.

5.3 Feasibility Learned by Feasibility-MDP

The method to learn feasibility in Sec. 5.2 requires the inverse dynamics function, which is not always available for different environments. Even though the inverse dynamics function could be learned from transitions in the target environment, it is hard to collect sufficient transitions to cover the whole state-action space. Thus, we develop a feasibility-MDP to learn the feasibility only by interacting with the target environment without other requirements. Specifically, we model the imitator environment as an MDP and build a feasibility Markov Decision Process (f-MDP) based on the imitator's MDP and the trajectories provided by the demonstrator. The optimal policy for the f-MDP maximally follows the behavior of the demonstrations but is limited by the imitator's environment. This optimal policy helps assign a feasibility score over the demonstrations.

The feasibility of a trajectory depends on the feasibility of each state transition in the trajectory, i.e., if (s_t, s_{t+1}) is feasible for all time steps. A state transition (s_t, s_{t+1}) is feasible when there exists an action $a_t \in A$ such that $\mathcal{T}(s_t, a_t, s_{t+1}) = 1$ for deterministic transitions or $\mathcal{T}^i(s_t, a_t, s_{t+1}) > 0$ for stochastic transitions. In this section, we discuss the deterministic MDP setting and discuss the stochastic setting in Appendix.

Feasibility can be directly measured by a perfect inverse dynamics model $f : \mathcal{S} \times \mathcal{S} \to \mathcal{A}$ that takes a state transition $(s_t, s_{t+1}) \in \mathcal{S} \times \mathcal{S}$ as the input and outputs the action $a_t \in \mathcal{A}$ that achieves the transition if feasible or outputs 'Infeasible'. However, having access to this model is often non-trivial and such a binary feasibility measurement as f discards all infeasible demonstrations without considering any useful information from slightly infeasible trajectories.

Our goal is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ for the imitator to maximally achieve the state transitions in the demonstrations. This means that if the state transition (s^d_t, s^d_{t+1}) from a demonstration is feasible, the next state produced by π, i.e., $s_{t+1} = \mathcal{T}(s^d_t, \pi(s^d_t))$ should be equal to s^d_{t+1}. Otherwise, we would like the policy to output an action that ensures the next state s_{t+1} is as close as possible to the next state from the demonstration s^d_{t+1}. Therefore, the distance between s_{t+1} and s^d_{t+1} can serve as a

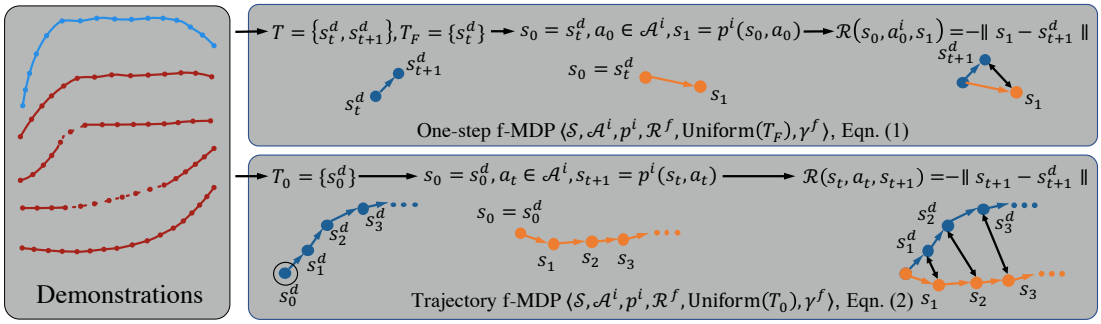


Figure 5.2: The illustration and comparison of one-step f-MDP and trajectory f-MDP. The blue state transition and trajectory are from the demonstrations while the orange state transition and trajectory are rollouts in the f-MDP. One-step f-MDP collects the states in the *Former Set* and uses the uniform distribution over the states as the initial state distribution. Trajectory f-MDP collects the initial state of all the demonstrations and uses a uniform distribution over them as the initial state distribution.

measure of feasibility, where a smaller distance corresponds to a higher likelihood of feasibility. To learn this policy, we design a feasibility MDP (f-MDP), where we ensure that the optimal policy of the f-MDP satisfies the above requirement. f-MDP is defined as $M^f = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}^f, \rho_0^f, \gamma^f \rangle$. We will now discuss our choices for the components of f-MDP.

One-step f-MDP. First, recall that our goal is to learn a policy *for the imitator* to *maximally achieve the state transitions in the demonstrations*. So the policy should be learned in an environment with the same state-action space and transition probability as the imitator. We would like the reward of the f-MDP to encourage maximally achieving the state transitions in the demonstrations. Let us first collect all the state transitions $T = \{(s_t^d, s_{t+1}^d)\}$ in all of the demonstrations. We define the *Former Set* to be the set of states in the demonstrations that one can transition from: $T_F = \{s_t^d : (s_t^d, s_{t+1}^d) \in T\}$. The initial state distribution ρ_0^f can be defined uniformly over the Former Set as $\text{Uniform}(T_F)$. Here, we assume that all the states in the Former Set can be visited by the imitator. We define the reward of a *One-step f-MDP* so that it matches the one-step transitions from the Former Set:

$$s_t^d \sim \text{Uniform}(T_F), \quad s = s_t^d, \quad s' = \mathcal{T}(s, a), \quad R^f(s, a, s') = -f_{\text{dis}}(s', s_{t+1}^d), \tag{5.5}$$

where (s_t^d, s_{t+1}^d) is a state transition in the demonstrations and $a \in A$ is sampled from the action space of the f-MDP. f_{dis} is a function that measures the distance between the states (e.g., the L2 distance). We define the reward to penalize the distance between s' and s_{t+1}^d.

Trajectory f-MDP. The one-step f-MDP suffers from an important shortcoming: the assumption that all states in the Former Set must be visited by the imitator can be violated because the demonstrators have different dynamics from the imitator and some demonstration states can never be reached by the imitator. So we cannot set Uniform(T_F) as the initial state distribution for the f-MDP. We instead collect the initial state s_0^d of all the demonstrations, $T_0 = \{s_0^d\}$, and define the initial state distribution of the *Trajectory f-MDP* as Uniform(T_0). Since all the demonstrators and the imitator share the initial state distribution, all states in T_0 can be visited by the imitator. We define the reward as:

$$s_0^d \sim \text{Uniform}(T_0), \quad s_0 = s_0^d, \quad s_{t+1} = \mathcal{T}(s_t, a), \quad R^f(s_t, a, s_{t+1}) = -f_{\text{dis}}(s_{t+1}, s_{t+1}^d), \tag{5.6}$$

We use the L2 distance for f_{dis} in our experiments. Similar to the one-step f-MDP $a \in A^i$ is sampled from the action space of the imitator.

Here we use L2 distance between states in the reward function but this does not mean that L@ distance is not the only choice. We can change the distance depending on the specific state space. For example, for a state space with unit vectors, we can use cosine distance as the distance metric. We investigate the influence of different distance functions in Sec. 5.4.3 in the Appendix.

Learning Feasibility. Given the Trajectory f-MDP defined above, for each demonstration trajectory ξ, the highest reward achieved in this f-MDP reflects the feasibility score of the trajectory. We use reinforcement learning to learn the optimal policy of the Trajectory f-MDP, π^*. We then derive the feasibility of each demonstration trajectory ξ as a function of the trajectory f-MDP reward:

$$w(\xi) = \exp\left(\frac{-\sum_{t=1}^{N}(\gamma^f)^t f_{\text{dis}}(s_t, s_t^d) - C}{\sigma}\right). \tag{5.7}$$

s_t is the state at step t in the rollout derived by the policy π^*. We use an exponential function of the cumulative reward since the cumulative reward is always negative and the exponential function can bound the feasibility in the range of $[0, 1]$. The parameter C is used to shift the function to avoid the situation where the cumulative reward is extremely negative, while the parameter σ controls how low the reward can be, and when a demonstration can be fully filtered out by assigning a feasibility of close to 0. In practice, C is usually set as the maximal cumulative reward over all demonstrations to ensure the maximal feasibility is 1.

For the feasibility of each state transition (s_t^d, s_{t+1}^d), we use the feasibility of the trajectory it belongs to: $w((s_t^d, s_{t+1}^d)) = w(\xi_i)$, where $(s_t^d, s_{t+1}^d) \in \xi_i$. We do not use the state distance at each time step between s_{t+1} and s_{t+1}^d as in the One-step f-MDP because such measurement suffers from the fact that within a trajectory, the reward of later steps is influenced by former steps. For example, if s_t diverges from s_t^d, s_{t+1} will diverge more from s_{t+1}^d. So the per-step reward is an unfair measure of feasibility for the state transition (s_t^d, s_{t+1}^d) at different time steps t. Therefore, we use the accumulative reward of the whole trajectory as our feasibility measure, where all the state transitions share the same value.

The discount factor γ^f is usually set as $\gamma^f < 1$ to reduce compounding errors. Specifically, the length of a rollout in the f-MDP is the same as the corresponding demonstration, which can be very long. If the state in the rollout starts to diverge from the demonstration trajectory at t, meaning that $\|s_t - s_t^d\| > 0$, the steps after time step t even diverge more from the demonstration. This makes the trajectory reward for all the infeasible trajectories very low and does not allow for discriminating among different infeasible trajectories. Therefore, we set a discount factor of $\gamma^f < 1$ to discount or even ignore the trajectory reward at later steps.

Leveraging our Trajectory f-MDP design, feasible trajectories still receive the maximal reward of 0 since each state in the rollout will perfectly match the demonstration thus having a feasibility of 1 as in Eqn. (5.7). Instead, infeasible trajectories receive negative rewards leading to smaller feasibility scores, which reflect how far away the demonstration is from the closest feasible trajectory.

One may worry about the time complexity of our approach since we need additional

RL training to learn an optimal policy for the f-MDP. However, the f-MDP is a lot simpler compared to the imitator's MDP since the initial distribution is reduced from the distribution of all possible states in the demonstration set to a discrete distribution over the initial states of the demonstrations. This can simplify the time complexity of finding the optimal policy for the f-MDPs.

Algorithm. Using the feasibility metric in Eqn. (5.7), we assign each state transition with the same feasibility of the trajectory it belongs to. Directly weighing the imitation loss as [48] may lead to gradients that are close to 0 if a batch of data all has low feasibility. This can make the algorithm inefficient by wasting samples from many iterations. Instead, for more efficient training, we define a discrete probability distribution p_w over the collection of state transitions in all the demonstrations: T, where the probability of a state-transition (s_t^d, s_{t+1}^d) as $p_w((s_t^d, s_{t+1}^d)) = \frac{w((s_t^d, s_{t+1}^d))}{\sum_{\left(s_{t'}^d, s_{t'+1}^d\right) \in T} w((s_{t'}^d, s_{t'+1}^d))}$. State transitions with larger feasibility will be sampled more often. Using the sampling distribution p_w, we can embed our method into any imitation learning algorithm to enable learning from demonstrations with different dynamics.

Sampling More Demonstrations with the Feasibility Score. When the existing useful demonstrations are too scarce to learn a well-performing imitation learning policy, we need to acquire more demonstrations from the demonstrators. But collecting new demonstrations can be expensive, so we often can only acquire a limited budget of demonstrations. We thus need to collect the most useful demonstrations within this limited budget. The proposed feasibility metric provides a criterion to decide the similarity between the imitator and each demonstrator. If a demonstrator has a higher similarity, we sample more from this demonstrator because its demonstrations are more likely to be feasible. Specifically, we create a probability distribution p_j over all demonstrators:

$$p_j = \frac{\frac{1}{|\Xi^j|} \sum_{\xi^j \in \Xi^j} w(\xi^j)}{\sum_{j=1}^{N} \frac{1}{|\Xi^j|} \sum_{\xi^j \in \Xi^j} w(\xi^j)}. \tag{5.8}$$

We repeatedly and independently query the demonstrator j according to p_j and collect a demonstration. The proposed sampling strategy samples more demonstrations from closer demonstrators. We empirically show that the sampling strategy derived from our feasibility performs better than uniform sampling or using other feasibility metrics

as in [10].

Choice of Discount Factor The choice of the discount factor γ^f depends on how fast the compounding error increases with respect to the number of steps in the environment. Faster increasing compounding errors need smaller γ^f. In practice, this depends on the scale of the 'movement' of the agent at each step. For example, for a robot arm, if each joint can only move at a small angle at each step, we can set γ^f to be larger or if each joint can move at a larger angle at each step, γ^f should be set smaller. In our experiments, we fix the $\gamma^f = 0.9$, and empirically it works well for all the environments.

The Extension to Stochastic MDPs In our main text, we discussed the deterministic MDP setting as all our experiments are in the deterministic setting, but here we would like to extend the discussion to the stochastic MDP case. In a stochastic MDP, our goal is still to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ for the imitator to maximally achieve the state transitions in the demonstrations. This means that if the state transition (s_t^d, s_{t+1}^d) from a demonstration is more likely to be feasible, the expected distance between the next state s_{t+1} produced by π and s_{t+1}^d should be small, i.e., $\mathbb{E}_{s_{t+1}\sim\mathcal{T}(s_t^d,\pi(s_t^d))}[f_{\text{dis}}(s_{t+1}, s_{t+1}^d)]$ should be small. Therefore, the expected distance between s_{t+1} and s_{t+1}^d can serve as a measure of feasibility, where a smaller distance corresponds to higher feasibility.

Under a feasibility metric defined by the expected distance, we can use the same design of f-MDP as the deterministic MDP case: $M^f = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}^f, \rho_0^f, \gamma^f \rangle$. The state space, the action space, and the transition probability are all the same as the imitator's. The initial state distribution is defined as Uniform(T_0). The reward is defined as:

$$s_0^d \sim \text{Uniform}(T_0), \quad s_0 = s_0^d, \quad s_{t+1} = \mathcal{T}(s_t, a), \quad R^f(s_t, a, s_{t+1}) = -f_{\text{dis}}(s_{t+1}, s_{t+1}^d). \tag{5.9}$$

Maximizing the expected return in the f-MDP in such a design will minimize the expected state distance between the learned policy and the demonstrations, which matches our definition of feasibility. After we learn the optimal policy π^* for the f-MDP, we can derive the feasibility for each trajectory ξ with the expected state

distance between the demonstration and the policy:

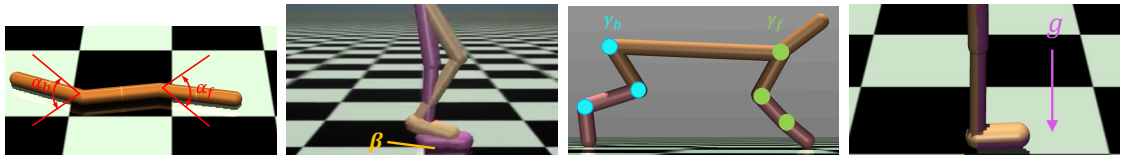
$$w(\xi) = \exp\left(\frac{-\mathbb{E}_{s_t\sim\pi^*}\left[\sum_{t=1}^{N}(\gamma^f)^t f_{\text{dis}}(s_t, s_t^d)\right] - C}{\sigma}\right). \tag{5.10}$$

In our experiments, we only consider the case where the MDP is deterministic.

5.4 Experiments

We experiment with four MuJoCo environments, a simulated Franka Panda Arm, and a real Franka Panda Arm. We also show results on various compositions of demonstrations of different dynamics and the performance gain when we are given a larger budget to collect demonstrations. We compare our approach with a standard imitation learning algorithm: GAIL [20], imitation learning from demonstrations with different dynamics methods without a measure of feasibility: SAIL [26], and with a feasibility score: ID-Feas [10], which uses an inverse dynamics model to estimate feasibility.

5.4.1 MuJoCo Experiments



(a) Swimmer (b) Walker2d (c) HalfCheetah (d) Hopper

Figure 5.3: Illustration of different dynamics in (a) Swimmer: varying the joint limit of the front and back joints (α_f and α_b). (b) Walker2d: varying the friction of the feet (β). (c) HalfCheetah: varying the joint control force of the front and back joints by multiplying a factor γ_f and γ_b with the front and back joint force. (d) Hopper: varying the gravitational constant respectively.

Swimmer. The swimmer agent has three links and two joints. The goal of the agent is to move forward by rotating the joints. As shown in Fig. 5.3(a), we create

different dynamics by setting the joint limit of the front and the back joints, denoted by $(\alpha_f,\ \alpha_b)$. The original Swimmer environment has $(\alpha_f, \alpha_b) = (100°, 100°)$. We create four demonstrator environments (α_f, α_b): (i) $(100°, 12°)$, (ii) $(100°, 20°)$, (iii) $(100°, 100°)$, and (iv) $(10°, 100°)$. We also create the imitator environment by setting $(\alpha_f, \alpha_b) = (100°, 10°)$. The demonstrators (i) and (ii) are closer to the imitator in terms of their dynamics, while the demonstrators (iii) and (iv) are farther.

Walker2d. The Walker2d is an agent with two legs where each leg consists of 3 joints. We create different dynamics by using different frictions β for the feet, i.e., the link that touches the ground. The original Walker2d uses $\beta = 0.9$. We create two settings to show the high friction and low friction of the imitator with a mix of frictions for the demonstrators. In the first setting, there are four demonstrators: (i) $\beta = 19.9$, (ii) $\beta = 9.9$, (iii) $\beta = 0.9$, and (iv) $\beta = 0.7$. The imitator has $\beta = 24.9$. In the second setting, there are four demonstrators: (i) $\beta = 29.9$, (ii) $\beta = 19.9$, (iii) $\beta = 1.1$, and (iv) $\beta = 0.7$. The imitator has $\beta = 0.9$.

HalfCheetah. The HalfCheetah is an agent with two legs at the front and back of the body, where each leg consists of three joints. We create different dynamics by varying the control force limit of joints of the front leg and back leg, where we multiply a factor γ_f with the original control force limit of the front leg and multiply γ_b with the limit of the back leg. We create two settings, where the demonstrators have low and high similarities with each other. In the first setting, there are four demonstrators with (γ_f, γ_b): (i) $(0.05, 1)$, (ii) $(0.5, 1)$, (iii) $(1, 0.5)$, and (iv) $(1, 0.05)$. The imitator has $(\gamma_f, \gamma_b) = (0.01, 1)$. In the second setting, there are four demonstrators with (γ_f, γ_b): (i) $(0.01, 1)$, (ii) $(0.05, 1)$, (iii) $(1, 0.05)$, and (iv) $(1, 0.01)$. The imitator has $(\gamma_f, \gamma_b) = (0.01, 1)$.

Hopper. The Hopper is an agent with one leg consisting of 3 joints. We create different dynamics by using different gravitational constants g. The original Hopper uses $g = 9.81$. We create four demonstrator environments: (i) $g = 20.0$, (ii) $g = 9.81$, (iii) $g = 5.0$, and (iv) $g = 2.0$. We also create the imitator environment by setting $g = 15.0$.

The detailed composition of demonstrations for all four environments is in Table 5.1. We design the composition of demonstrations to ensure that directly performing

Table 5.1: The composition of demonstrations for each environment

Environment	Number of Demonstrations			
	i	ii	iii	iv
Swimmer	50	50	500	500
Walker2d (first setting)	50	50	500	500
Walker2d (second setting)	500	500	10	10
HalfCheetah (first setting)	25	25	500	500
HalfCheetah (second setting)	500	500	25	25
Hopper	50	50	500	500

imitation learning on all the demonstrations cannot learn an optimal policy as otherwise, we do not need to consider the problem of learning from demonstrations from agents with different dynamics.

Implementation Details. To implement our algorithm, we use TRPO [39] as the RL algorithm to learn the optimal policy from f-MDP, and we use the GAIL from Observation algorithm [46] as our imitation learning technique to learn the optimal policy from the reweighted demonstrations. For each demonstration, we create a separate f-MDP for its demonstrations and train an optimal policy for the f-MDP to generate the feasibility for its demonstrations.

Compared to the final imitation learning algorithm, which requires about 7×10^7 interactive steps with the environment to converge, learning the optimal policy from f-MDP only needs about 5×10^5 time steps, which is significantly smaller. This indicates that the proposed feasibility learning is efficient even with an additional RL learning process.

For all the Mujoco environments, we evaluate the expected return of each policy by rolling out 100 trajectories in the environment with the policy and compute the average expected return of the 100 trajectories. We run each experiment for 5 times and show the mean and the standard deviation.

Results. We show the expected return with respect to the number of steps for the four different environments in Fig. 5.4. We show the results of the second setting for the Walker2d and the HalfCheetah in the Appendix. We observe that our proposed feasibility achieves the best performance among all the methods. The highest p-value

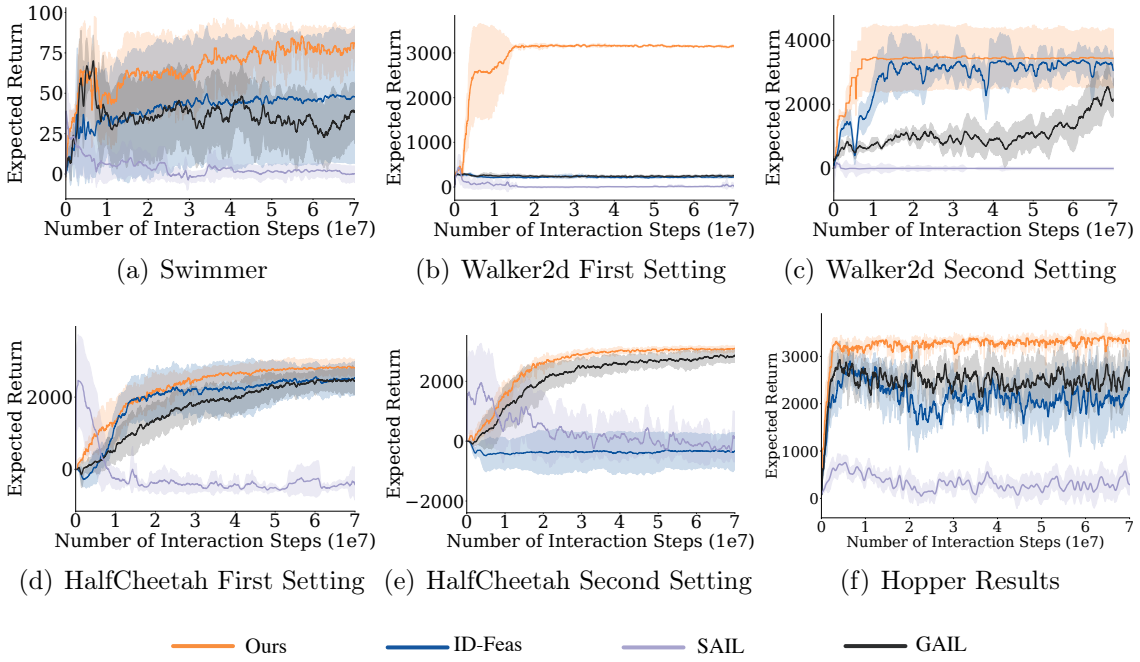


Figure 5.4: The expected return of the four MuJoCo environments.

comparing our method to baselines is 0.116 with ID-Feas for Swimmer, $2.55e-14$ with GAIL for Walker2d First Setting (statistically significant), 0.297 with ID-Feas for the Walker2d Second Setting, 0.188 with ID-Feas for HalfCheetah First Setting, 0.0037 with ID-Feas for the HalfCheetah Second Setting (statistically significant) and 0.026 with GAIL for Hopper (statistically significant). In particular, our method outperforms ID-Feas, which indicates that the proposed feasibility more accurately filters out far from feasible demonstrations. SAIL performs even worse than GAIL, this is because SAIL can more strictly follow the state sequences of demonstrations than GAIL including those far from feasible demonstrations. Our demonstration set is composed of a high percentage of demonstrations from unrelated dynamics, which can mislead SAIL's learned policy.

5.4.2 Simulated and Real Robot Arm Experiments

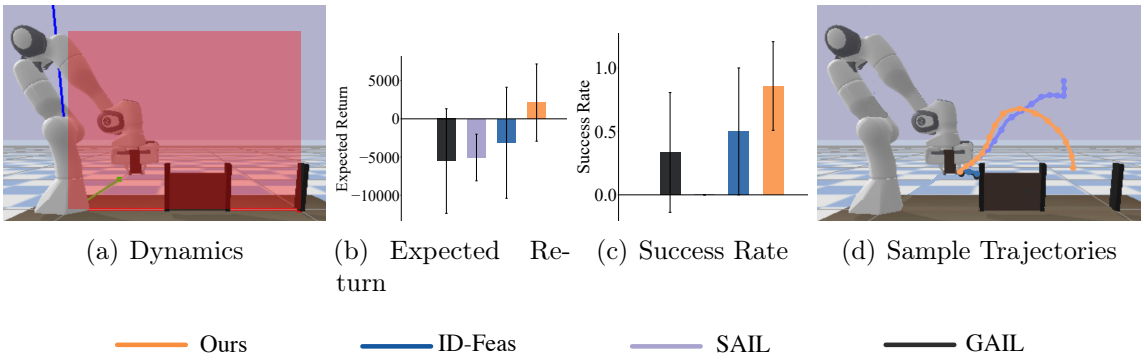
Setup. We create a simulated robot arm based on a Panda Robot Arm implemented in the PyBullet [12] and a real robot arm environment using a Franka Panda Arm[1]. As shown in Fig. 5.6(a), we create a task of moving a book to the shelf but the closest region on the shelf is full. So we need to move the book to the empty area of the shelf without colliding with the shelf and the existing books on the shelf. We create two dynamics for the robot arm: using a 7-DoF control which can move freely in the 3D space, and using a 3-DoF control, which is limited to moving on the red plane area. We collect demonstrations from both 7-DoF and 3-DoF controllers and aim to learn an optimal policy for the 3-DoF robot. The demonstration set is composed of 5 trajectories from the 3 DoF Panda robot arm with disabled joints and 43 trajectories from the 7-DoF Panda arm for both the simulated and the real arm environments.

Implementation Details. To implement our algorithm, we use TRPO [39] as the RL algorithm to learn the optimal policy from f-MDP. To learn the optimal policy from the reweighted demonstrations, we learn a beta-VAE [19] to imitate the state transition sampled from the reweighted distribution of state transitions p_w. After learning the state transitions, we recover the joint actions from the changes in the end-effector's position using inverse kinematics.

Compared to learning the beta-VAE model, which requires about 2.56×10^4 interactive steps with the environment to converge, learning the optimal policy from f-MDP only needs about 5.12×10^3 time steps, which is negligible with respect to the imitation learning algorithm. This indicates that the proposed feasibility learning is efficient even with an additional RL learning process.

For evaluation, we use two metrics: (1) The expected return based on a reward: $r = -s - 10000h + 5000g$, where r is the reward, s is the number of steps, $h \in \{0, 1\}$ represents whether the robot hits any object in the environment, and $g \in \{0, 1\}$ represents whether the robot achieves the goal. The reward penalizes collision with the shelf and existing books while rewarding the successfully moving the book to the empty area of the shelf within the time limit. (2) The success rate of finishing the

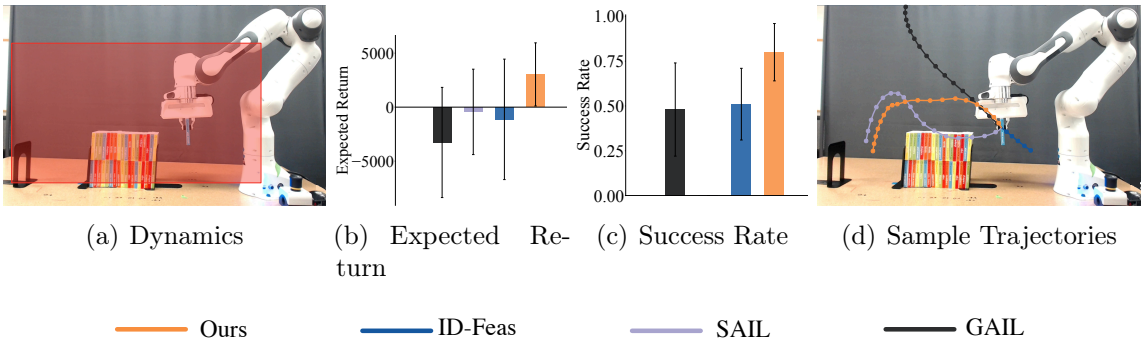
[1] https://www.franka.de



(a) Dynamics (b) Expected Return (c) Success Rate (d) Sample Trajectories

Figure 5.5: (a) The illustration of different dynamics in the simulated robot arm environment. The 7 DoF robot arm can move in the whole 3D space while the 3 DoF arm can only move in the red plane; (b-c) The bar plot for the expected return and the success rate for the simulated robot arm environment; (d) Sampled trajectories for different methods in the simulated robot arm environment.

task over 100 trials.



(a) Dynamics (b) Expected Return (c) Success Rate (d) Sample Trajectories

Figure 5.6: (a) Illustration of different dynamics in the real robot arm environment. (b-c) The bar plots show the expected return and success rate compared to other methods. (d) Sampled trajectories using different methods.

Results. We observe that the proposed approach outperforms the baseline methods both in expected return and success rate as shown in Fig. 5.5 and 5.6. For the simulated environment, the highest p-value for the expected return is 7.252×10^{-9} and the success rate is 1.047×10^{-8} (both with ID-Feas), which are both statistically significant. For the real robot environment, the highest p-value for the expected return

is 2.432×10^{-7} and the success rate is 3.534×10^{-8} (both with ID-Feas), which are statistically significant. The sampled trajectories in Fig. 5.5(d) show that the proposed approach achieves an efficient trajectory successfully moving the book to the empty area of the shelf.

5.4.3 Analysis

We conduct experiments with varying compositions of demonstrations and investigate the performance of different approaches when we have the budget to acquire additional demonstrations. We show the results of varying the number of demonstrations from all demonstrators in the Appendix.

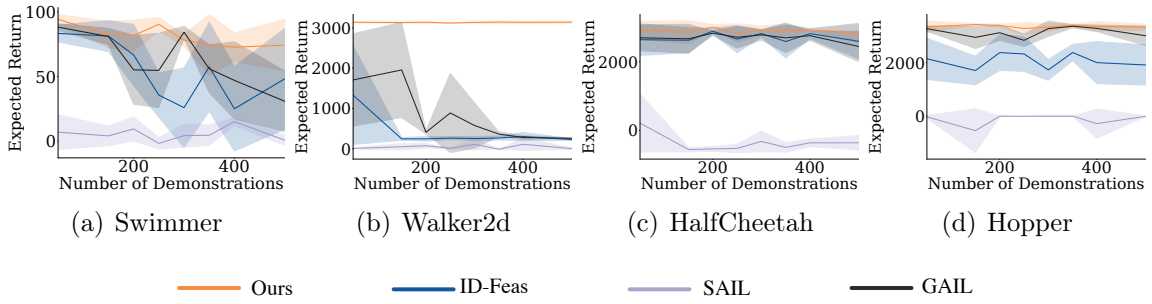


Figure 5.7: (a-d) The expected return when increasing the number of demonstrations from agents with unrelated dynamics. The results in Fig. 5.4 correspond to using 500 demonstrations from each unrelated dynamics. In both of these settings, there will also be a fixed number of demonstrations from agents with related dynamics as shown in Appendix.

Varying the Number of Demonstrations from each Unrelated Demonstrator. For the first three experiment settings in the Mujoco environment, we have two demonstrators with similar dynamics to the imitator and two demonstrators with far apart dynamics. We vary the number of demonstrations from the far apart demonstrators to investigate their influence on the different methods. We conduct experiments on the first setting for the Swimmer, Walker2D, HalfCheetah, and Hooper and report the results in Fig. 5.7(a), 5.7(b), 5.7(c) and 5.7(d). With an increasing number of demonstrations from the far apart demonstrators, the expected return of all the methods decreases, while our method shows the best performance consistently

across different numbers of demonstrations. This demonstrates that our feasibility can effectively filter out far from feasible demonstrations and ensure the policy learns from useful demonstrations.

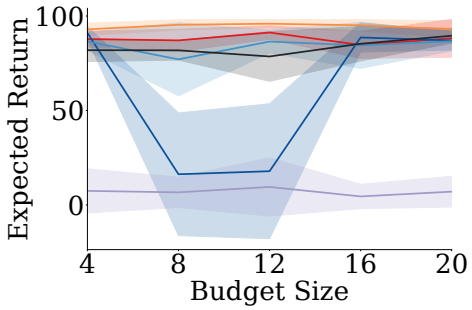


Figure 5.8: The expected return with a varying budget of additional demonstrations in Swimmer.

Performance with a Budget of Additional Demonstrations. We now consider a setting, where we start with a limited set of demonstrations, but acquire more demonstrations under a limited budget. Our feasibility metric can assess how likely it is for a demonstrator to produce feasible demonstrations, and hence can help us select which demonstrator to query for more demonstrations. We start with one demonstration from each demonstrator in the Swimmer environment and evaluate the performance as we add demonstrations. For our method and ID-Feas, we can acquire demonstrations proportional to the computed feasibility score. We compare the expected return with demonstrations selected based on feasibility (Ours, ID-Feas) to the expected return with demonstrations uniformly acquired from each demonstrator (Ours-Uniform, ID-Feas-Uniform). We further compare with SAIL and GAIL, where no feasibility is defined and we uniformly acquire demonstrations. As shown in Fig. 5.8, Ours outperforms ID-Feas, which demonstrates that the proposed feasibility can better reflect how likely each demonstrator produces feasible demonstrations and acquire more demonstrations from helpful demonstrators. Ours outperforms all the other methods including Ours-Uniform (although not with statistical significance), which indicates that the demonstrations acquired based on the feasibility gain more useful information.

Table 5.2: The performance of varying percentages of demonstrations used with respect to the original setting for Swimmer and Walker2d First Setting.

Method	Swimmer			Walker2d		
	20%	50%	100%	20%	50%	100%
GAIL	40.0±34.3	37.9±35.2	30.9±23.5	276.9±18.3	313.1±63.0	261.1±5.6
SAIL	-7.7±19.7	-5.5±24.6	-3.0±28.4	8.5±17.5	-5.3±56.4	19.0±30.1
RAL	23.1±33.9	30.6±28.1	48.2±39.0	244.3±27.4	261.0±46.7	227.0±24.2
Ours	**68.2**±24.7	**76.4**±22.1	**74.3**±20.1	**3137.6**±50.2	**3127.0**±30.7	**3144.3**±23.5

Table 5.3: The performance of varying percentages of demonstrations used with respect to the original setting for HalfCheetah First Setting and Hopper.

Method	HalfCheetah			Hopper		
	40%	60%	100%	20%	50%	100%
GAIL	2464.9±460.2	2597.2±399.0	2443.6±440.7	2798.3±351.1	2996.6±623.2	3009.6±362.4
SAIL	-556.7±365.8	-503.3±299.3	-603.0±389.6	-252.6±432.6	-1622.2±1780.1	-7.00±11.4
RAL	2604.4±423.1	2515.3±311.0	2594.4±508.7	2040.3±408.3	2108.9±611.9	1916.1±750.4
Ours	**2716.7**±301.6	**2812.4**±261.2	**2830.6**±292.6	**3273.3**±180.2	**3351.0**±146.3	**3329.6**±115.2

Varying the Number of All Demonstrations We vary the number of demonstrations from all demonstrators. We conduct experiments on the first setting of all the Mujoco environments. For the Swimmer and Walker2d environments, we test with 20% and 50% of the original demonstrations. For the HalfCheetah environment, we test with 40% and 60% of the original demonstrations since we have much fewer demonstrations (25 vs 50) from the demonstrators similar to the imitator. As shown in Table 5.2 and 5.3, we observe that our approach outperforms all the other methods when having access to a different number of demonstrations.

Table 5.4: The expected return of the learned policy in the Swimmer environment (with standard deviation).

Method	Expected Return
GAIL [20]	31.20±22.25
SAIL [26]	0.56±4.27
DCC [52]	5.32±3.43
ID-Feas [10]	48.96±38.50
Ours	74.89±19.68

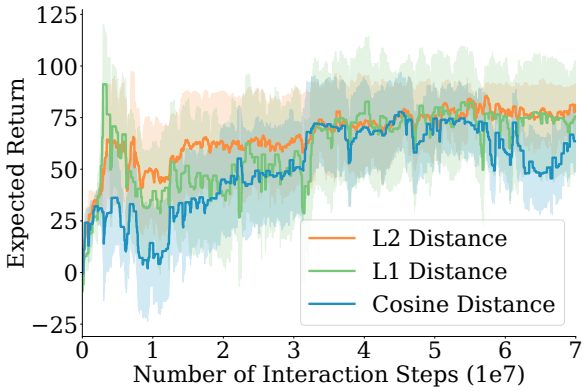


Figure 5.9: The expected return with respect to the number of steps with different choices of distance metrics.

Discussion of the Choice of Distance Functions We use L2 distance between states in the reward function in Eqn. (5.5) and (5.6) in the main text and in all the experiments. This is because, in all our environments, the L2 distance can accurately measure the distance between states. However, this does not mean that the distance metric in our reward of f-MDP is restricted to the L2 distance. We can change the distance depending on the specific state space. For example, for a state space with unit vectors, we can use cosine distance as the distance metric.

In Fig. 5.9, we show the expected return of our method by using different distances in the Swimmer environment. We use L1 distance and Cosine distance (the cosine of the angles between two state vectors) as examples. We observe that L1 distance, which is another distance derived by norm, performs close to L2 distance, but Cosine distance performs worse than L2 distance because Cosine distance only cares about the distance on the angle but ignores the scale of the vectors, while in Swimmer, the scale of the states is also important. The results show that the choice of this distance function is flexible and depends on the specific choice of the state space in our problem.

Comparison with Mapping-Based Methods Mapping-based methods translate the demonstrations across different environments by learning state mappings and action mappings [52], which can be used in our problem setting by mapping the source demonstrations to the target environment. However, our problem setting does not

ensure that there exists a mapping between the demonstrators and the imitator, which violates the assumption of the mapping-based methods. We thus do not include any mapping-based methods in our experiments in the main body of our work. However, here as an additional experiment, we compare our method with the state-of-the-art mapping-based method, DCC [52].

DCC requires random trajectories from both the demonstrators and the imitator to learn a mapping, but we do not have access to the demonstrators' environment and only have access to demonstrators' demonstrations. So we use the demonstrations and the imitator's random trajectories as the input to DCC. As shown in Table 5.4, the performance of DCC is much worse than our method and even worse than GAIL. This is because DCC itself is a good mapping-based method but mapping-based methods are not quite suitable for our problem setting. In fact, there should not exist a mapping between demonstrations and the imitator's random trajectories. Building such a mapping causes a severe mismatch between states and actions of different environments and makes the translated demonstrations distort the original demonstrations.

Table 5.5: The average expected return of demonstrations in different environments and the expected return of our method.

	Swimmer	Walker2d		HalfCheetah		Hopper	Simulated Robot	Real Robot
		First	Second	First	Second			
Demonstrations	106±3	3098±118	3720±336	3229±170	3337±67	3460±87	1823±110	2531± 362
Ours	75±20	3147±10	3424±645	2832±291	3142±89	3330±115	2127±5053	2746±2712

Comparison with the Collected Demonstrations We compare the expected return of our approach with the demonstrations in Table 5.5. We observe that in the first setting of Walker2d environment, Hopper environment, the simulated robot, and the real robot environments, our approach performs comparably to the expected return of demonstrations, which are optimal demonstrations for different demonstrators. In the Swimmer, the second setting of Walker2d and the HalfCheetah environments, the performance is worse than the demonstrations. This is because only a few demonstrations are feasible for the imitator and that may not be enough to learn an optimal policy. However, the margin between our approach and the demonstrations

is still not large. The results show that the proposed feasibility can select useful demonstrations for the imitator to imitate.

5.5 Chapter Summary

In this chapter, we aim to develop algorithms to remove the influence of infeasible demonstrations when learning from demonstrations of other agents. We first define a continuous feasibility measure by using a distance function based on the inverse dynamics model that allows for reproducing the trajectories. However, it is difficult to collect sufficient transitions to cover the whole state-action space to learn the inverse dynamics function. Therefore, we develop a feasibility MDP (f-MDP) and derive the feasibility by learning the optimal policy for the f-MDP. We show that the policy learned from the demonstrations reweighted by the proposed feasibility score outperforms other imitation learning methods in various environments and different compositions of demonstrations.

Chapter 6

Final Words

Imitation learning is a promising learning paradigm for robot learning. However current imitation learning algorithms only allow the robots to imitate behavior that is already existing in the demonstrations. This leads to the strong assumption of the demonstrations being optimal and being collected directly on the robot of interest. This severely limits the usage of imitation learning in real-world robot learning problems and also is far from human-level learning ability, where human beings are born with the ability to reenact others' behavior without any constraint. Also, preliminary works [47, 27] show that imperfect demonstrations naturally exist in the demonstration set, and also demonstrations on the robot of interest are usually insufficient for imitation learning [52]. Thus, new algorithms need to be designed to relax this strong assumption.

In general, this thesis is an important step toward the wide applicability of imitation learning. Instead of learning from in-domain and optimal demonstrations as required by standard imitation learning, this thesis addresses learning from imperfect demonstrations, where we can leverage a wider range of demonstrations. We categorize imperfect demonstrations into suboptimal demonstrations, cross-domain demonstrations, and infeasible demonstrations and develop algorithms to tackle each component of imperfect demonstrations respectively.

6.1 Limitations and Future Work

We propose several algorithms to address learning from imperfect demonstrations; however, the current algorithms still suffer from several assumptions. Furthermore, new challenges arise when imperfect demonstrations have multiple modalities. We would like to discuss the limitations and future directions of the methods discussed in this thesis.

6.1.1 Learning from Suboptimal Demonstrations

Although CAIL is a flexible framework to address the problem of imitation learning from demonstrations with varying optimality, it is still limited in a few ways: To learn a well-performing policy, we still require that the dataset consists of demonstrations that encode useful knowledge for policy learning. We also require that the demonstrations and the imitation agent have the same dynamics. In the future, we plan to learn from demonstrations with more failures and relax the assumptions of the demonstrator and imitator having the same dynamics.

For the adversarial confidence transfer method, we still require the source and target environments to be correspondent. In the future, we plan to relax this assumption by defining weaker correspondence only between confidence instead of the strict definition of correspondence on dynamics.

6.1.2 Learning from Cross-Domain Demonstrations

Though we leverage the easy-to-access weak supervision to improve correspondence learning, this type of supervision still requires domain knowledge or human experts to annotate. One potential future direction is to learn from unlabeled or unpaired data. In the future, we also plan to automatically detect the abstraction needed for weak supervision and try to reduce the size of the required annotation to ease the labor work.

6.1.3 Learning from Infeasible Demonstrations

Our methods for learning from infeasible demonstrations only address the problem of filtering out far from feasible demonstrations but do not solve the problem of learning a policy from those demonstrations that are feasible but suboptimal. There are situations where demonstrations are feasible but not optimal for the imitator, especially when the ability of the demonstrator is more restricted than the imitator. In the future, we aim to study these more general settings.

6.1.4 Future Work on Learning from Multimodal Imperfect Demonstrations

In all of the methods discussed in this thesis, the state space or the observation space for each environment only contains one modality, such as the joint state or the RGB image. However, in practice, we may have demonstrations with a multi-modal state. For example, in a driving scenario, besides the RGB modality, we can have depth and lidar modalities to provide more information for decision-making. The multiple modalities introduce new challenges for learning from imperfect demonstrations. As demonstrated in [29], within the multiple modalities, there exist modalities that provide no new useful information. Such extraneous modalities can cause the policy to overfit the training data and lead to poor generalizability of the learned policy. Thus, it is a potential future direction for imperfect demonstrations to remove the influence of these extraneous modalities.

6.2 Closing Thoughts

This thesis is only a step towards addressing learning from imperfect demonstrations, which we believe to be an important challenge for the general applicability of imitation learning or even for developing human-level intelligence robots. As we discussed in this chapter, there are still many limitations and challenges that need to be addressed for a general framework of learning from imperfect demonstrations. These challenges

still need further research and investigation from both the machine learning and the robotics community and need collaborations of researchers from different fields.

Appendix A

Proofs

A.1 Proof of Theoretical Results in Section 3.3

In this section, we provide the proofs of the theorems proposed in this paper.

A.1.1 Preliminaries

Definition 1. *(Lipschitz-smooth) Function $f(x) : \mathbb{R}^d \to \mathbb{R}$ is Lipschitz-smooth with constant L if*

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y||, \quad \forall x, y \in \mathbb{R}^d \tag{A.1}$$

Lemma 1. *If function $f(x)$ is Lipschitz-smooth with constant L, then the following inequality holds:*

$$(\nabla f(x) - \nabla f(y))^T (x - y) \leq L||x - y||^2 \tag{A.2}$$

Proof. The proof is straightforward that

$$\begin{aligned} &(\nabla f(x) - \nabla f(y))^T (x - y) \\ &\leq ||\nabla f(x) - \nabla f(y)|| \cdot ||x - y|| \\ &\leq L||x - y||^2 \end{aligned} \tag{A.3}$$

The first equation follows from the Cauchy-Schwarz inequality, and the second inequality comes from the definition of Lipschitz-smooth. □

Lemma 2. *If function $f(x)$ is Lipschitz-smooth with constant L, then the following inequality holds:*

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}||y - x||^2, \quad \forall x, y \tag{A.4}$$

Proof. Define $g(t) = f(x + t(y - x))$. If $f(x)$ is Lipschitz-smooth with constant L, then from Lemma 1, we have

$$\begin{aligned} & g'(t) - g'(0) \\ & = \left(\nabla f(x + t(y - x)) - \nabla f(x)\right)^T (y - x) \\ & = \frac{1}{t} \left(\nabla f(x + t(y - x)) - \nabla f(x)\right)^T \left((x + t(y - x)) - x\right) \\ & \leq \frac{L}{t}||t(y - x)||^2 = tL||y - x||^2 \end{aligned} \tag{A.5}$$

We then integrate this equation from $t = 0$ to $t = 1$:

$$\begin{aligned} f(y) = g(1) &= g(0) + \int_0^1 g'(t)dt \\ &\leq g(0) + \int_0^1 g'(0)dt + \int_0^1 tL||y - x||^2 dt \\ &= g(0) + g'(0) + \frac{L}{2}||y - x||^2 \\ &= f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}||y - x||^2 \end{aligned} \tag{A.6}$$

□

A.1.2 Main Proofs

Theorem 3. *(Convergence) Suppose the outer loss $\mathcal{L}_{out}$ is Lipschitz-smooth with constant L, the inequality*

$$\nabla_\theta \mathcal{L}_{out}(\theta_{\tau+1})^\top \nabla_\theta \mathcal{L}_{in}(\theta_\tau, \beta_{\tau+1}) \geq C||\nabla_\theta \mathcal{L}_{in}(\theta_\tau, \beta_{\tau+1})||^2 \tag{A.7}$$

holds for a constant $C \geq 0$ in every step τ[1], and the learning rate satisfies $\mu \leq \frac{2C}{L}$, then the outer loss decreases along with each iteration: $\mathcal{L}_{out}(\theta_{\tau+1}) \leq \mathcal{L}_{out}(\theta_\tau)$, and the equality holds if $\nabla_\beta \mathcal{L}_{out}(\theta_\tau) = 0$ or $\theta_{\tau+1} = \theta_\tau$.

Proof. Since $\mathcal{L}_{\text{out}}$ is Lipschitz-smooth, following Lemma 2, we have

$$\begin{aligned}
&\mathcal{L}_{\text{out}}(\theta_{\tau+1}) - \mathcal{L}_{\text{out}}(\theta_\tau) \\
&\leq \nabla_\theta \mathcal{L}_{\text{out}}(\theta_\tau)^T(\theta_{\tau+1} - \theta_\tau) + \frac{L}{2}||(\theta_{\tau+1} - \theta_\tau)||^2 \\
&= -\mu \nabla_\theta \mathcal{L}_{\text{out}}(\theta_{\tau+1})^T \nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1}) \\
&+ \frac{L}{2}\mu^2 ||\nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1})||^2 \\
&\leq -\left(\mu C - \frac{L}{2}\mu^2\right) ||\nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1})||^2 \\
&\leq 0
\end{aligned} \tag{A.8}$$

The first inequality comes from Lemma 2, and the second inequality holds because we update θ_τ to $\theta_{\tau+1}$ only when $\nabla_\theta \mathcal{L}_{\text{out}}(\theta_{\tau+1})^T \nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1}) \geq C||\nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1})||^2$ holds, otherwise $\theta_{\tau+1} = \theta_\tau$ so $\mathcal{L}_{\text{out}}(\theta_{\tau+1}) = \mathcal{L}_{\text{out}}(\theta_\tau)$. The third inequality holds because we choose the learning rate to satisfy $\mu \leq \frac{2C}{L}$.

Then if $\nabla_\theta \mathcal{L}_{\text{out}}(\theta_\tau) = 0$, and if Eqn. (A.7) is satisfied, we have $\nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1}) = 0$. Following the updating rule of α in Eqn. (10), we can derive $\theta_{\tau+1} = \theta_\tau$, so $\mathcal{L}_{\text{out}}(\theta_{\tau+1}) = \mathcal{L}_{\text{out}}(\theta_\tau)$. Besides, if Eqn. (11) is not satisfied, we also have $\theta_{\tau+1} = \theta_\tau$, and thus $\mathcal{L}_{\text{out}}(\theta_{\tau+1}) = \mathcal{L}_{\text{out}}(\theta_\tau)$. □

We now provide the proof of Theorem 2 on the convergence rate of the algorithm.

Theorem 4. *(Convergence Rate) Under the assumptions in Theorem 3, let*

$$g(\theta, \beta) = \theta - \mu \nabla_\theta \mathcal{L}_{in}(s, a; \theta, \beta). \tag{A.9}$$

We assume that $\mathcal{L}_{out}(g(\theta, \beta))$ is Lipschitz-smooth w.r.t. β with constant L_1, $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ have σ-bounded gradients, and the norm of $\nabla_\beta \nabla_\theta \mathcal{L}_{in}(\theta; \beta)$ is bounded by σ_1. L is

[1]We remove (s, a) in $\mathcal{L}_{\text{in}}$ for notation simplicity.

the Lipschitz-smooth constant for $\mathcal{L}_{out}$ w.r.t. $g(\theta, \beta)$ as shown in Theorem 3. Consider the total training steps as T, we set $\alpha = \frac{C_1}{\sqrt{T}}$, for some constant C_1 where $0 < C_1 \leq \frac{2}{L_1}$ and $\mu = \frac{C_2}{T}$ for some constant C_2. CAIL can achieve:

$$\min_{1\leq\tau\leq T} \mathbb{E}[||\nabla_\beta \mathcal{L}_{out}(\theta_\tau)||^2] \leq O\left(\frac{1}{\sqrt{T}}\right). \tag{A.10}$$

Proof. According to the update rule of θ, we have

$$\begin{aligned}
&\mathcal{L}_{\text{out}}(\theta_{\tau+1}) - \mathcal{L}_{\text{out}}(\theta_\tau) \\
&= \mathcal{L}_{\text{out}}(\theta_\tau - \mu\nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1})) \\
&\quad - \mathcal{L}_{\text{out}}(\theta_{\tau-1} - \mu\nabla_\theta \mathcal{L}_{\text{in}}(\theta_{\tau-1}, \beta_\tau)) \\
&= \{\mathcal{L}_{\text{out}}(\theta_\tau - \mu\nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1})) \\
&\quad - \mathcal{L}_{\text{out}}(\theta_{\tau-1} - \mu\nabla_\theta \mathcal{L}_{\text{in}}(\theta_{\tau-1}, \beta_{\tau+1}))\} \\
&\quad + \{\mathcal{L}_{\text{out}}(\theta_{\tau-1} - \mu\nabla_\theta \mathcal{L}_{\text{in}}(\theta_{\tau-1}, \beta_{\tau+1})) \\
&\quad - \mathcal{L}_{\text{out}}(\theta_{\tau-1} - \mu\nabla_\theta \mathcal{L}_{\text{in}}(\theta_{\tau-1}, \beta_\tau))\} \\
&= \{\mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_{\tau+1})) - \mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_{\tau+1}))\} \\
&\quad + \{\mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_{\tau+1})) - \mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_\tau))\}
\end{aligned} \tag{A.11}$$

We remove (s, a) in the $\mathcal{L}_{\text{in}}$ for notation convenience. For the first term,

$$\begin{aligned}
&\mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_{\tau+1})) - \mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_{\tau+1})) \\
&\leq \nabla\mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_{\tau+1}))^T \Delta g + \frac{L}{2}||\Delta g||^2
\end{aligned} \tag{A.12}$$

where

$$\begin{aligned}
\Delta g &= g(\theta_\tau, \beta_{\tau+1}) - g(\theta_{\tau-1}, \beta_{\tau+1}) \\
&= [\theta_\tau - \mu\nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1})] \\
&\qquad - [\theta_{\tau-1} - \mu\nabla_\theta \mathcal{L}_{\text{in}}(\theta_{\tau-1}, \beta_{\tau+1})] \\
&= -\mu[\nabla_\theta \mathcal{L}_{\text{in}}(\theta_\tau, \beta_{\tau+1}) + \nabla_\theta \mathcal{L}_{\text{in}}(\theta_{\tau-1}, \beta_\tau) \\
&\qquad - \nabla_\theta \mathcal{L}_{\text{in}}(\theta_{\tau-1}, \beta_{\tau+1})]
\end{aligned} \tag{A.13}$$

Since $\mathcal{L}_{\text{in}}$ has σ-bounded gradients, we take the norm on both sides and use the triangle

inequality, so

$$||\Delta g|| \leq 3\mu\sigma \tag{A.14}$$

Substitute this into Eqn. (A.12), we have

$$\begin{aligned} &\mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_{\tau+1})) - \mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_{\tau+1})) \\ &\leq 3\mu\sigma^2 + \frac{9}{2}L\mu^2\sigma^2 \end{aligned} \tag{A.15}$$

And for the second term,

$$\begin{aligned} &\mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_{\tau+1})) - \mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_\tau)) \\ &\leq \nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_\tau))^T(\beta_{\tau+1} - \beta_\tau) + \frac{L_1}{2}||\beta_{\tau+1} - \beta_\tau||^2 \\ &= -\alpha\nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_\tau))^T\nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_\tau)) \\ &+ \frac{L_1\alpha^2}{2}||\nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_\tau))||^2 \\ &= -(\alpha - \frac{L_1\alpha^2}{2})||\nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_\tau))||^2 \\ &+ \alpha(\nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_\tau)) - \nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_\tau)))^T\nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_\tau)) \end{aligned} \tag{A.16}$$

Since $\nabla_\beta\nabla_\theta \mathcal{L}_{\text{in}}(\theta, \beta)$ is bounded by σ_1 and $\mathcal{L}$ has σ-bounded gradients, then

$$\begin{aligned} &\nabla_\beta \mathcal{L}_{\text{out}}(g(\theta, \beta)) \\ &= \nabla_\beta g(\theta, \beta)^T\nabla_g \mathcal{L}_{\text{out}}(g(\theta, \beta)) \\ &= -\mu\nabla_\beta\nabla_\theta \mathcal{L}_{\text{in}}(\theta, \beta)^T\nabla_g \mathcal{L}_{\text{out}}(g(\theta, \beta)) \\ &\leq \mu\sigma\sigma_1 \end{aligned} \tag{A.17}$$

So

$$\begin{aligned} &\mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_{\tau+1})) - \mathcal{L}_{\text{out}}(g(\theta_{\tau-1}, \beta_\tau)) \\ &\leq -(\alpha - \frac{L_1\alpha^2}{2})||\nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_\tau))||^2 \\ &\quad + 2\alpha\mu\sigma\sigma_1 \end{aligned} \tag{A.18}$$

Combining the two parts, we can derive that

$$\begin{aligned}
&\mathcal{L}_{\text{out}}(\theta_{\tau+1}) - \mathcal{L}_{\text{out}}(\theta_\tau) \\
&\leq -(\alpha - \frac{L_1\alpha^2}{2})||\nabla_\beta \mathcal{L}_{\text{out}}(g(\theta_\tau, \beta_\tau))||^2 \\
&\quad + 3\mu\sigma^2 + \frac{9}{2}L\mu^2\sigma^2 + 2\alpha\mu\sigma\sigma_1
\end{aligned} \tag{A.19}$$

Summing up both sides from $\tau = 1$ to T, and rearranging the terms, we can derive that

$$\begin{aligned}
&\sum_{\tau=1}^{T}(\alpha - \frac{L_1\alpha^2}{2})||\nabla_\beta \mathcal{L}_{\text{out}}(\theta_\tau)||^2 \\
&\leq \mathcal{L}_{\text{out}}(\theta_1) - \mathcal{L}_{\text{out}}(\theta_{T+1}) + T\left(3\mu\sigma^2 + \frac{9}{2}L\mu^2\sigma^2 + 2\alpha\mu\sigma\sigma_1\right)
\end{aligned} \tag{A.20}$$

Since $\alpha - \frac{L_1\alpha^2}{2} \geq 0$, we have

$$\begin{aligned}
&\min_\tau \mathbb{E}[||\nabla_\beta \mathcal{L}_{\text{out}}(\theta_t)||^2] \\
&\leq \frac{\sum_{\tau=1}^{T}(\alpha - \frac{L_1\alpha^2}{2})||\nabla_\beta \mathcal{L}_{\text{out}}(\theta_t)||^2}{T(\alpha - \frac{L_1\alpha^2}{2})} \\
&\leq \frac{1}{T\alpha(1 - \frac{L_1\alpha}{2})}\left[\mathcal{L}_{\text{out}}(\theta_1) - \mathcal{L}_{\text{out}}(\theta_{T+1})\right. \\
&\quad \left. + T\left(3\mu\sigma^2 + \frac{9}{2}L\mu^2\sigma^2 + 2\alpha\mu\sigma\sigma_1\right)\right] \\
&\leq \frac{1}{\alpha\sqrt{T}(\sqrt{T}-1)}\left[\mathcal{L}_{\text{out}}(\theta_1) - \mathcal{L}_{\text{out}}(\theta_{T+1})\right. \\
&\quad \left. + T\left(3\mu\sigma^2 + \frac{9}{2}L\mu^2\sigma^2 + 2\alpha\mu\sigma\sigma_1\right)\right] \\
&= \frac{\mathcal{L}_{\text{out}}(\theta_1) - \mathcal{L}_{\text{out}}(\theta_{T+1})}{\alpha\sqrt{T}(\sqrt{T}-1)} + \frac{\sigma\mu\sqrt{T}}{\alpha(\sqrt{T}-1)}\left(3\sigma + \frac{9}{2}L\mu\sigma + 2\alpha\sigma_1\right) \\
&= \frac{\mathcal{L}_{\text{out}}(\theta_1) - \mathcal{L}_{\text{out}}(\theta_{T+1})}{C_1(\sqrt{T}-1)} + \frac{\sigma C_2}{C_1(\sqrt{T}-1)}\left(3\sigma + \frac{9}{2}L\mu\sigma + 2\alpha\sigma_1\right) \\
&= O\left(\frac{1}{\sqrt{T}}\right)
\end{aligned} \tag{A.21}$$

The second inequality holds since $1 - \frac{L_1\alpha}{2} \geq 1 - \frac{1}{\sqrt{T}}$. □

Theorem 5. *RK in Eqn. (14) in the main text is Lipschitz.*

Proof. We consider the case where $\eta_{\xi_i} > \eta_{\xi_j}$. The case for $\eta_{\xi_i} <= \eta_{\xi_j}$ can be demonstrated similarly. We prove that RK is Lipschitz by definition. Let $z = \xi_i' - \xi_j'$, the derivative of RK is

$$\nabla RK_z = \begin{cases} 0 & z > \epsilon, \\ \frac{1}{2\epsilon}(z - \epsilon) & -\epsilon \leq z \leq \epsilon, \\ -1 & z < -\epsilon \end{cases}$$

Thus, the second-order derivative of RK satisfies that $|\nabla(\nabla RK_z)_z| \leq \frac{1}{2\epsilon}$. If we take $L > \frac{1}{2\epsilon}$, then $\forall z_1, z_2, |\nabla(z_1) - \nabla(z_2)| < L|z_1 - z_2|$. This proves that RK is Lipschitz smooth.

□

Appendix B

Algorithm

B.1 Algorithm for Section 3.2

The detailed description of the algorithm is shown in Algorithm 1.

Algorithm 1: Algorithm

Input: Demonstrations Ξ
Compute the optimality $\forall \xi$ in Ξ using Eqn. (3.1)
Derive the probability distribution p_w as $p_{w_i} = \frac{w_i}{\sum_i w_i}$
while *not converging* **do**
  Train π with a state-based imitation learning algorithm with state transitions sampled from p_w;
end
Output: Learned optimal policy π^*.

B.2 Algorithm for Section 3.4

We go through the steps of the algorithm of learning from imperfect demonstrations via adversarial confidence transfer in Algorithm 2. Lines 2-6 show the first stage of our framework as in Fig. 3.7 (left): training E^{src} with source confidence-labeled data. Lines 7-14 show the process of the second stage in Fig. 2 (left): aligning the distribution of source and target state-action pairs in the common latent space. Lines 8-11 and Lines

12-14 show we iteratively update the target encoder E and the discriminators D_k and D'_k.

After we learn the confidence prediction function $F(E)$, we predict the confidence for each state-action pair in the target demonstrations and conduct standard imitation learning on the re-weighted target demonstrations similar to the approaches in Sec. 3.2 and 3.3.

Algorithm 2: Algorithm for learning the target confidence predictor

Input: The demonstration set of the source environment Ξ^{src} and of the target environment Ξ. The confidence function c^{src} for the source environment.

Initialize E^{src}, E, F, $D_i - D_K$ and $D'_i - D'_k$;

while *not converging* **do**

  Sample a batch of state-action pairs $\{(s^{\text{src}}, a^{\text{src}})\}$ from Ξ^{src};

  Compute the confidence for state-action pairs in $\{(s^{\text{src}}, a^{\text{src}})\}$ with c^{src};

  Train E^{src} and F with $\{((s^{\text{src}}, a^{\text{src}}), c^{\text{src}}(s^{\text{src}}, a^{\text{src}}))\}$ according to the loss in Eqn. (1) and the optimization objective in Eqn. (4);

end

Fix the parameters of E^{src} and F. **while** *not converging* **do**

  for $k = 1 \rightarrow K$ **do**

    Sample a batch of partial trajectories $\{(s_1, a_1, \ldots, s_k, a_k)\}$ with length k;

  end

  Train E with partial trajectories of all lengths: $\{(s_1, a_1, \ldots, s_k, a_k)\}|_{k=1,\cdots,K}$ according to the loss in Eqn. (2) and (3), and the optimization objective in Eqn. (6);

  for $k = 1 \rightarrow K$ **do**

    Train D_k and D'_k with the partial trajectories $\{(s_1, a_1, \ldots, s_k, a_k)\}$ according to the loss in Eqn. (2) and (3), and the optimization objective in Eqn. (5);

  end

end

Output: The target confidence predictor $F \circ E$.

B.3 Algorithm for Section 5.3

We go through the steps of the algorithm of learning feasibility to imitate demonstrators with different dynamics in Algorithm 3. For the state-based imitation learning algorithm used to learn the final policy from reweighted demonstrations, we use the state-based GAIL as [10]. Lines 1-6 show the process of constructing N f-MDPs (one for each demonstrator) and training the optimal policy for each f-MDP. Line 7 shows how to compute feasibility with the optimal policy for each f-MDP. Lines 8-11 show imitation learning over the newly reweighted demonstrations.

Algorithm 3: Algorithm

Input: Demonstrations Ξ^j from each demonstrator $\mathcal{M}_j^d$, $(1 \leq j \leq N)$.
for *j=1 to N* **do**
  Construct the trajectory f-MDP M_j^f based on the demonstration set Ξ_j according to Eqn. (2);
  Train an optimal policy π_j^* for the trajectory f-MDP M_j^f;
  Compute the feasibility $w(\xi_j)$ for each trajectory $\xi_j \in \Xi_j$ as in Eqn. (3);
  Assign the feasiblity $w(\xi_j)$ to each state transitions in ξ_j;
end
Compute $p_w((s_t^d, s_{t+1}^d)) \leftarrow \frac{w((s_t^d, s_{t+1}^d))}{\sum_{\left(s_{t'}^d, s_{t'+1}^d\right) \in T} w((s_{t'}^d, s_{t'+1}^d))}$
while *not converging* **do**
  Sample a batch of state transitions from p_w;
  Train π with the sampled batch of state transitions by a state-based imitation learning algorithm: state-based GAIL

$$\min_{\theta_\pi} \max_{\theta_d} \mathbb{E}_{(s_t, s_{t+1}) \sim \pi}(\log(D(s_t, s_{t+1}))) + \mathbb{E}_{(s_t, s_{t+1}) \sim p_w}(1 - \log(D(s_t, s_{t+1}))), \tag{B.1}$$

  where θ_π is the parameter of π and θ_d is the parameter for the discriminator D in state-based GAIL;
end
Output: Learned optimal policy π^* for $\mathcal{M}^i$.

Currently, we consider the state transitions in each trajectory to share the same feasibility but do not consider the case where parts of the trajectories are more feasible than some other parts. This is because the state-based GAIL in our algorithm as well as many standard imitation learning algorithms rely on learning from the full

trajectory from the start to the end state. If a segment of the trajectory is far from feasible or harmful, then the remaining part is also not going to be useful for our algorithm. Therefore, we only learn from trajectories that are helpful in all parts.

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.

[2] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, 1995.

[3] Aayush Bansal, Shugao Ma, Deva Ramanan, and Yaser Sheikh. Recycle-gan: Unsupervised video retargeting. In *Proceedings of the European conference on computer vision (ECCV)*, pages 119–135, 2018.

[4] Stan Birchfield and Carlo Tomasi. Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, 35(3):269–293, 1999.

[5] Erdem Bıyık, Dylan P Losey, Malayandi Palan, Nicholas C Landolfi, Gleb Shevchuk, and Dorsa Sadigh. Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences. *arXiv preprint arXiv:2006.14091*, 2020.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[7] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, pages 783–792. PMLR, 2019.

[8] Daniel S Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Conference on Robot Learning*, pages 330–359. PMLR, 2020.

[9] Zhangjie Cao, Erdem Bıyık, Woodrow Z Wang, Allan Raventos, Adrien Gaidon, Guy Rosman, and Dorsa Sadigh. Reinforcement learning based control of imitative policies for near-accident driving. *arXiv:2007.00178*, 2020.

[10] Zhangjie Cao and Dorsa Sadigh. Learning from imperfect demonstrations from agents with varying dynamics. *IEEE Robotics and Automation Letters (RA-L)*, 2021.

[11] Letian Chen, Rohan Paleja, and Matthew Gombolay. Learning from suboptimal demonstration via self-supervised reward regression. In *Conference on Robot Learning*. PMLR, 2020.

[12] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[13] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

[14] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *ICLR*, 2018.

[15] 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.

[16] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[17] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

[18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[19] 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.

[20] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NeurIPS*, volume 29, 2016.

[21] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[22] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning*, pages 1989–1998. PMLR, 2018.

[23] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[24] Ananya Harsh Jha, Saket Anand, Maneesh Singh, and VSR Veeravasarapu. Disentangling factors of variation with cycle-consistent variational auto-encoders. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 805–820, 2018.

[25] Jonathan Lee, Ching-An Cheng, Ken Goldberg, and Byron Boots. Continuous online learning and new insights to online imitation learning. *arXiv preprint arXiv:1912.01261*, 2019.

[26] Fangchen Liu, Zhan Ling, Tongzhou Mu, and Hao Su. State alignment-based imitation learning. In *ICLR*, 2019.

[27] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

[28] Dylan P Losey, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, and Dorsa Sadigh. Controlling assistive robots with learned latent actions. In *ICRA*, 2020.

[29] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

[30] Andrew N Meltzoff. Understanding the intentions of others: re-enactment of intended acts by 18-month-old children. *Developmental psychology*, 31(5):838, 1995.

[31] Chrystopher L Nehaniv, Kerstin Dautenhahn, et al. The correspondence problem. *Imitation in animals and artifacts*, 41, 2002.

[32] Mark Nielsen. 12-month-olds produce others' intended but unfulfilled acts. *Infancy*, 14(3):377–389, 2009.

[33] Ellen R. Novoseller, Yibng Wei, Yanan Sui, Yisong Yue, and J. Burdick. Dueling posterior sampling for preference-based reinforcement learning. In *Uncertainty in Artificial Intelligence (UAI)*, 2020.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga,

et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[35] Ahmed H. Qureshi, Byron Boots, and Michael C. Yip. Adversarial imitation via variational inverse reinforcement learning. In *ICLR*, 2019.

[36] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[37] Stefan Schaal. Learning from demonstration. *Advances in neural information processing systems*, 9, 1996.

[38] 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.

[39] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[41] 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.

[42] Samarth Shukla, Luc Van Gool, and Radu Timofte. Extremely weak supervised image-to-image translation for semantic segmentation. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3368–3377. IEEE, 2019.

[43] Herbert Alexander Simon. *Models of bounded rationality: Empirically grounded economic reason*. MIT press, 1997.

[44] Laura Smith, Nikita Dhawan, Marvin Zhang, Pieter Abbeel, and Sergey Levine. Avid: Learning multi-stage tasks via pixel-level translation of human videos. *arXiv preprint arXiv:1912.04443*, 2019.

[45] 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.

[46] Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018.

[47] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning*, pages 6818–6827, 2019.

[48] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *ICML*, 2019.

[49] Jing Zhang, Zewei Ding, Wanqing Li, and Philip Ogunbona. Importance weighted adversarial nets for partial domain adaptation. In *CVPR*, 2018.

[50] Marc Yanlong Zhang, Zhiwu Huang, Danda Pani Paudel, Janine Thoma, and Luc Van Gool. Weakly paired multi-domain image translation. *Proceedings BMVC 2020*, 2020.

[51] Qiang Zhang, Tete Xiao, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Learning cross-domain correspondence for control with dynamics cycle-consistency. *arXiv preprint arXiv:2012.09811*, 2020.

[52] Qiang Zhang, Tete Xiao, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Learning cross-domain correspondence for control with dynamics cycle-consistency. In *International Conference on Learning Representations*, 2021.

[53] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[54] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.