Addressing Distribution Shift in Robotic Imitation Learning

by

Trevor Louis Ablett

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Institute for Aerospace Studies
University of Toronto

Abstract

Addressing Distribution Shift in Robotic Imitation Learning

Trevor Louis Ablett
Doctor of Philosophy
Graduate Department of Institute for Aerospace Studies
University of Toronto
2025

The widespread use of robotic systems remains out of reach for a variety of reasons. Even assuming perfect sensing and modelling, determining the optimal actions to perform to complete tasks is challenging, particularly as task complexity rises. An appealing alternative to explicit modelling is imitation learning (IL), where control policies are learned directly from expert data containing raw observations and actions. Unfortunately, IL is subject to *distribution shift*: when a robot encounters data not adequately covered by its training dataset, it can fail, sometimes catastrophically. This thesis presents approaches to resolve distribution shift in robotic IL. The first two methods presented directly modify the training dataset as it is collected: one allows better coverage of the full distribution encountered in a mobile manipulation setting, and the next modifies individual trajectories to address an inevitable shift resulting from the use of kinesthetic teaching, a common approach to generating demonstrations. Our third approach directly detects distribution shift and allows the expert to intervene and provide corrective data in response. Finally, we investigate inverse reinforcement learning (IRL), which ostensibly resolves distribution shift, but in practice can suffer from deceptive learned rewards causing locally maximal but globally poor behaviour. In our fourth approach, we learn policies from demonstrations of auxiliary tasks, in addition to the main task, to break out of these local maxima in IRL. Our fifth approach extends this exploration improvement to the more difficult case of merely having *examples* of completed tasks and auxiliary tasks, improving performance while simultaneously significantly reducing the collection burden on the expert. These approaches are demonstrated on both simulated and real robotic manipulators, and we provide open source code to reproduce and build upon our results.

Epigraph

We all admire great accomplishments in the sciences, arts, and humanities — but we rarely acknowledge how much we achieve in the course of our everyday lives.

Marvin Minsky, *The Emotion Machine*

A failure is not always a mistake; it may simply be the best one can do under the circumstances. The real mistake is to stop trying.

B. F. Skinner, *Beyond Freedom and Dignity*

There will be mountains you won't move.

Frank Ocean, *Godspeed*

To Christina, whose encouragement, grace, and above all patience, made this possible.

Acknowledgements

Completing this thesis has been, as I am sure most other graduate students can relate to, extraordinarily difficult. I am exceptionally lucky to have family, friends, and colleagues who have supported me throughout the best and worst of the experience, and who provided me the mental fortitude to push through. What follows is a non-exhaustive list of people who have helped shape me into the researcher and human being that I am today.

First and foremost, I must thank my advisor, Jonathan Kelly. Your support and patience has been immeasurable over these years, and your willingness to allow me to pursue just about anything I could come up with gave me the strength to follow my own judgment, as well as the maturity to overcome limitations and failures. Next, my committee members, Gabriele D'Eleuterio and Angela Schoellig. You have both provided continued patience and guidance as my research took many unexpected twists and turns, and I cannot have asked for a better committee. I also must thank Florian Shkurti, who generously met with me multiple times early in my studies, providing many fruitful discussions and research directions that informed much of this work in its early days. Finally, I'd like to thank Douglas Down, whose superb teaching and kind demeanor helped convince me to pursue further studies, and who was also kind enough to provide innumerable reference letters.

Of course, I must thank my colleagues at the STARS lab (and adjacent robotics labs), whose willingness to engage in thoughtful and provocative discussion, both in and out of the lab, has always made this the group a particularly welcoming bunch. In particular, I'd like to thank Oliver, Filip, Brandon, Jordan, Matt, Emmett, Olivier, Justin, Adam Hall, Adam Heins, Siqi, and Melissa: starting with you all, having countless deep and hilarious discussions, and watching you all grow into brilliant and accomplished researchers has been inspiring. I'd also like to thank Valentin, Lee, Karime, Adam S., and Chris M. for your guidance, advice, and friendship as the resident senior students. Mia, I'd like to thank you for making co-supervision easy as you pursue your Master's, and also for being a great friend and sounding board for all of our common struggles throughout this journey. I must also mention Chris G., Erin, Andrej, Abhinav, Philippe, Katie, Selina, Miguel, and Karyna: all of you have been a pleasure to work with, talk with, and watch grow. Finally, I must mention Carmela and Joan, whose friendship and guidance at UTIAS over the years has always made coming into the facility, despite its inconvenient location, a treat.

I've been lucky enough to have many coauthors over the years who helped make research projects far more fun and satisfying. In particular, Bryan, your technical chops and perseverance made you an absolute delight to work with, and I'm so excited to see where your current studies take you in the future. I must also thank Daniel, Jayce, Cathy, Luke, Jayanti, and Yuchen: your aptitude as undergraduate researchers made completing research with you an easy experience, and I'm also very excited to see where your future studies take you.

I was also lucky enough to spend a year as a researcher at the Samsung AI Center in Montreal, where I met and interacted with phenomenal researchers. I have particular gratitude towards Francois Hogan, Gregory Dudek, and Kaleem Sidiqi, whose guidance was kind and valuable and made my integration into the center smooth and simple. I'd also like to thank Dmitriy, Adam S., Affan, Abhisek, Rui

Heng, Jean-Francois, Wei-Di, Charlotte, Bobak, and Laurena for their friendship and help throughout.

Early in my studies, I also spent two months at the LAMOR lab at FER in Zagreb, Croatia, where I was welcomed with open arms and learned much about robotics and the Croatian way of life. I'd like to thank Filip (again) for encouraging me to pursue the opportunity, as well as Ivan Petrović, Ivan Marković, and Ivan Hrvoić for making me feel at home, with particular thanks to Ivan P. for providing multiple reference letters. In the lab, I must mention Juraj, Tomislav, Antea, and Luka for their help and friendship.

My friends outside of the world of academia have been particularly important for maintaining my sanity, and reminding me that there is, in fact, more to life than a publishing record. Adam, Sara F., Sara G., Sasha, Danielle, Matt, Jay, Kyle, Jen, Victoria, Kendall, Will H., Dawn, Michael, Liz, Karina, Tom, Zoe, Louise, Ahan, Ryan, Tristan, Shauna, Julia, Anna, Nell, Will J., Paige, Josh, Aidan, and Nick (and I probably forgot more than a few), whether it be soccer (Cuties and Bobcats, unite!), karaoke, group vacations, dungeons and dragons, or some other goofy activity, your friendship has meant the world to me. Heidi C., your insightful and patient guidance was particularly instrumental during some of the darkest moments of this process, and I'll always be grateful for it.

Of course, I wouldn't be anywhere that I am now without the early and ongoing guidance and support of my loving family. My mother and father introduced me to math, science, and engineering, and their cultivation of my interest in it is likely the main reason I'm on the path I am now. My sister Devon has always been an inspiration and source of constant encouragement. Pam, Nonna, Nonno, Matthew and Kyla, you have all consistently shown interest, curiosity, and excitement about my studies, and I couldn't have asked for more.

Above all, I must devote the majority of my thanks to my partner, Christina. Before and throughout this process, I have been consistently humbled by your wisdom, kindness, and support. Our family has doubled in size since we (and it certainly is we, not I) began this journey, and you, Bowie, and Wesley keep me grounded in reality and, again, remind me that there is much, much more to life than academic pursuits. Through the best and worst of this process, your confidence in me has never wavered, and I simply cannot believe my good fortune in having found you and having the privilege to grow and learn with you.

Contents

Notation

General

x	:	A real scalar.
$\mathbf{x}$	:	A real column vector.
$\mathbf{X}$	:	A real matrix.
$\mathcal{D}$	:	A dataset of input-output pairs for supervised learning.

Geometry

$\underrightarrow{\mathcal{F}}_a$	:	A reference frame in three dimensions.

Probability

$\mathcal{N}(\boldsymbol{\mu}, \mathbf{R})$	:	Normally distributed with mean $\boldsymbol{\mu}$ and covariance matrix $\mathbf{R}$.
$U([l, u])$	:	Uniformly distributed between lower limit l and upper limit u.
$\mathbb{E}[\cdot]$	:	The expectation operator.
X	:	Random variable corresponding to $\mathbf{x}$.
$p(x)$	:	Probability that random variable X takes the value x. For continuous distributions, density of distribution for X at $X = x$.
$p(y \mid x)$	:	Probability that random variable Y takes the value y given that random variable X takes the value x. For continuous distributions, density values for $Y = y$ and $X = x$.

Markov Decision Processes

$\mathcal{M}$	:	A Markov decision process (MDP).
$\mathcal{S}$	:	A state space.
$U(\mathcal{A})$	:	A discrete uniform distribution over the action space.

$\mathcal{A}$	:	An action space.
R	:	A reward function.
$\mathcal{P}$	:	A transition distribution.
ρ_0	:	An initial state distribution.
γ	:	A discount factor.
s	:	A system state.
o	:	An observation in a partially observable MDP.
a	:	An action.
r	:	A reward.
t	:	A discrete timestep.
T	:	The final timestep in a finite-horizon MDP.
$(\cdot)_t$	:	A variable with an associated time t.
π	:	A policy distribution or function.
τ	:	A trajectory of successive (s, a) pairs.
s'	:	The state after s in a trajectory.
a'	:	The action after a in trajectory.
$V(s)$	:	A value function for state s.
$Q(s, a)$	:	An action-value Q-function for state s and action a.
$\mathcal{B}$	:	A buffer of many (s, a, r, s') or (s, a, s') tuples.
$\mathcal{T}$	:	A predefined auxiliary task in an an MDP.

Imitation Learning

$\mathcal{D}_E$	:	A dataset of expert state-action or observation-action pairs (used for supervised learning in an MDP).
$\mathcal{D}_\pi$	:	A dataset of policy state-action or observation-action pairs.
D	:	A discriminator function for classifying expert-generated and policy-generated data.
$\mathcal{B}^E$	:	A buffer of expert (s, a, r, s') or (s, a, s') tuples (used for expert data in reinforcement learning or inverse reinforcement learning).

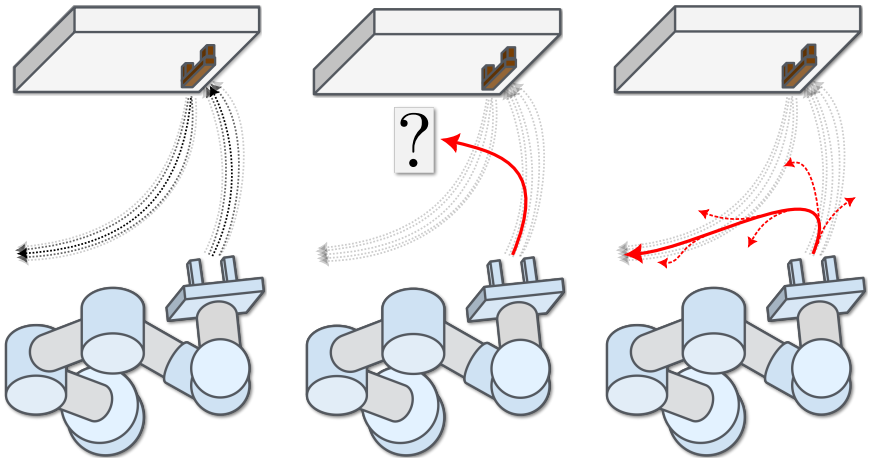
Chapter 1

Introduction

> We are all prodigious olympians in perceptual and motor areas, so good that we make the difficult look easy.
>
> Hans Moravec, *Mind Children*

Autonomous robots are still not able to complete many tasks that we, as humans, find easy. Robots excel at tasks where no environment feedback is required (e.g., point-to-point welding tasks). If feedback is required, they can excel with accurate state estimation, a correct dynamics model, and an effective strategy for choosing optimal actions. In most tasks we care about, our robots must be highly responsive to sensor feedback, state estimation has errors, the world model is inaccurate if available at all, and the task complexity makes it very difficult to know what an optimal action should be. Better, more human-like hardware and sensing is certainly an important part of improving the capability of our robots, but it is obvious that our current platforms greatly underperform their mechanical capability. Consider that a self-driving car has identical mechanical capability, and generally far greater sensing capability, than a human, and yet, for complicated driving scenarios, humans still outperform self-driving cars. Similarly, humans are capable of teleoperating our existing robotic manipulators to complete tasks far beyond the capability of our state-of-the-art algorithms. This human-algorithm performance gap shows that our machines are capable of completing far more complex tasks than they currently do, but continue to be limited by our own inability to program our decision-making knowledge into them.

This human-algorithm gap has helped motivate both *reinforcement learning (RL)* and *imitation learning (IL)* as data-driven alternatives to hand-crafted modelling of solutions to sequential decision making. In both RL and IL, the goal is to learn a *control policy* (or, simply, policy) that directly maps environment states (or observations) as input to actions as output. In RL, a practitioner designs a reward function and an agent, including a policy, autonomously interacts with an environment. The agent is progressively updated to maximize cumulative reward. In IL, a practitioner instead provides a set of expert demonstrations of a task being completed. A policy is then trained directly on the raw expert observations and actions via supervised learning, or the expert demonstrations are used as part of RL,



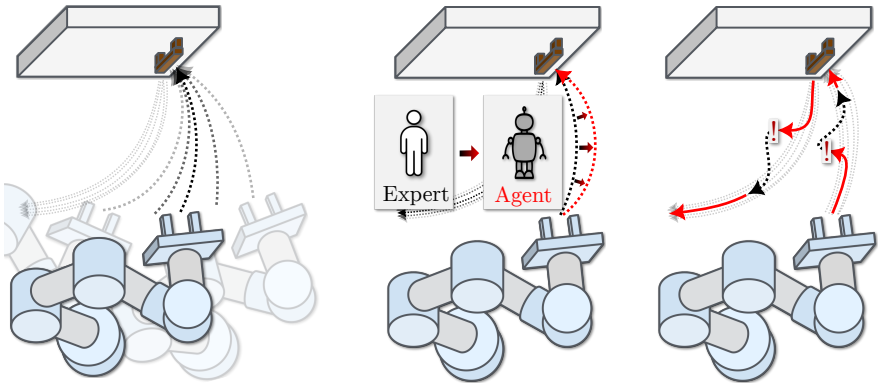
(a) Demonstrations. **(b)** Distribution shift. **(c)** Suboptimal policy.

Figure 1.1: A manipulator robot trained with IL is subject to distribution shift. Cascading errors can lead the policy to encounter states that are further and further from the distribution of expert demonstrations, ultimately leading to failure. Inverse reinforcement learning can potentially resolve the problem, but can also result in a suboptimal policy that partially matches the distribution but does not solve the task. Diagram inspired by Ross (2013).

either to model a reward (in a process known as inverse RL), or as initial data for the agent to bootstrap from. RL and IL not only allow us to circumvent the design of control policies, but being data-driven techniques, they also allow us to relax the requirements of accurate state estimation and dynamics modelling, both of which can be quite difficult. Compared with RL, IL is particularly promising for robotics, for two major reasons: (i) reward design continues to be difficult, and potentially requires task knowledge that makes designing effective policies by hand difficult in the first place, and (ii) RL can be notoriously inefficient.

Unfortunately, IL can be susceptible to distribution shift (sometimes referred to as dataset shift) (Quionero-Candela et al., 2009), where a trained model encounters data that is different from the training distribution (sometimes referred to as out-of-distribution or OOD data). This is problematic in all cases, but for robots, which continuously affect their environment to generate their next observation, it can be particularly catastrophic (Bagnell, 2005; Ross et al., 2011). A single datapoint that is only slightly out-of-distribution can result in a mild error in the policy output, generating data that is even more out-of-distribution, ultimately resulting in cascading errors leading to failure. This was perhaps first identified in the application of IL to robotics by Pomerleau (1989), where expert data for a self-driving car was only collected with the car driving safely forward. If the car started to veer left or right, generating OOD data, the policy would fail.

The distribution shift represented by the simple example from Pomerleau (1989) continues to have dramatic consequences wherever a supervised approach to IL is applied. Inverse RL ostensibly resolves this distribution shift, but in practice, an agent that encounters novel data during the RL process can learn deceptive rewards (Ecoffet et al., 2021) that stagnate learning. This thesis presents a range of methods for mitigating distribution shift in IL for robotics, sequenced with progressively lower data



(a) Multiview data. **(b)** Force-matched data. **(c)** Interventions.

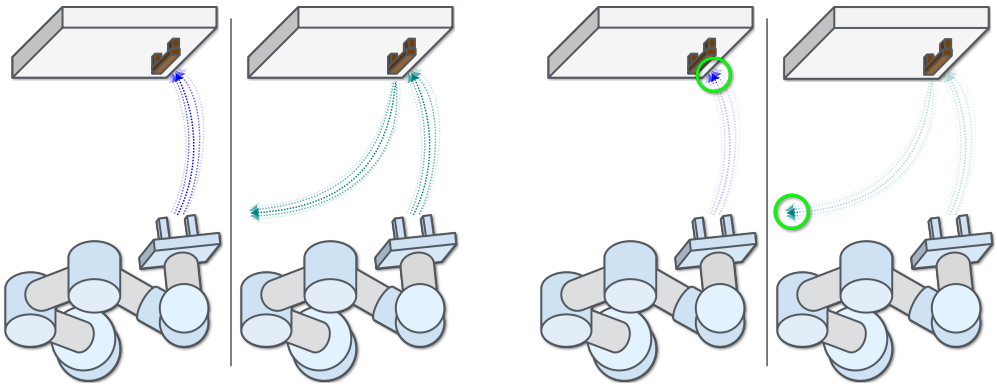
Figure 1.2: Visual summaries of approaches to resolving distribution shift investigated in Chapters 3 to 5, including multiview demonstrations (Chapter 3), force-matched demonstrations (Chapter 4), and failure prediction with interventions (Chapter 5).

collection costs to the expert from Chapters 3 to 7. In Chapters 3 and 4, we directly modify the training distribution given an expected shift. In Chapter 5, we detect out-of-distribution data during policy execution and allow the expert to provide additional, corrective data. Finally, in Chapter 6, we allow an agent to learn how to return to the expert distribution, autonomously, while avoiding suboptimal but locally maximal policies, and in Chapter 7 expand the method to use only examples of completed tasks rather than full expert trajectories.

1.1 Structure and Contributions

We begin in Chapter 2 by building the knowledge and intuition of supervised learning (Section 2.1), sequential decision making (Section 2.2) and reinforcement learning (Section 2.3). We then describe the general approaches to using expert data in imitation learning: via supervised learning with behavioural cloning (Section 2.4.1), and via self-improving execution with inverse reinforcement learning (Section 2.4.4). We also provide details on relationship of both of these methods to distribution shift in Sections 2.4.2 and 2.4.5, respectively. The remainder of this section provides a summary of the original contributions presented in Chapters 3 to 7.

1. **Multiview Manipulation from Demonstrations**
   Chapter 3 presents a method for mitigating distribution shift by considering expected changes in the distribution that may occur when a policy is executed, and generating a more diverse training distribution that leverages mutual information between viewpoints to compensate. Deliberately widening the training distribution is a general technique for addressing distribution shift, and we apply it to mobile manipulation. Specifically, leveraging the knowledge that a mobile manipulator will never attempt the same task from the exact same angle, we modify the initial state distribution of manipulation tasks to collect demonstrations from multiple viewpoints. We show that a multiview dataset, with the same *quantity* of expert data as a baseline



(a) Full-trajectory auxiliary task data. **(b)** Example-based auxiliary task data.

Figure 1.3: Visual summaries of expert data types investigated in Chapters 6 and 7 to address learned deceptive rewards in inverse reinforcement learning. We use full-trajectory demonstrations of auxiliary tasks in addition to the main task in Chapter 6, and example-based auxiliary task data in Chapter 7.

fixed-base dataset, generates far more robust policies that can even generalize to some degree of out-of-distribution data.

2. **Force-Matched Demonstrations**
   In Chapter 4, we modify the training distribution to address distribution shift, but in this case, the distribution shift is caused by an *embodiment change* between the expert and the agent (Osa et al., 2018; Billard et al., 2016). Demonstrations are modified to accommodate this change, and we experimentally validate that the modification generates far more robust policies. Specifically, the robot-environment contact force is not typically considered during kinesthetic teaching, which we resolve with a novel procedure for matching forces read with an inexpensive visuotactile sensor. We furthermore show that the data provided by multimodal visuotactile sensing is necessary for high-performing policies in contact-rich tasks.

3. **Failure Identification for Interventions**
   Chapter 5 bridges the gap between offline, supervised approaches to IL in Chapters 3 and 4 and online, reinforcement learning-based approaches in Chapters 6 and 7. Specifically, under the assumption that all IL methods based on supervised learning are likely to encounter OOD data, we present a general method for *interactive* IL (Celemin et al., 2022). We identify OOD data during policy execution and subsequently allow an expert to provide a *corrective* demonstration, appending this new data to the expert dataset and retraining the policy on the updated data. In multiple manipulation tasks, including multiview ones based on those presented in Chapter 3, we show that interactive data improves sample efficiency compared to simply collecting more data, as is done by the method from Chapter 3.

4. **Learning from Guided Play**
   In Chapter 6, we move on to an approach based on inverse reinforcement learning (IRL), where an agent autonomously attempts to learn how to remain within the expert distribution. We

find that, in practice, IRL can fail dramatically if an agent only partially matches the expert distribution, without actually completing the task. We add in simple and reusable auxiliary tasks to encourage the agent to more deeply explore its environment, and ultimately achieve the goal of matching the expert distribution without requiring any additional expert intervention.

5. **Auxiliary Control from Examples**
   Finally, in Chapter 7, we remove the need for having full expert trajectories, which can be sub-optimal, biased, or difficult to acquire. Instead, the agent is provided only *examples of success*, fully alleviating the potential for the distribution shift explored in Chapters 3 to 6. This sparse form of feedback presents a particularly challenging exploration problem, but we show that the technique from Chapter 6 is effective even with only examples of completed auxiliary tasks. This application can result in unstable learning due to highly overestimated values, and we resolve the problem with a novel scheme for value penalization.

1.2 Applications Beyond Manipulation

As we allude to in Figs. 1.1 to 1.3, the experiments in this thesis are primarily focused on robotic manipulation. However, it is worth pointing out that the *methods* presented in this work are not strictly limited to manipulation. The approaches presented in Chapters 5 to 7 can be directly applied to self-driving cars, drones, wheeled robot locomotion, humanoid or legged robot locomotion, and potentially other sequential decision-making or control domains. Additionally, the core concept of Chapter 3—in which the problem setup is exploited to deliberately widen the training distribution through more overlapping data—is not limited to manipulation either.

1.3 Associated Publications

This dissertation is comprised of work from the following first-authored papers:

1. Ablett, T., Zhai, Y., and Kelly, J. (2021b). Seeing All the Angles: Learning Multiview Manipulation Policies for Contact-Rich Tasks from Demonstrations. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'21)*, pages 7843–7850, Prague, Czech Republic

2. Ablett, T., Limoyo, O., Sigal, A., Jilani, A., Kelly, J., Siddiqi, K., Hogan, F., and Dudek, G. (2024b). Multimodal and Force-Matched Imitation Learning With a See-Through Visuotactile Sensor. *IEEE Transactions on Robotics*, 41:946–959

3. Ablett, T., Marić, F., and Kelly, J. (2020). Fighting Failures with FIRE: Failure Identification to Reduce Expert Burden in Intervention-Based Learning. Technical Report STARS-2020-001, University of Toronto

4. Ablett, T., Chan, B., and Kelly, J. (2021a). Learning from Guided Play: A Scheduled Hierarchical Approach for Improving Exploration in Adversarial Imitation Learning. In *Proceedings of the Neural Information Processing Systems (NeurIPS'21) Deep Reinforcement Learning Workshop*, Online

5. Ablett, T., Chan, B., and Kelly, J. (2023). Learning From Guided Play: Improving Exploration for Adversarial Imitation Learning With Simple Auxiliary Tasks. *IEEE Robotics and Automation Letters*, 8(3):1263–1270

6. Ablett, T., Chan, B., Wang, J. H., and Kelly, J. (2024a). Fast Reinforcement Learning without Rewards or Demonstrations via Auxiliary Task Examples. In *CoRL 2024 Workshop on Mastering Robot Manipulation in a World of Abundant Data*, Munich, Germany

7. Ablett, T., Chan, B., Wang, J. H., and Kelly, J. (2025). Efficient Imitation Without Demonstrations via Value-Penalized Auxiliary Control from Examples. In *IEEE International Conference on Robotics and Automation (ICRA'25)*, Atlanta, GA, USA

Chapter 2

Background

In this chapter, we describe the key concepts and terminology that the later chapters in the dissertation build upon. We start by briefly discussing supervised learning. Next, we present background on sequential decision making, which provides context for sections on reinforcement learning and imitation learning that follow.

2.1 Supervised Learning

Supervised learning is a form of machine learning in which the goal is to learn a mapping, or model, f from inputs $\mathbf{x} \in \mathcal{X}$ to outputs (or labels) $\mathbf{y} \in \mathcal{Y}$, given a training dataset of N input-output pairs, $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N}$ (Murphy, 2022). In this dissertation, examples of $\mathcal{X}$ (or parts of $\mathcal{X}$ containing multiple modalities) include raw RGB camera images, robot poses, and object poses, while examples of $\mathcal{Y}$ include robot movement control and gripper movement control.

Supervised learning requires some form of loss function $\ell\left(\mathbf{y}_n, f(\mathbf{x}_n; \boldsymbol{\theta})\right)$ for determining the error between a true label $\mathbf{y}_n$ and predicted output $f(\mathbf{x}_n; \boldsymbol{\theta})$, given model parameters $\boldsymbol{\theta}$. Generally, ℓ is used both for both evaluating and improving the quality of f. More specifically, training a supervised model can be defined as finding a setting of parameters $\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ that minimizes the average training set loss

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \ell\left(\mathbf{y}_n, f(\mathbf{x}_n; \boldsymbol{\theta})\right), \tag{2.1}$$

2.1.1 Classification

A classification problem constrains $\mathcal{Y}$ to a discrete set of C unordered categories or classes, $\mathcal{Y} = \{1, 2, \ldots, C\}$. If $\mathcal{Y}$ contains only two classes, the problem can be equivalently referred to as *binary classification*. It is desirable to maintain a probabilistic estimate of our predictions $f(\mathbf{x}_n; \boldsymbol{\theta})$ to give us a notion of uncertainty, which we can describe by the conditional distribution

$$p(y = c \mid \mathbf{x}; \boldsymbol{\theta}) = f_c(\mathbf{x}; \boldsymbol{\theta}), \tag{2.2}$$

where $f : \mathcal{X} \to [0, 1]^C$ maps $\mathcal{X}$ to a probability distribution over the C possible output labels, and therefore $\sum_{c=1}^{C} f_c = 1$. Given the unconditional distribution $f(\mathbf{x}; \boldsymbol{\theta})$, we can describe ℓ as the negative log probability

$$\ell(y, f(\mathbf{x}; \boldsymbol{\theta}) = -\log p(y \mid f(\mathbf{x}; \boldsymbol{\theta})), \tag{2.3}$$

with a corresponding negative log loss (NLL) of the training set given by

$$\text{NLL}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^{N} \log p(y \mid f(\mathbf{x}; \boldsymbol{\theta})), \tag{2.4}$$

which allows us to generate the maximum likelihood estimate (MLE), $\hat{\boldsymbol{\theta}}_{\text{mle}} = \arg\min_{\boldsymbol{\theta}} \text{NLL}(\boldsymbol{\theta})$. If f is a linear model, this approach is known as logistic regression. In this dissertation, classification losses are used to train discriminators, which are further described in Section 2.4.4 and play a large role in Chapters 5 to 7.

2.1.2 Regression

A regression problem allows $y \in \mathcal{Y}$ to contain any real number $\mathbb{R}$, and requires a different loss function from classification.[1] In many cases, the ℓ_2 (quadratic) loss function

$$\ell_2 \left(y_n, f(\mathbf{x}_n; \boldsymbol{\theta})\right) = \left(y_n - f(\mathbf{x}_n; \boldsymbol{\theta})\right)^2 \tag{2.5}$$

is sufficient, or even optimal, giving us the mean squared error (MSE) for our training set loss when we substitute Eq. (2.5) as our ℓ in Eq. (2.1).

The most common choice of probabilistic model for regression is to assume that the output y follows a Gaussian (normal) distribution, defined by

$$\mathcal{N}(y \mid \mu, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}, \tag{2.6}$$

where μ is the mean of the distirubtion and σ^2 is its variance. We can then make μ, and optionally σ^2, depend on our model as $\mu = f_\mu(\mathbf{x}_n; \boldsymbol{\theta})$ and $\sigma^2 = f_{\sigma^2}(\mathbf{x}_n; \boldsymbol{\theta})$. If σ^2 is fixed, then the NLL for regression is proportional to the MSE. If f is a linear model, this approach is known as linear regression, where $f(x; \boldsymbol{\theta}) = \mathbf{w}^\top \mathbf{x} + b$ and the learnable parameters are $\boldsymbol{\theta} = \{\mathbf{w}, b\}$. If f instead contains M higher-order terms, the approach is known as polynomial regression, where $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$ and $\boldsymbol{\phi}$ is a feature vector $\boldsymbol{\phi}(x) = \left[1, x, x^2, \ldots, x^M\right]$.

In this dissertation, regression losses are used for training behaviour policies with behavioural cloning in Chapters 3 to 5, as well as for training value functions in inverse reinforcement learning (IRL) in Chapters 6 and 7. Policy actions in IRL are sampled from a Gaussian distribution, where a neural network learns both μ and σ^2. More details are given in Section 2.3, Section 2.4.1 and Section 2.4.4.

[1] For notational convenience, this section deals with one-dimensional outputs y, but the extension to multidimensional outputs $\mathbf{y}$ is straightforward.

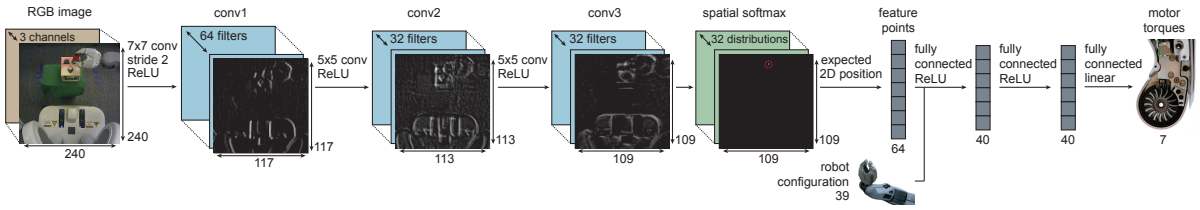


Figure 2.1: The deep neural network architecture used in (Levine et al., 2016), including a series of convolutional layers followed by a series of fully connected layers. We show that a similar architecture can be extended from fixed-view manipulation to mobile manipulation in Chapter 3.

2.1.3 Deep Learning

If we wish to model more complicated relationships with $\phi(\mathbf{x})$, we can compose multiple simple functions together and allow $\phi(\mathbf{x})$ to have its own parameters. We can describe our model f as a composition of simpler functions,

$$f(\mathbf{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\cdots(f_1(\mathbf{x}))\cdots)), \tag{2.7}$$

where f_l is the function at layer l and the final layer f_L is linear, with the form $f_L(\mathbf{x}) = \mathbf{w}^\top f_{1:L-1}(\mathbf{x})$. To model non-linear relationships, non-linear differentiable functions (referred to as *activation* functions in deep learning literature) are added at each layer, meaning that a typical layer f_l has the form

$$\mathbf{z}_l = f_l(\mathbf{z}_{l-1}) = \varphi_l(\mathbf{b}_l + \mathbf{W}_l\mathbf{z}_{l-1}). \tag{2.8}$$

Here, $\mathbf{z}_l$ is known as the hidden output at layer l, φ_l is a differentiable, non-linear function applied per-element, $\mathbf{b}_l \in \mathbb{R}^{M_{l_{\text{out}}}}$ is the learned bias vector at layer l, $\mathbf{W}_l \in \mathbb{R}^{M_{l_{\text{out}}}} \times \mathbb{R}^{M_{l_{\text{in}}}}$ is the learned weight matrix at layer l, and $M_{l_{\text{out}}}$ and $M_{l_{\text{in}}}$ are fixed *hyperparameters*.[2] A neural network with this specific form is known as a *fully connected network*. Because the whole network is differentiable, training can be completed via *backpropagation* (Rumelhart et al., 1986): the loss or error is computed with, for example, Eq. (2.4) or Eq. (2.5), the gradient of the weights of the entire network are determined via the chain rule, and the network weights are updated to minimize the loss function via gradient descent.

A universal approximation theorem has provided proof that a sufficiently wide single-layer neural network can represent any function (Hornik et al., 1989) to arbitrary accuracy, but ongoing research in the community has found that a variety of deeper networks, with many variations on specific architectures, obtain far higher performance than wide single-layer networks. The literature has varying hypotheses for this phenomenon, with explanations typically alluding to the hierarchical structure of many forms of real data: in a robotic domain, for example, an image make contain local edges, which can be compiled together to generate semantic objects such as a door or drawer, which can then be interpreted as being opened or closed.

In this dissertation, all chapters introduce methods that use deep neural networks for function

[2] A hyperparameter is a term from machine learning literature, referring to a value which is hand-selected or tuned and then fixed throughout the learning process. The word *hyper* indicates that they are "above" the *weights*, which are the *model* parameters that are updated autonomously by the algorithm itself.

approximation. We use simple fully connected networks (i.e., as described above) in Chapters 5 to 7. We combine fully connected networks with convolutional neural networks (CNNs) in Chapters 3 to 5 (see Fig. 2.1 for an example). CNNs perform a two-dimensional discrete convolutional operation, making them well-suited for image-based data (Murphy, 2022).[3]

2.1.4 Generalization and Distribution Shift

Our goal in supervised learning is to minimize a loss function on a particular dataset. In reality, reducing the loss is only a proxy for our true goal: achieving high prediction accuracy on new, previously-unseen, data. With a highly flexible model, such as a deep neural network, it is theoretically possible to reduce training loss to 0. Typically, in a process known as *overfitting*, this comes at the expense of creating a model that will likely not generalize to new data beyond $\mathcal{D}$. To avoid overfitting, it is common to split the dataset $\mathcal{D}$ into a *training set* $\mathcal{D}_{\text{train}}$, a *test set* $\mathcal{D}_{\text{test}}$, and *validation set* $\mathcal{D}_{\text{valid}}$. $\mathcal{D}_{\text{train}}$ is used to directly change the model parameters $\boldsymbol{\theta}$, $\mathcal{D}_{\text{test}}$ is used to estimate model performance after training, and $\mathcal{D}_{\text{valid}}$ is used for model selection during training, but not for changing $\boldsymbol{\theta}$ directly. Performance on $\mathcal{D}_{\text{test}}$ is also known as *test risk*.

In real world applications, including robotics, a trained machine learning model may encounter new data that does not resemble anything from $\mathcal{D}$, meaning that our test risk is no longer accurate. This phenomenon is referred to as *distribution shift* or *dataset shift* (Quionero-Candela et al., 2009; Zhang et al., 2023), and data that are sufficiently different from training data are referred to as *out-of-distribution*. We can further divide distribution shift into (i) *covariate shift*: a change in the distribution of inputs $p(\mathbf{x})$, (ii) *label shift*: a change in the distribution of outputs $p(\mathbf{y})$, and (iii) *concept shift*: a change in the relationship between $p(\mathbf{x})$ and $p(\mathbf{y})$. Any of these factors can cause a machine learning model to perform far worse than as estimated via our test risk. In this dissertation, Chapters 3 to 7 all introduce methods meant to handle potential covariate shift.

2.2 Sequential Decision Making and Markov Decision Processes

Section 2.1 describes a general form of data $\mathcal{D}$ where $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ pairs are *independent and identically distributed* (*IID*). Many domains involve sequential data, where we will instead define our data in terms of input *states* $s \in \mathcal{S}$ and output *actions* $a \in \mathcal{A}$ (where, compared with $\mathbf{x}$ and $\mathbf{y}$, we have also dropped the vector notation for convenience, but $\mathcal{S}$ and $\mathcal{A}$ can be multi-dimensional). Sequential decision making has three major departures from supervised learning: (i) a notion of time is introduced, where, for example, a data pair (s_t, a_t) now has an associated discrete[4] timestep t, (ii) actions a_t are executed in an environment, and have an effect on future observations $s_{t+1:\infty}$, and (iii) model performance is usually measured in terms of a *reward* function, which we attempt to maximize over time (Hardt and

[3] Further detail of the nuances of neural network architectures, including layer types, activations functions, and layer sizes are beyond the scope of this dissertation, and we refer the reader to (Murphy, 2022; Goodfellow et al., 2016) for a more thorough treatment.

[4] Of course, there are sequential decision making tools for handling continuous-time problems, but they are beyond the scope of this dissertation.

Recht, 2022).

Finite Markov decision processes (MDPs) provide a convenient formalization of sequential decision making that will be used heavily throughout this dissertation, which we summarize here, largely following (Sutton and Barto, 2018; Achiam, 2018). This section describes three aspects of the problem: an environment (the MDP itself) which outputs a state and reward given an action, an agent which outputs an action given a state, and a value function which estimates how much reward a state or action could eventually generate.[5]

2.2.1 Environment

We can define an MDP, or environment, as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, R, \mathcal{P}, \rho_0, \gamma \rangle$. The sets $\mathcal{S}$ and $\mathcal{A}$ are the state and action space and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function. $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the state-transition environment dynamics distribution, where $\mathcal{P}(s' \mid s, a)$ tells us the probability of the next state s' occurring given the current state s and action a. The initial state distribution $\rho_0 : \mathcal{S} \to [0, 1]$ generates the first state that is provided to the agent. The discount factor $\gamma \leq 1$ is a parameter that is tuned to increase the priority of rewards that occur sooner, and also to ensure that the return of an infinite-horizon MDP has a finite value. An MDP can be *episodic* or *continuing*: in episodic tasks, the environment is "reset" using ρ_0 after a condition is met (typically successful completion of the task, failure, or time-out), while in continuing tasks, the environment never resets. The name *Markov* decision problem refers to the idea that the system obeys the Markov property, meaning each state s_{t+1} only depends on s_t and a_t.[6]

Partial Observability

There are many MDPs for which the agent receives only partial *observations*, rather than states, of the environment; such MDPs are referred to as *partially observable* (POMDPs). In POMDPs, the agent receives observations $o \in \mathcal{O}$, which depend on the state but do not necessarily contain all state-relevant information. For example, in a manipulation task, a camera image of a scene could be an observation, and the corresponding poses of the objects seen in the image could be the state. While there is substantial literature investigating a formal treatment of POMDPs, the remaining analysis in this section, which focuses exclusively on MDPs, can be applied directly to POMDPs in the case of using function approximation (Sutton and Barto, 2018).

2.2.2 Agent, Trajectories, and Return

The agent includes a policy distribution $\pi(a \mid s)$ or function $\pi(s)$, which interacts with $\mathcal{P}$ and R to yield experience (s_t, a_t, r_t, s_{t+1}) for $t = 0, \ldots, T$, where $s_0 \sim \rho_0(\cdot), a_t \sim \pi(\cdot \mid s_t), s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t), r_t = R(s_t, a_t)$, and T is either a terminal timestep (for episodic tasks) or $T = \infty$ (for continuing

[5] These concepts equivalently exist in optimal control theory as controller (agent), plant (environment), cost (reward), and control signal (action).

[6] In practice, this turns out to not always be true for systems in which we apply this formalism, but there are workarounds.

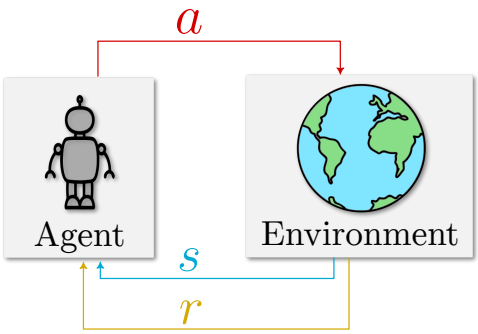


Figure 2.2: The agent-environment loop in a Markov decision process. Adapated from Sutton and Barto (2018).

tasks). We can define a trajectory starting from time t as $\tau_t = \{(s_t, a_t), (s_{t+1}, a_{t+1}), \dots (s_T, a_T)\}$, and can further define *return* G as the sum of rewards for following trajectory τ_t,

$$\begin{aligned} G(\tau_t) &\triangleq \sum_{k=t}^{T} \gamma^{k-t} R(s_k, a_k), \\ &= R(s_t, a_t) + \gamma G(\tau_{t+1}), \end{aligned} \tag{2.9}$$

where we have also included a recursive definition of G. The aim in an MDP is to learn an optimal deterministic policy $\pi^*(s)$ that maximizes the expected return starting from the initial state

$$\begin{aligned} \pi^*(s) &\triangleq \arg\max_{\pi} \mathbb{E}_{\pi} \left[G(\tau_0) \right], \\ &= \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{k=0}^{T} \gamma^k R(s_k, a_k) \right], \end{aligned} \tag{2.10}$$

where $\mathbb{E}_{\pi}[\cdot]$ denotes the expected value of following policy π in the environment. Formally,

$$\mathbb{E}_{\pi}[\cdot] \triangleq \mathbb{E}_{s_0 \sim \rho_0} \left[\mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} \left[\mathbb{E}_{s_1 \sim \mathcal{P}(\cdot|s_0,a_0)} \left[\mathbb{E}_{a_1 \sim \pi(\cdot|s_1)} \left[\mathbb{E}_{s_2 \sim \mathcal{P}(\cdot|s_1,a_1)} \left[\cdots [\cdot] \cdots \right] \right] \right] \right], \tag{2.11}$$

$$= \sum_{s_0 \in \mathcal{S}} \rho_0(s_0) \sum_{a_0 \in \mathcal{A}} \pi(a_0 \mid s_0) \sum_{s_1 \in \mathcal{S}} \mathcal{P}(s_1 \mid s_0, a_0) \sum_{a_1 \in \mathcal{A}} \pi(a_1 \mid s_1) \sum_{s_2 \in \mathcal{S}} \mathcal{P}(s_2 \mid s_1, a_1) \cdots [\cdot], \tag{2.12}$$

which factors into terms that only rely on the current state (e.g., $\pi(a_t \mid s_t)$) due to the Markov property.

2.2.3 The Value Functions and the Bellman Equations

To maximize the expected return in Eq. (2.10), many algorithms make use of the *value* function of a particular state,

$$\begin{aligned} V^{\pi}(s) &\triangleq \mathbb{E}_{\pi} \left[G(\tau_t) \mid s_t = s \right], \\ &= \mathbb{E}_{\pi} \left[R(s, a) + \gamma G(\tau_{t+1}) \mid s_{t+1} = s' \right], \\ &= \mathbb{E}_{a \sim \pi(\cdot|s), s' \sim \mathcal{P}(\cdot|s,a)} \left[R(s, a) + \gamma V^{\pi}(s') \right], \end{aligned} \tag{2.13}$$

which we have also defined recursively using Eq. (2.9), and used the notation $G(\tau_t) \mid s_t = s$ to refer to the return of trajectory τ_t where $s_t = s$. This recursive formulation is also known as the *Bellman equation* for value (Bellman, 1957). Similarly, the *action-value* of a particular state-action pair is

$$\begin{aligned} Q^\pi(s,a) &\triangleq \mathbb{E}_\pi \left[G(\tau_t) \mid s_t = s, a_t = a \right], \\ &= \mathbb{E}_\pi \left[R(s,a) + \gamma G(\tau_{t+1}) \mid s_{t+1} = s', a_{t+1} = a' \right], \\ &= R(s,a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a), a' \sim \pi(\cdot|s')} \left[Q^\pi(s',a') \right]. \end{aligned} \tag{2.14}$$

Both V^π and Q^π average the return of all possible states and actions, weighing the return of each partial trajectory by the probability that it occurs. We can relate V^π and Q^π, respectively, as

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)} \left[Q^\pi(s,a) \right], \tag{2.15}$$

$$Q^\pi(s,a) = R(s,a) + \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \left[V^\pi(s') \right]. \tag{2.16}$$

Finally, V^π and Q^π have corresponding *optimal* value functions that are defined, respectively, as

$$\begin{aligned} V^*(s) &\triangleq \max_\pi V^\pi(s) \\ &= \max_a \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \left[R(s,a) + \gamma V^*(s') \right], \end{aligned} \tag{2.17}$$

$$\begin{aligned} Q^*(s,a) &\triangleq \max_\pi Q^\pi(s,a) \\ &= R(s,a) + \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \left[\gamma \max_{a'} Q^*(s',a') \right], \end{aligned} \tag{2.18}$$

where V^* and Q^* define the maximum expected value of the return given a starting state of s or state-action pair of (s,a).

For most practical systems, precisely finding π^*, V^*, or Q^* with Eqs. (2.10), (2.17) and (2.18) is intractable. Sections 2.3 and 2.4 describe practical solutions, using reinforcement learning and imitation learning, respectively, for solving Eq. (2.10) by approximating Eqs. (2.17) and (2.18).

2.3 Reinforcement Learning

Reinforcement learning (RL) is an approach to solving decision making problems via trial and error. RL algorithms generally follow a simple, repeating pattern: (i) (optionally) collect experience as an agent executes actions in the environment, (ii) calculate the value of the experience (*policy evaluation*), and (iii) update the agent's policy to maximize the expected value (*policy improvement*). In this section, we will discuss how this procedure, known as *generalized policy iteration* (GPI), can be applied to sequential MDPs through temporal difference (TD) methods (Sutton and Barto, 2018). [7] This discussion will build up to modern approaches to TD learning, which are used as the the backbone for the algorithms

[7] Although a branch of RL focuses on the nonassociative, non-sequential setting (known as multi-armed bandits), in which actions do not affect future timesteps, discussion of this topic is beyond the scope of this dissertation.

Algorithm 1 Generalized Policy Iteration (Sutton and Barto, 2018)

Initialize $V(s)$ and $\pi(s)$ randomly for all $s \in \mathcal{S}$.
policy-stable $\leftarrow$ false
while not policy-stable **do**
  policy-stable $\leftarrow$ true
  (Potentially partial) policy evaluation sweep through $\mathcal{S}$
  for $s \in \mathcal{S}$ **do**
    $V(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, \pi(s)) \left[V(s')\right]$ (Eq. (2.13))
  end for
  (Potentially partial) policy improvement sweep through $\mathcal{S}$
  for $s \in \mathcal{S}$ **do**
    old-action $\leftarrow \pi(s)$
    $\pi(s) = \arg\max_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) \left[V(s')\right]$ (Eq. (2.17))
    if old-action $\neq \pi(s)$ **then**
      policy-stable $\leftarrow$ false
    end if
  end for
end while
Output: $\pi \approx \pi^*, V \approx V^*$

introduced in Chapters 6 and 7. We also include material on reward design and hierarchical agents, which are also relevant to Chapters 6 and 7.

2.3.1 Generalized Policy Iteration

As stated, generalized policy iteration (GPI) is a broad description of how most RL algorithms function: an iterative pattern of (i) (optionally) collecting experience with a policy, (ii) policy evaluation, and (iii) policy improvement. Historically, these algorithms were part of *dynamic programming* (DP) (Bellman, 1957), in which the environment dynamics $\mathcal{P}$ are assumed to be perfectly accurate, removing the need to actually collect experience.

GPI is called *policy iteration* if the policy evaluation sweep in Algorithm 1 is executed until convergence. Alternatively, it is called *value iteration* if the update rule of the policy evaluation sweep is replaced with

$$V(s) \leftarrow \max_a R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) \left[V(s')\right], \tag{2.19}$$

where we have replaced $\pi(s)$ with a, and include the $\max_a$ term to directly incorporate policy improvement and policy evaluation together. Policy evaluation can be switched to policy improvement without having gone through every $s \in \mathcal{S}$, and vice versa. Regardless of whether full sweeps are completed, GPI is known to converge to the optimal value function and optimal policy (Sutton and Barto, 2018).

Unfortunately, for most environments we are interested in, $\mathcal{S}$ or $\mathcal{A}$ (or both) are not finite and $\mathcal{P}$ is not perfectly accurate, meaning that we cannot directly apply Algorithm 1. To address these issues, in Section 2.3.2, we turn to modern RL algorithms that incorporate environment exploration and function approximation, such as with deep neural networks (see Section 2.1.3).

2.3.2 Temporal Difference Learning

Many aspects of most practical problems make it impractical or impossible to apply Algorithm 1 directly. Most problems either do not have a finite state space $\mathcal{S}$, or if they do, it is intractable to iterate through the entire space. Furthermore, the environment dynamics $\mathcal{P}$ are usually inaccurate or unknown. To address both of these limitations, we can introduce the collection of *experience*, where an agent selects actions in the environment, generating (s, a, r, s'), and we use the experience collected to evaluate and improve the policy. *Temporal difference* (TD) learning is a core approach to RL that continuously improves the estimate for $V(s)$ or $Q(s, a)$ with collected experience, rather than a known environment model.

Assuming we have experience (s, a, r, s'), where $a \sim \pi(\cdot \mid s)$, $r = R(s, a)$, and $s' \sim \mathcal{P}(\cdot \mid s, a)$ (but we can only sample from $\mathcal{P}$), we can generate a *temporal difference* (TD) rule for optimally updating a Q function:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right], \tag{2.20}$$

where α is a parameter to determine the update rate. Eq. (2.20) is the update rule in *Q-learning* (Watkins and Dayan, 1992), which uses *bootstrapping*: $Q(s, a)$ is updated using $Q(s', a')$, as is done with $V(s)$ and $V(s')$ in Algorithm 1. A modern breakthrough using Eq. (2.20) for playing Atari games is Deep Q-Networks (DQN) (Mnih et al., 2015), which approximates $Q(s, a)$ using a deep neural network.[8]

Exploration

With TD learning, we can no longer guarantee that we update $Q(s, a)$ using all possible state-action pairs in $\mathcal{S}$ and $\mathcal{A}$ as we did in Algorithm 1, so we require a means of *exploring* the environment. Environment exploration is an active research area in RL, but a simple approach that works for many cases with a discrete action space is ε*-greedy* action selection. If $U([0, 1])$ is a uniform distribution between 0 and 1, and $U(\mathcal{A})$ is a discrete uniform distribution over the action space, our policy is:

$$\pi(a \mid s) = \begin{cases} \max_a Q(s, a), & \text{if } x \geq \varepsilon, \\ a \sim U(\mathcal{A}), & \text{otherwise,} \end{cases} \tag{2.21}$$

where $0 \leq \varepsilon \leq 1$ is a parameter that can be initialized high to encourage exploration, and gradually lowered to move towards a greedy policy, which still guarantees convergence in GPI (Sutton and Barto, 2018).

Actor-Critic Approaches for Continuous Action Spaces

So far, this section has dealt exclusively with discrete, finite action spaces $\mathcal{A}$, but many environments (including robotics environments) have a continuous action space. In these cases, we cannot use

[8] Mnih et al. (2015) also incorporates several other improvements. See remark on *the deadly triad* at the end of *Off-Policy Actor-Critic* in this section for more details.

Eq. (2.20) directly: when $\mathcal{A}$ is discrete, we can simply sample all values of $a \in \mathcal{A}$ and choose the maximum, but when $\mathcal{A}$ is continuous, $\max_a Q(s,a)$ becomes non-trivial. A simple solution is to switch to state-action-reward-state-action (SARSA) (Rummery and Niranjan, 1994), defined as such because we use the experience (s, a, r, s', a'):

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s', a') - Q(s,a) \right], \tag{2.22}$$

where the only difference from Eq. (2.20) is the removal of the $\max_a$ term, and the use of the true next action a'.

Since we can no longer easily estimate $\max_a Q(s,a)$, continuous action spaces require us to choose a new approach to generating a behaviour policy. Following Lillicrap et al. (2016), we parametrize our policy as a separate differentiable model $\pi_\theta(s)$ with parameters θ and unconstrained output, and after updating $Q(s,a)$, we can update θ via gradient ascent as

$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta Q(s, \pi_\theta(s)), \tag{2.23}$$

where α_θ is the policy learning rate and ∇_θ is the gradient with respect to θ. Approaches that use separate models for Q and π are known as *actor-critic* methods (where π is the actor, and Q or V is the critic). The use of Eq. (2.22) is an *on-policy* approach to TD learning, meaning that the *behaviour* policy used to generate the experience is the same as the *target* policy used to update the model.

Off-Policy Actor-Critic

On-policy methods can be very inefficient, since collected (s, a, r, s', a') must be discarded after π has been updated.[9] It is much more desireable to use *off-policy* methods, where the target policy can be different from the behaviour policy which generated the experience. The approach taken with deep deterministic policy gradient (DDPG), as well as other popular methods that build on it (Fujimoto et al., 2018; Haarnoja et al., 2018, 2019), is to use the policy in the update for Q. Assuming Q_ϕ is a differentiable model and we have experience (s, a, r, s'), we update ϕ using the following regression loss

$$\ell((s,a,r,s'), Q_\phi) = \left(Q_\phi(s,a) - \left(r + \gamma Q_{\phi_{\text{targ}}} \left(s', \pi_{\theta_{\text{targ}}}(s') \right) \right) \right)^2, \tag{2.24}$$

where $Q_{\phi_{\text{targ}}}$ and $\pi_{\theta_{\text{targ}}}$ are copies of Q_ϕ and π_θ that are updated slowly (see Eq. (2.29)). Eqs. (2.23) and (2.24) correspond to policy improvement and policy evaluation, respectively, from Algorithm 1. Since $\pi(s)$ is deterministic in Eqs. (2.23) and (2.24), and Eq. (2.21) only applies to discrete $\mathcal{A}$, a popular approach to encourage exploration is to add zero-mean Gaussian noise to the actions as

$$a = \pi(s) + x, \quad x \sim \mathcal{N}(0, \sigma), \tag{2.25}$$

[9] Specifically, they must be discarded because once the policy has changed, there is no guarantee that $a' = \pi(s')$, meaning that using (s, a, r, s', a') to calculate $Q(s,a)$ will no longer be correct.

where σ must be tuned for each MDP, and the covariance is (typically) diagonal if $\mathcal{A}$ is multidimensional.

An alternative, originally introduced by Williams and Peng (1991), is to learn σ via a policy that maximizes *entropy* in addition to maximizing return. Information entropy, introduced by Shannon (1948), is a measure of the average uncertainty of a random variable, defined as

$$\begin{aligned} H(p) &= \mathbb{E}_{x\sim p}\left[-\log p(x)\right] \\ &= -\sum_{x\in\mathcal{X}} p(x)\log p(x). \end{aligned} \tag{2.26}$$

The higher H is, the less predictable x is. Intuitively, by simultaneously maximizing both entropy and reward, a policy acts as randomly as possible while still completing the task, encouraging the policy to explore, rather than settle for local maximums. We can modify our bootstrapping policy evaluation regression objective in Eq. (2.24) as

$$\ell((s,a,r,s'),Q_\phi) = \left(Q_\phi(s,a) - \left(r+\gamma\left(Q_{\phi_{\text{targ}}}(s',a') - \alpha_H \log\pi(a'\mid s')\right)\right)\right)^2, \quad a'\sim\pi(\cdot\mid s'), \tag{2.27}$$

and our policy improvement step in Eq. (2.23) as

$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta Q(s,a) - \alpha_H \log\pi_\theta(a\mid s), \quad a\sim\pi_\theta(\cdot\mid s), \tag{2.28}$$

where α_H is a parameter that controls the relative contribution of the entropy bonus, $\pi(a \mid s)$ is parametrized as a Gaussian distribution, and differentiability of $\pi(a \mid s)$ is maintained through the *reparemetrization trick* (Kingma et al., 2015). This approach was introduced by Haarnoja et al. (2018) as soft actor-critic (SAC). Notably, we have substituted one hand-crafted parameter (σ) for another (α_H), but in practice, we use a modified version of SAC, introduced by Haarnoja et al. (2019), where α_H is also learned. SAC has been shown to produce efficient learning across a variety of environments, so we use it as the primary backbone algorithm in Chapters 6 and 7.

2.3.3 Reward Design

In reinforcement learning (RL), the reward function $R(\cdot)$ is a crucial part of the learning process. Much research in RL is focused on algorithm design, but without an effective reward function, an optimal policy may never be learned. Many tasks where RL has achieved groundbreaking success involve reward functions that were already clearly defined by the environment, such as score in a video game (Mnih et al., 2015) or winning at a board game (Silver et al., 2016). Even in these regimes, however, the known rewards may not automatically lead to optimal or even sufficiently effective policies. This difficulty is summarized by the *credit assignment problem* (Minsky, 1961): how should credit be distributed for success among a (potentially large) number of decisions? From the perspective reward design, an attempt to answer this question has led practitioners to divide rewards into two categories: *sparse* and *dense*. Sparse rewards provide feedback only upon successful completion of a task, while

The "deadly triad" in RL, experience replay, and delayed target updates

Sutton and Barto (2018) have noted that the combination of bootstrapping, function approximation, and off-policy learning results in instability that can significantly hinder or stop policy learning altogether. Nonetheless, many modern approaches each use all three of these tools to achieve state-of-the-art results (Mnih et al., 2015; Fujimoto et al., 2018; Haarnoja et al., 2018). Although many design choices have helped improve stability in this challenging regime, two particular improvements, originally introduced by Mnih et al. (2015), are *experience replay* and *delayed target updates*.

Experience replay: instead of updating $Q(s, a)$ as each s is encountered and a is executed, all encountered (s, a, r, s') tuples are stored in a *replay buffer* $\mathcal{B}$, which we sample from to update Q (and possibly π). We can optionally set a maximum size for the replay buffer (making it first-in-first-out), depending on the task.

Delayed target updates: rather than directly updating $Q_\phi(s, a)$ via bootstrapping from $Q_\phi(s', a')$ (and possibly $a' = \pi_\theta(s')$), $Q_{\phi_{\text{targ}}}(s', a')$ and $\pi_{\theta_{\text{targ}}}(s')$ are used for calculating bootstrap estimates, and the weights ϕ_{targ} and θ_{targ} are updated gradually using, for example, polyak averaging:

$$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho\, \phi_{\text{targ}} + (1 - \rho)\, \phi, \quad 0 \leq \rho < 1, \\ \theta_{\text{targ}} &\leftarrow \rho\, \theta_{\text{targ}} + (1 - \rho)\, \theta, \quad 0 \leq \rho < 1. \end{aligned} \tag{2.29}$$

dense rewards provide feedback throughout the task.

Consider an illustrative example: say we have a robot that we would like to wash a set of dishes for us. A sparse reward could be

$$R(\cdot) = \begin{cases} 1, & \text{if all dishes are cleaned and placed in the dish rack,} \\ 0, & \text{otherwise,} \end{cases} \tag{2.30}$$

while a dense reward could be

$$R(\cdot) = \sum_{d \in \text{dishes}} \texttt{cleanliness}(d) - (\texttt{loc}(d) - \texttt{loc}(\text{dish rack}))^2, \tag{2.31}$$

where `cleanliness`() and `loc`() are functions that evaluate how clean a dish is and provide the location of an object, respectively.

Providing only a sparse reward results in a challenging exploration problem (Minsky, 1961), leading to potentially inefficient or even completely stalled learning. Consider the above example in the case when there are a dozen or more dishes, and $\mathcal{A}$ are the motor torques for 6-DOF robotic arm: through random exploration alone, it would take an inordinate amount of time for the agent to receive non-zero reward with Eq. (2.30), and it would likely damage itself and many dishes along the way. On

the other hand, while Eq. (2.31) is only maximized when all dishes are clean and placed on the dish rack, Eq. (2.31) provides a *partial* signal if some dishes (or even one dish) are close to the dish rack or partly cleaned. Providing a dense reward is related to the concept of *shaping*, introduced by Skinner (1953) for training animals. Shaping was designed to address the difficulty of using sparse rewards. In shaping, the reward function is progressively modified by the practitioner to ensure that the agent always receives some reward for partial completion of a task. A dense reward provides a similar benefit, but is fixed throughout training. Indeed, both dense rewards and shaping can significantly accelerate learning (Sutton and Barto, 2018; Ng and Jordan, 2003).

Unfortunately, both dense rewards and shaping require careful design by a practitioner, and the negative consequences of poorly designed reward functions are not always immediately obvious. Dense rewards have been shown to be highly subject to a phenomenon known as *reward hacking* (Skalse et al., 2022), in which an agent *exploits* the reward function, gaining high reward as defined but not actually completing the desired task. In our dishwashing example, the robot may learn to throw all of the dishes at the dish rack, breaking each one, but still maximizing the $(\texttt{loc}(d) - \texttt{loc}(\text{dish rack}))^2$ term from Eq. (2.31). Shaped rewards require constant interaction between a system and a practitioner, and may still lead to reward hacking if one is not careful. In summary, sparse, dense, and shaped rewards each have pros and cons, and none are guaranteed to work in all cases. In Chapters 5 to 7, we focus on an alternatives to hand-crafted rewards, where rewards are learned from human demonstrations (Chapters 5 and 6) or human-provided examples of success (Chapter 7). We describe further background on this process, known as *inverse reinforcement learning*, in Section 2.4.4.

2.3.4 Hierarchical Agents

The MDP framework and RL are appealing because they are quite general and can be applied to many different systems. From a practical perspective, however, MDPs and RL do not address how to actually choose the state space $\mathcal{S}$ and action space $\mathcal{A}$. In many cases, a hierarchical decomposition of $\mathcal{S}$, $\mathcal{A}$, or both, may be beneficial for efficiently learning an effective agent (Nachum et al., 2019).

Consider, again, our dishwashing robot example from Section 2.3.3: we might choose $\mathcal{A}$ to be the joint torques of the robot arm. Imagine that two of our dishes are identically shaped plates, and our sensor system is capable of detecting the poses of these plates. As described, our policy π would need to output different joint torques for moving towards and grasping each of these two plates, even if many aspects of the motion to grasp each plate are similar. Instead, we could predefine reach and grasp as separate subtasks, sometimes defined as *options* (Sutton et al., 1999). In this case, we could define new MDPs: $\mathcal{M}_{\text{dishwasher}}$, which has a new $\mathcal{A}$ consisting of the selection of a target dish d and an option policy π_ω, where $\omega \in \{\text{reach}, \text{grasp}, \dots\}$, and $\mathcal{M}_{\text{reach}}$ and $\mathcal{M}_{\text{grasp}}$, that have dish d appended to their state space $\mathcal{S}$ and a reward function R_ω that is specific to their task. These option policies can also have corresponding value functions Q_ω. The new MDPs do not have to be predefined; the choice of whether to predefine or learn them is up to the designer, and is generally an open research question.

The learning methods described throughout this section can be modified to accommodate hierarchical agents, but the explicit details of doing so are beyond the scope of this section. In Chapters 6

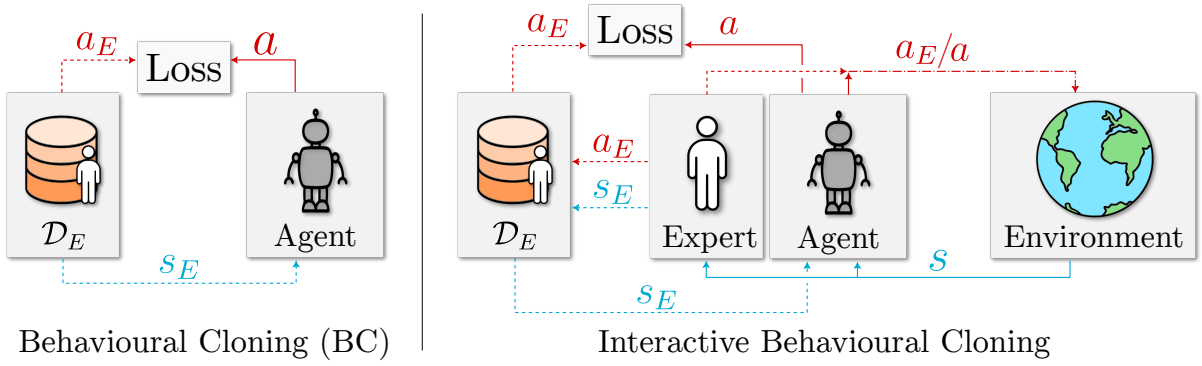


Figure 2.3: Modifications to the agent-environment loop in behavioural cloning (BC) and interactive BC. In interactive BC, the action executed in the environment is either the expert's or the agent's, and new expert actions (and states) are continuously appended to the expert dataset $\mathcal{D}_E$. BC is used as the base algorithm in Chapters 3 and 4, and interactive BC is used in Chapter 5.

and 7, we describe a hierarchical modification to the off-policy TD methods described in Section 2.3.2.

2.4 Imitation Learning

The goal of imitation learning (IL), sometimes referred to as learning from demonstrations, is to generate an agent with desired behaviour by leveraging examples of an expert (often a human) demonstrating the behaviour. Compared with reinforcement learning (RL), IL has two main advantages: (i) it can be far more sample efficient (Sun et al., 2017), and (ii) it can remove the need for reward design (see Section 2.3.3). However, IL is particularly subject to distribution shift (see Section 2.1.4): offline supervised learning can lead to *cascading errors*, while online learning can provide insufficient information to avoid suboptimal local maximums. In this section, we describe the two primary approaches to IL, and how they can each suffer from distribution shift: offline, supervised learning, known as *behavioural cloning* when applied to IL (used in Chapters 3 to 5), and online reinforcement learning with a reward learned from expert data, known as *inverse reinforcement learning* (used in Chapters 6 and 7).

2.4.1 Behavioural Cloning

Behavioural Cloning (BC) is an approach to IL where we treat our goal of generating an effective agent as a supervised learning problem (see Section 2.1) (Bain and Sammut, 1996; Pomerleau, 1989). Replacing the supervised learning notation from Section 2.1 with MDP notation from Section 2.2, the goal in BC is to learn a policy π given input states (or observations) $s \in \mathcal{S}$ and output actions $a \in \mathcal{A}$, given a training dataset of N state-action pairs, $\mathcal{D}_E = \{s_n, a_n\}_{n=1}^{N}$.[10] Apart from a shift in notation,

[10] Compared with Section 2.1, we have dropped the vector notation for convenience, but both $\mathcal{S}$ and $\mathcal{A}$ can be multidimensional.

BC is algorithmically identical to supervised learning, and is trained with a loss function

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \ell \left(a, \pi(s; \theta) \right), \tag{2.32}$$

which can be applied to both discrete actions (Section 2.1.1) and continuous actions (Section 2.1.2).

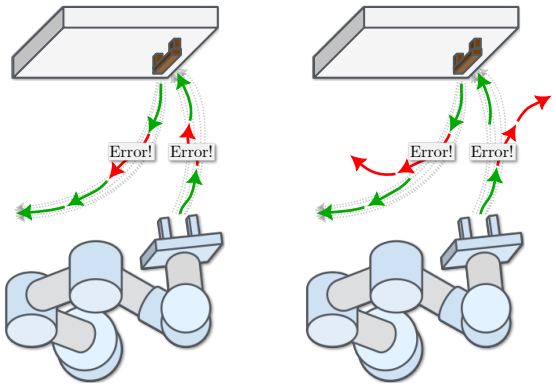
2.4.2 Distribution Shift in Behavioural Cloning

As described in Section 2.1.4, supervised learning approaches (including behavioural cloning) are designed to generalize to data beyond the training dataset, but are not expected to do so outside of the *distribution* of the training dataset. If the distribution underlying the test dataset $\mathcal{D}_{\text{test}}$ differs sufficiently from distribution of $\mathcal{D}_E$, performance of π is likely to be poor due to distribution shift.[11] With BC, the problem is magnified: a single state s_{ood} may only be slightly out-of-distribution (OOD), causing π to generate an action $a_{\text{suboptimal}}$ with some small difference from the optimal action $a^* = \pi(s)$. For a standard ood learning problem (i.e., one where this is no sequential decision making), this would be the end of the problem, and hopefully the OOD state s_{ood} and the suboptimal action $a_{\text{suboptimal}}$ are discovered before it causes significant damage. In BC, $a_{\text{suboptimal}}$ is executed in the MDP, generating s' as $s' \sim \mathcal{P}(\cdot \mid s_{\text{ood}}, a_{\text{suboptimal}})$. Because both s_{ood} and $a_{\text{suboptimal}}$ are now slightly OOD compared with $\mathcal{D}_E$, $\mathcal{P}$ may generate an s' that is *even more out-of-distribution* than s_{ood}, and the process repeats, creating cascading errors.

This problem is well-established in literature on IL, and is usually considered to result from covariate shift (due to the change in $p(s)$), but could also qualify as label shift due to the change in $p(a)$. One approach to resolving the problem is to change the initial state distribution ρ_0 to ensure a wider distribution of $\mathcal{S}$ and $\mathcal{A}$ are collected. In Chapter 3, we leverage known structure in a mobile manipulation domain to significantly widen $\mathcal{S}$, and can generate a more robust policy that even partially generalizes to OOD states. In Chapter 4, we identify an inevitable distribution shift resulting from the use of kinesthetic teaching, a popular approach to collecting demonstrations in which an operator directly pushes a robotic arm. The true cause of the shift is a change in embodiment between a demonstrator and an agent, which is also a well-established problem in IL that can exacerbate covariate shift. We resolve the problem by modifying the demonstration collection process to accommodate the embodiment change.

Assuming one has access to an optimal teacher policy $\pi^*(s)$, Ross et al. (2011) showed that you can converge to a far better policy by alternately running π in the environment, collecting the states visited by policy, labelling each with the expert policy, and retraining on the new data. Formally, newly encountered states are stored in a policy dataset $\mathcal{D}_\pi$, each $s \in \mathcal{D}_\pi$ is labelled with the optimal action $a^* = \pi^*(s)$, $\mathcal{D}_\pi$ is appended to $\mathcal{D}_E$ as $\mathcal{D}_E \leftarrow \mathcal{D}_E \cup \mathcal{D}_\pi$, and π is retrained on $\mathcal{D}_E$. The process is known as dataset aggregation (DAgger). Unfortunately, access to π^* is not possible in most scenarios we are interested in, making it impossible to apply DAgger directly to most real-world scenarios. In Chapter 5, we investigate an approach to *interactive* imitation learning (Celemin et al., 2022) inspired

[11] Concrete methods for directly measuring statistical distances are beyond the scope of this section, but "how OOD" a testing dataset or single sample is can be measured, for example, via divergence, uncertainty, or entropy.



(a) Error recovery. **(b)** Distribution shift.

Figure 2.4: Contrasting two behavioural cloning scenarios: on the left, an idealized case where BC keeps the policy in-distribution, allowing for the recovery from errors; and on the right, a more typical outcome in which errors compound due to distributional shift. In the scenario on the right, assuming an error rate of ϵ and a worst-case scenario where an error occurs on the first timestep, the total worst-case error grows as $\mathcal{O}(T^2\epsilon)$, as discussed in Section 2.4.3. DAgger (Ross et al., 2011) recreates the scenario on the left by removing distribution shift, resulting in total expected error of $\mathcal{O}(T\epsilon)$.

by DAgger. Specifically, we predict whether s is OOD and switch control from π to a human expert, allowing the human to guide the agent back to in-distribution states, and retrain on the corrective data.

2.4.3 Formal Guarantees of DAgger

Ross and Bagnell (2010) established that error accumulation resulting from running a policy trained via behavioural cloning (BC), assuming an error rate of ϵ, will result in a worst-case total error of

$$\mathcal{O}(T^2\epsilon), \tag{2.33}$$

where T is the horizon length. In BC, each decision incurs a per-step error of ϵ, so across T timesteps we would expect $T\epsilon$ errors. However, in the worst case, each mistake shifts the learner into regions unseen during training, inflating the *per-timestep* expected cost. Assuming all future timesteps will now also be erroneous, per-timestep cost is inflated by a factor of T, resulting in Eq. (2.33). Fig. 2.4 shows a visualization of this expectation compared to the reality of learning with BC.

Ross et al. (2011) showed that Eq. (2.33) can be reduced to the expected linear growth in T from regular supervised learning,

$$\mathcal{O}(T\epsilon), \tag{2.34}$$

by implementing the DAgger algorithm described in Section 2.4.2. This ensures that the training distribution and the test distribution are matched, eliminating distribution shift (in the long run). Ross et al. (2011) also showed that the worst-case number of DAgger iterations, or trajectories, required to

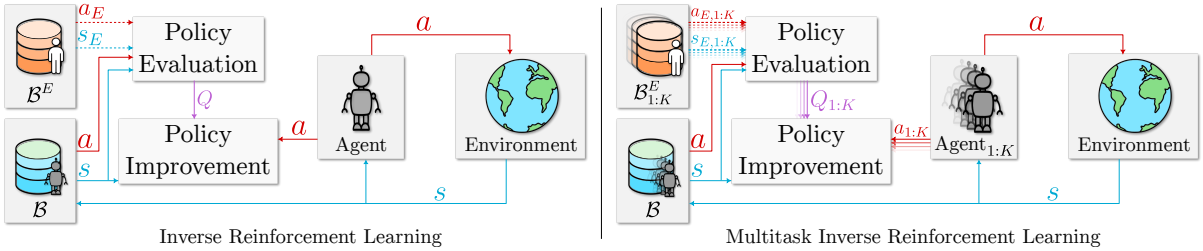


Figure 2.5: Modifications to the agent-environment loop in inverse reinforcement learning (IRL) and multitask IRL. In multitask IRL, we have access to K expert buffers, allowing us to also learn K agents (and corresponding Q-functions) simultaneously, while the agent ultimately selects a single action to take in the environment. In Chapters 6 and 7, we use multitask IRL as the base algorithm.

learn a satisfactory policy can, with certain assumptions, be

$$\mathcal{O}\left(T \log T\right). \tag{2.35}$$

With BC, there is no way to provide a similar guarantee.

DAgger relies on two assumptions: (i) no-regret learning, that is, the learning algorithm performs nearly as well as the best fixed decision in hindsight, as supervised learning often does; and (ii) access to an expert policy that can provide offline labels, which, for many real-world cases, does not exist. Even without access to this expert policy, however, DAgger and its corresponding reduction in error bound can still provide guidelines on how to reduce distribution shift. Each method presented in Chapters 3 to 7 can be thought of as a way of reducing the worst-case error from Eq. (2.33) to a value lower than Eq. (2.33), but still higher than Eq. (2.34). While our methods are more empirically motivated than (Ross and Bagnell, 2010; Ross et al., 2011), we will attempt to provide connections to this theory throughout this thesis so practitioners understand how Chapters 3 to 7 reduce distribution shift compared with Eq. (2.33) and Eq. (2.34).

2.4.4 Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) is another popular approach to IL, where IL is formulated as reinforcement learning, but the reward function R is learned or defined using an expert buffer $\mathcal{B}^E$ of (s, a, s') tuples. $\mathcal{B}^E$ can optionally be appended to the experience buffer $\mathcal{B}$ during learning, potentially further improving learning efficiency to help train π and Q in addition to R.[12] Early approaches to IRL attempted to first learn a linear reward function that would explain the expert data, and subsequently maximize this function with RL (Ng and Russell, 2000; Abbeel and Ng, 2004). This process presents many challenges, including the need to predefine appropriate linear features, and the clear possibility that there are infinitely many reward functions that can generate an optimal policy, although this can be partially mitigated by adding a term for simultaneously maximizing policy entropy (Ziebart et al., 2008).

[12] Appending $\mathcal{B}^E$ to $\mathcal{B}$ is not, by itself, considered IRL. Much work in IRL does not use this technique, and some work in RL uses this technique with a predefined reward function. Approaches in Chapters 6 and 7 perform both IRL, in which no reward function is available, as well as appending $\mathcal{B}^E$ to $\mathcal{B}$.

Newer approaches that scale to learned features with deep neural networks are based on *adversarial imitation learning* (AIL) (itself based on *generative adversarial networks* (GANs) (Goodfellow et al., 2014)), where the policy is directly recovered through a reward function that is learned or defined based on $\mathcal{B}^E$ (Ho and Ermon, 2016; Fu et al., 2018a). AIL methods used in this work frame IRL as an adversarial process with the minimax objective

$$J(\pi, D) \triangleq \min_{\pi} \max_{D} \mathbb{E}_{(s,a)\sim\mathcal{B}^E} \left[\log(D(s,a))\right] + \mathbb{E}_{a\sim\pi, s\sim\mathcal{P}} \left[\log(1 - D(s,a))\right], \tag{2.36}$$

where $D(s,a)$ is a *discriminator*, or a binary classifier (see Section 2.1.1). Intuitively, $J(\pi, D)$ is maximized when D gives high outputs to expert data from $\mathcal{B}^E$ and low outputs to policy data, and $J(\pi, D)$ is minimized when D gives high outputs to policy data and low outputs to expert data. In practice, Eq. (2.36) is maximized by training D as a classifier, and minimized by training π in a reinforcement learning loop with the reward function R set based on the output of D. When training in the off-policy regime, Reddy et al. (2020) introduced a modification where D is simply defined, such as

$$D(s,a) = \begin{cases} 1, & \text{if } (s,a) \in \mathcal{B}^E, \\ 0, & \text{if } (s,a) \in \mathcal{B}. \end{cases} \tag{2.37}$$

2.4.5 Distribution Shift in Inverse Reinforcement Learning

IRL can, ostensibly, resolve the behavioural cloning distribution shift problem described in Section 2.4.2, since the agent autonomously explores until the distribution induced by sampling π and $\mathcal{P}$ matches the distribution induced by sampling $\mathcal{B}^E$. As π inevitably encounters states s_{ood} that are out-of-distribution with respect to $\mathcal{B}^E$, these states are simply appended to $\mathcal{B}$, rated with a lower reward than those from $\mathcal{B}^E$, and through generalized policy iteration (Section 2.3.1), the policy learns the highest-rated action to take when encountering s_{ood}.

In practice, IRL can generate highly suboptimal policies because of poor exploration, resulting in reaching a local maximum policy that receives some reward, but ultimately does not complete the task. Similar to the distribution shift problem in behavioural cloning, because we do not have access to π^*, we cannot necessarily generate optimal actions for s_{ood}. Policies can then learn to output suboptimal actions that result in states that only partially match $\mathcal{B}^E$. This can also be interpreted as a form of the reward hacking described in Section 2.3.3. We further explore this defect of IRL in Chapter 6, and resolve it by adding demonstrations of auxiliary tasks that enforce more thorough exploration. We can do even better by converting the expert data in IRL from dense to sparse (see Section 2.3.3): a dense expert corresponds to a full trajectory, while a sparse expert corresponds to examples of a completed task only. In Chapter 7, we expand the ideas in Chapter 6 to the domain of sparse expert data, further ensuring that the distribution induced by $\mathcal{B}^E$ cannot be exploited by the policy.

Chapter 3

Multiview Manipulation from Demonstrations

In this chapter, we present a task-specific method for handling distribution shift.[1] If, before collecting data, we are aware of potential differences between the training and testing distributions, we can attempt to deliberately generate a wider variety of training data. We show that mobile manipulation policies trained via behavioural cloning are susceptible to distribution shift: specifically, the covariate shift caused by collecting data from a single, fixed viewpoint while executing the policy with multiple similar but different viewpoints. Consider a human moving through a kitchen. Each time they open the cabinet containing their dishes, they are able to open the cabinet, even if they're standing in a position that they may never have stood precisely in before, with a viewpoint they may have never seen before.

We create a simple scheme for randomizing the initial base pose of a mobile manipulator to directly collect a wider range of data under the assumption that we expect a distribution shift to occur. We illustrate the general applicability of the method by learning to complete several challenging multistage and contact-rich tasks, from numerous viewpoints, both in a simulated environment and on a real mobile manipulation platform. Furthermore, we analyze our policies to determine the benefits of learning from multiview data compared to learning with data collected from a fixed perspective. We show that learning from multiview data results in little, if any, penalty to performance for a fixed-view task compared to learning with an equivalent amount of fixed-view data. Finally, we examine the visual features learned by the multiview and fixed-view policies. Our results indicate that multiview policies implicitly learn to identify spatially correlated features.

3.1 Motivation

The use of end-to-end visuomotor policies, in which observations are mapped directly to actions through a learned model, has emerged as an effective alternative to the traditional sense-plan-act

[1] Project website: https://papers.starslab.ca/multiview-manipulation.

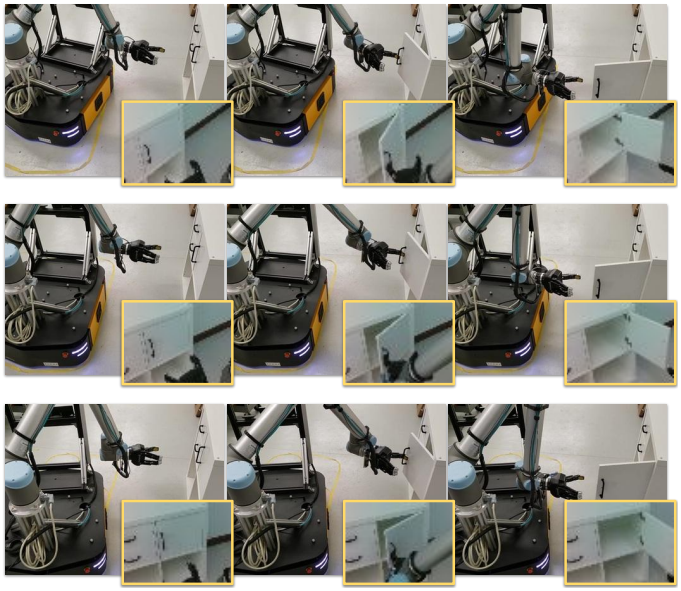


Figure 3.1: Snapshots of three successful trajectories involving different viewpoints for our real-world cabinet-opening task, executed by a single policy. Each trajectory is shown left-to-right. The yellow boxes highlight individual 64×48 RGB input image frames. Our end-to-end policy is able to generalize to the different images and base poses corresponding to each viewpoint.

approach for many robotic domains including autonomous driving (Pomerleau, 1989; Bojarski et al., 2016) and manipulation (Levine et al., 2016; Zhang et al., 2018; Laskey et al., 2017a). Visuomotor policies are particularly appealing for manipulation because programming a robot to complete even relatively basic tasks can pose a major challenge. Most research on learning end-to-end manipulation policies, where the inputs to the policy are easily-acquired camera images and proprioceptive sensor data, has focused on a fixed-base arm and a fixed camera viewpoint. If these policies are naïvely rolled out for a task that requires the camera angle or base position to change slightly, as is often the case for a mobile manipulator, one would not expect the policies to succeed. We seek to learn highly generalizable policies that are not brittle in the face of perturbations to the viewpoint and base position.

In this work, we investigate the application of supervised imitation learning for generating end-to-end *multiview* policies for complex, contact-rich tasks. Specifically, we create datasets containing trajectories with varying base poses, allowing single policies to learn to complete tasks from a variety of viewpoints (see Fig. 3.1). Policies learned in this way can be directly applied to mobile manipulators in conjunction with a separate navigation policy that moves the mobile base to the vicinity of the manipulation task workspace (Iriondo et al., 2019). Our main contributions in this chapter are the answers to the following questions:

1. How does supervised imitation learning perform in a series of challenging contact-rich tasks in the multiview domain?

2. Will a policy trained with multiview data be penalized when performing an equivalent fixed-perspective task, compared to a policy trained with an equal amount of exclusively fixed-perspective data?

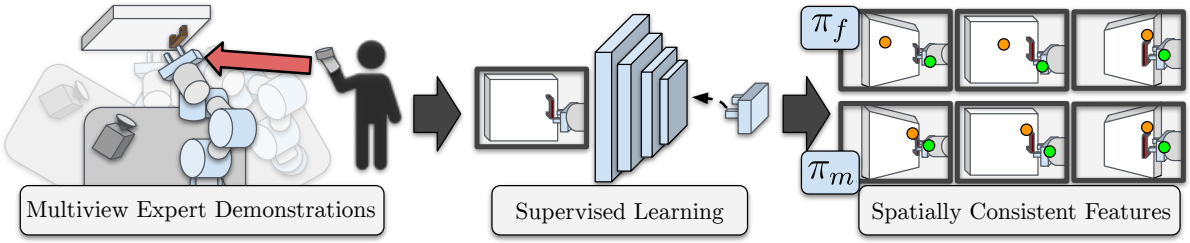


Figure 3.2: Our system for generating multiview policies: after collecting expert demonstrations from several viewpoints, we train a deep neural network policy that generalizes to multiple views. The learned visual features consistently appear on the same parts of objects.

3. How far can fixed-base and multiview policies be pushed beyond their training distributions?

4. Compared with a fixed-view policy, do the features learned by a multiview policy show greater spatial correlation between different views?

Somewhat surprisingly, we demonstrate that multiview policies have comparable performance to fixed-view policies in when testing from the specific fixed-base view, motivating their use in any case where a mobile base is present. We do not explicitly encode view-invariance in the loss function or policy architecture, and instead show that end-to-end multiview policies can be effectively trained implicitly by modifying the dataset.

3.2 Related Work

In this section, we begin by examining existing work on end-to-end policy learning for robotics, followed by learning-based mobile manipulation and other multiview manipulation research. We close with a brief discussion of supervised imitation learning, also known as behavioural cloning.

The use of deep visuomotor policies that map raw observations to actions has exploded in popularity recently, largely owing to representational power and generalization capabilities of deep convolutional neural networks (Levine et al., 2016; Finn et al., 2016; Codevilla et al., 2018). Encoding a policy this way offers the advantage of being able to learn directly from data, given either expert actions or an external reward signal, without requiring accurate world state information and a handcrafted behaviour policy. Such policies have the downside of being limited to operating on data that closely resemble the data they were trained on. In this work, we expand the training dataset to include multiple viewpoints, substantially improving the robustness of the learned policies.

Our approach can be compared to other learning-based methods used for mobile manipulation. The system developed in (Bajracharya et al., 2020) applies several learning techniques, combined with connected motion primitives, to complete several tasks. View-invariance is encoded within separate object recognition and planning modules. Our method, in comparison, uses an end-to-end approach for completing tasks given raw sensor data only.

Several research groups have attempted to learn policies that control both a mobile base and a manipulator simultaneously (Wang et al., 2020; Welschehold et al., 2017; Kindle et al., 2020). In each

of these cases, the authors make assumptions about the availability of lower-level state information (Welschehold et al., 2017; Wang et al., 2020), or confine their systems to perform only relatively simple reaching tasks (Wang et al., 2020; Kindle et al., 2020). While we do not attempt to control the base during task execution, we generate policies that can complete challenging, contact-rich tasks *without* access to privileged state information. The authors of (Laskey et al., 2017b) learn a mobile manipulation bed-making task with imitation learning, but fiduciary markers are required at the base positions to ensure precise localization.

There has been some interest in learning policies that generalize to multiple views even when a fixed-base manipulator is employed. Through view synthesis, simulated views (Amini et al., 2020) or their latent representations (Eslami et al., 2018) can be used for generating higher-quality policies. This approach has two major drawbacks: it requires a potentially prohibitive amount of training data and it operates on the assumption that all parts of images are relevant. Our method learns policies that output control signals given raw images, ensuring that only the parts of the scene relevant for control are extracted.

Other research has investigated the use of multiview representations learned with contrastive losses, determined from either time-aligned sequences from multiple camera views (Sermanet et al., 2018; Dwibedi et al., 2018; Maeda et al., 2020) or via pre-existing object-recognition software (Florence et al., 2018, 2020). Our method does not assume access to any extra information beyond raw RGB-D sensor data[2], but presumably, these representations could be used to improve the learned policies in our work.

In (Sadeghi et al., 2018), the authors applied domain randomization (Tobin et al., 2017) to learn policies that are able to complete a real-world multiview reaching task. The final policy is able to generalize to inputs from novel viewpoints. In contrast, our tasks require significantly higher dexterity than reaching alone.

Behavioural cloning (BC) is the common name given to imitation learning treated as supervised learning (Bain and Sammut, 1996; Pomerleau, 1989): after collecting an expert dataset, a policy is trained to regress to expert actions given the corresponding observations. As discussed in Section 2.4.2, a core assumption of supervised learning is that the training and test data are independently and identically distributed (IID). In BC, this translates to assuming that the policy dataset, generated by running the policy, is drawn from the same distribution as as the expert dataset. Unfortunately, in general this assumption is violated (Ross et al., 2011), but the problem can be mitigated by manually increasing the coverage of the expert dataset (Pomerleau, 1989; Zhang et al., 2018; Laskey et al., 2017a) or by employing an intervention-based strategy (Ablett et al., 2020). We investigate the effects of including and excluding multiple views and base poses in the expert dataset for both multiview and fixed-base tasks.

[2]We use depth information because it is easily acquired along with images using off-the-shelf RGB-D sensors.

3.3 Problem Formulation

We formulate our problem as a Markov Decision Process (MDP). Our goal is to learn a deterministic policy $\pi_\theta : O \rightarrow A$, parameterized by θ, for environment observations $o \in O$ and actions $a \in A$. Instead of maximizing a reward, in imitation learning, we attempt to match a learned policy π_θ to an expert policy π_E. In our case, we do not assume we have direct access to π_E and instead only have samples of human-generated demonstrations $\mathcal{D}_E := \{\tau_1, \ldots, \tau_n, \ldots, \tau_N\}, \tau_n := \{(o_0, a_0), \ldots, (o_t, a_t), \ldots, (o_{T-1}, a_{T-1})\}$, where T is the task horizon length. The initial observation o_0 is sampled from a pre-defined distribution $p(o_0)$.

We train our policies using behavioural cloning. The policy π_θ can be trained by minimizing the mean squared error,

$$\min_\theta \sum_{(o,a)\in\mathcal{D}_E} \left(\pi_\theta(o) - a\right)^2. \tag{3.1}$$

In our work, each individual task or environment $\mathcal{T}$ is a separate MDP that can be considered to be either *multiview* ($\mathcal{T}_m$) or *fixed-view* ($\mathcal{T}_f$), with π_m denoting a policy trained with data from $\mathcal{T}_m$ (and π_f for $\mathcal{T}_f$), where we omit θ for convenience. For $\mathcal{T}_f$, we can define the observation generating process $g_{O,f} : S_\mathcal{M} \times S_i \rightarrow O_{\mathcal{T}_f}$, noting that $o_{\mathcal{T}_f} \in O_{\mathcal{T}_f}$ are generated by an unknown function $g_{O,f}$ of the underlying states of our manipulator $s_\mathcal{M} \in S_\mathcal{M}$ and task-relevant objects $s_i \in S_i$. The initial states of each episode, $s_{\mathcal{M},0}$ and $s_{i,0}$, are uniformly randomized within predefined constraints. In contrast, for the multiview case, we define $g_{O,m} : S_\mathcal{M} \times S_i \times S_b \rightarrow O_{\mathcal{T}_m}$, where we have added the state of the base of our robot $s_b \in S_b$, noting that s_b is randomized only *between* episodes. In our formulation, measurements $o \in O$ are acquired from a sensor attached to the robot base and from the arm itself, so changing $s_{b,0}$ affects $s_{\mathcal{M},0}$, the view of task relevant objects, and the set of actions that are able to 'solve' the task.

3.4 Multiview Training and Shared Information

As stated in Section 3.1, we are interested in the comparison between a fixed-base task $\mathcal{T}_f$ and an equivalent multiview version $\mathcal{T}_m$, as well as policies π_f and π_m trained on observations $O_{\mathcal{T}_f}$ and $O_{\mathcal{T}_m}$. It is important to note that because $S_\mathcal{M}$ and S_i are shared between these environments and $\dim(O_{\mathcal{T}_m}) = \dim(O_{\mathcal{T}_f})$, we can generate actions from π_m or π_f with both $O_{\mathcal{T}_m}$ and $O_{\mathcal{T}_f}$.

3.4.1 Comparing $\mathcal{T}_m$ and $\mathcal{T}_f$

Considering the sizes of the sets of possible states for $\mathcal{T}_m$ and $\mathcal{T}_f$ leads to a well-known challenge in prediction problems, the curse of dimensionality (Bellman, 1957): since $\dim(S_\mathcal{M} \times S_i \times S_b) > \dim(S_\mathcal{M} \times S_i)$, we should require more training examples from $O_{\mathcal{T}_m}$ to learn π_m than from $O_{\mathcal{T}_f}$ to learn π_f (i.e., to achieve the same success rate). Stated differently, more examples are required to adequately populate the observation space $O_{\mathcal{T}_m}$ than to populate $O_{\mathcal{T}_f}$.

A natural conclusion is that, given the same quantity of training data, π_m will perform *worse* than π_f on $\mathcal{T}_f$, for two separate but related reasons: (i) π_m is required to learn a higher-dimensional problem than π_f, and (ii) π_m is provided with less (or possibly no) $o_{\mathcal{T}_f} \in O_{\mathcal{T}_f}$ at training time.

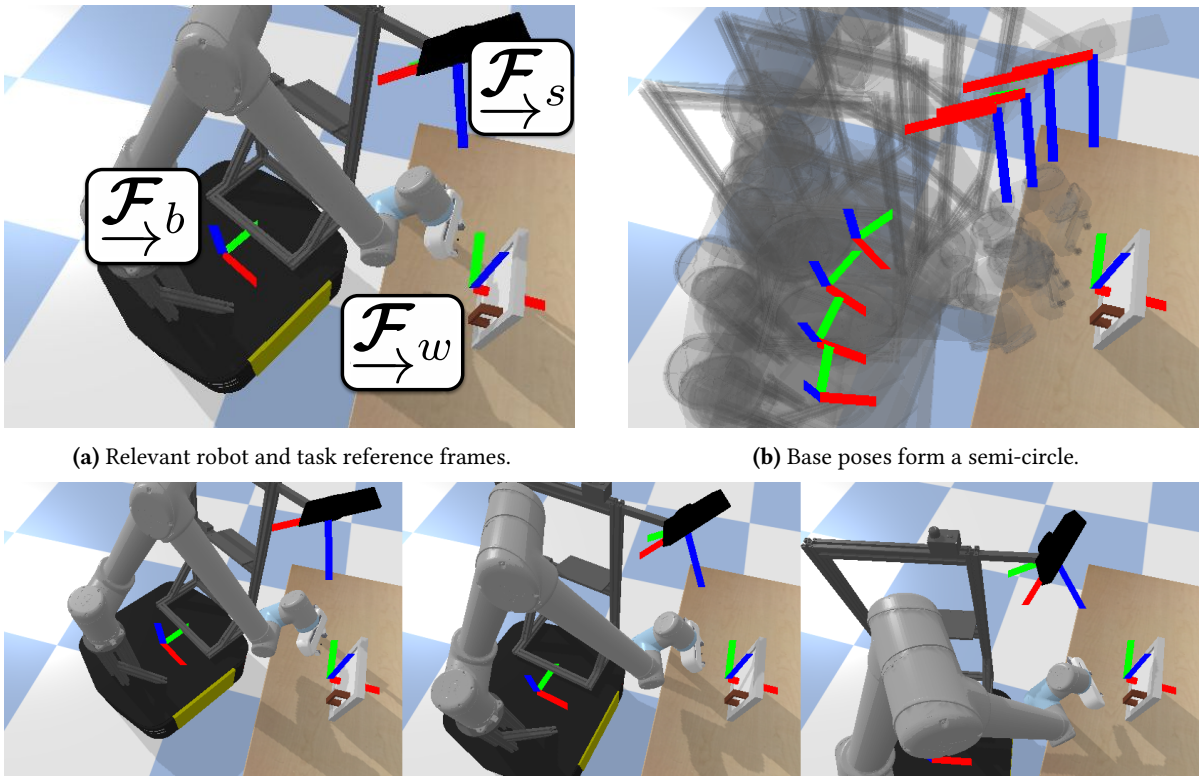
3.4.2 When Multiview Data Helps

Implicit in the above conclusion is the assumption that, for a specific task, the distributions of expert actions $p(a_E \mid s_b = \alpha)$ and $p(a_E \mid s_b = \beta)$ for two different base poses, $\alpha, \beta \in S_b$, are independent. However, this is not true for our problem, or for many other supervised learning tasks. The distributions of observations (and actions) for poses that are 'nearby' in the state space, S_b, must have nonzero mutual information, as well as smooth state, observation, and action spaces. If this were not the case, multiview policies would be unable to generalize to new poses. We experimentally show that our policies are able to generalize to new poses (in Section 3.7.1) and even to out-of-distribution data (in Section 3.7.2). We consider the task-dependent mutual information $I(A_{E,\alpha}; A_{E,\beta})$ between $A_{E,\alpha} \sim p(a_E \mid s_b = \alpha)$ and $A_{E,\beta} \sim p(a_E \mid s_b = \beta)$ below.

For tasks where $I(A_{E,\alpha}; A_{E,\beta})$ is large in general, π_m will provide little benefit over π_f in $\mathcal{T}_m$. However, when $I(A_{E,\alpha}; A_{E,\beta})$ is small in general, π_m may be prohibitively costly to learn and suffer compared with π_f in $\mathcal{T}_f$. This issue arises due to the increased number of expert demonstrations needed to cover the space of $O_{\mathcal{T}_m}$. For this reason, we expect π_m will provide the most benefit, compared to π_f learned from an equivalent amount of data, when $I(A_{E,\alpha}; A_{E,\beta})$ falls somewhere in the middle—each base pose generates similar observations and requires a similar, but not identical, trajectory of actions to allow successful completion of the task (see Fig. 3.1). As an example, consider a lifting task, where the robot has to lift an object sitting on a table. If, when varying s_b, the object poses in the set $S_{i,0}$ remain the same relative to both the imaging sensor and the robot base, a multiview policy would provide little benefit. Conversely, consider a door opening task, where the robot has to open a cabinet door: the initial pose of the cabinet $s_{i,0}$ does not change relative to the world. Therefore, if s_b is changed, the initial pose of the cabinet, relative to the imaging sensor and the robot base, will necessarily change, and a fixed-view policy will likely fail.

We can rephrase our first two experimental questions (see Section 3.1): provided that a task has an upper bound on the range of base poses to consider, is there sufficient mutual information between trajectories at different (but 'nearby') poses to enable a policy π_m to be learned that not only performs adequately in $\mathcal{T}_m$, but performs comparably to π_f in $\mathcal{T}_f$, given the same amount of training data? That is, if we reduce the sampling density (of expert demonstrations), is the mutual information between those demonstrations sufficient to ensure that π_m performs well in $\mathcal{T}_m$ *and* similarly to π_f in $\mathcal{T}_f$? We explore this question in Section 3.7.1.

Finally, despite nonzero mutual information, it remains true that π_m must learn in the larger space of $O_{\mathcal{T}_m}$, compared to π_f and $O_{\mathcal{T}_f}$. If π_m relies on mutual information, we would expect that many of the visual features learned by π_m would consistently refer to the same parts of the scene, regardless of viewpoint, a possibility which we examine in Section 3.7.3.



(a) Relevant robot and task reference frames.

(b) Base poses form a semi-circle.

(c) Base poses where b_ϕ is set to $b_{\phi,\text{min}}$, $0.5(b_{\phi,\text{min}} + b_{\phi,\text{max}})$, and $b_{\phi,\text{max}}$.

Figure 3.3: Base poses ($\underrightarrow{\mathcal{F}}_b$) are randomized along a semi-circle to ensure that the z-axis of the sensor frame ($\underrightarrow{\mathcal{F}}_s$) always points towards the world frame ($\underrightarrow{\mathcal{F}}_w$), defined to be the center of the task space.

3.5 Methodology

As previously noted, we assume that we have access to an "approach" policy that is capable of moving the mobile base to a pose where the task-relevant objects are (i) in view of the base-mounted sensor and (ii) within the reachable workspace of the manipulator.

We use an automated process to generate randomized base poses for each new training episode. We require (i) a rough estimate of the transform from the robot base frame to the sensor (often provided, and easily acquired through off-the-shelf calibration), (ii) a rough estimate of the robot base pose in the fixed world reference frame $\underrightarrow{\mathcal{F}}_w$ (the estimate from wheel odometry is adequate), (iii) a pre-selected centre point in $\underrightarrow{\mathcal{F}}_w$, and (iv) the desired distance (again, approximate) between the camera frame origin and the centre point in $\underrightarrow{\mathcal{F}}_w$. We desire poses of the mobile base where the main optical axis of the camera sensor, at $\underrightarrow{\mathcal{F}}_s$, always very nearly intersects with the selected centre point in $\underrightarrow{\mathcal{F}}_w$. Each pose in the feasible set lies on a circle, as shown in Fig. 3.3. Since the base poses are in $\text{SE}(2)$, they can be defined by b_ϕ, b_x and b_y: new poses are generated by uniformly randomly sampling $b_\phi \sim U(b_{\phi,\text{min}}, b_{\phi,\text{max}})$ (where $b_{\phi,\text{min}}$ and $b_{\phi,\text{max}}$ are set to ensure that there are no collisions with the environment) and computing the appropriate corresponding b_x and b_y using the constraints outlined above. After sampling and solving for the full base pose, we add a small amount of uniform

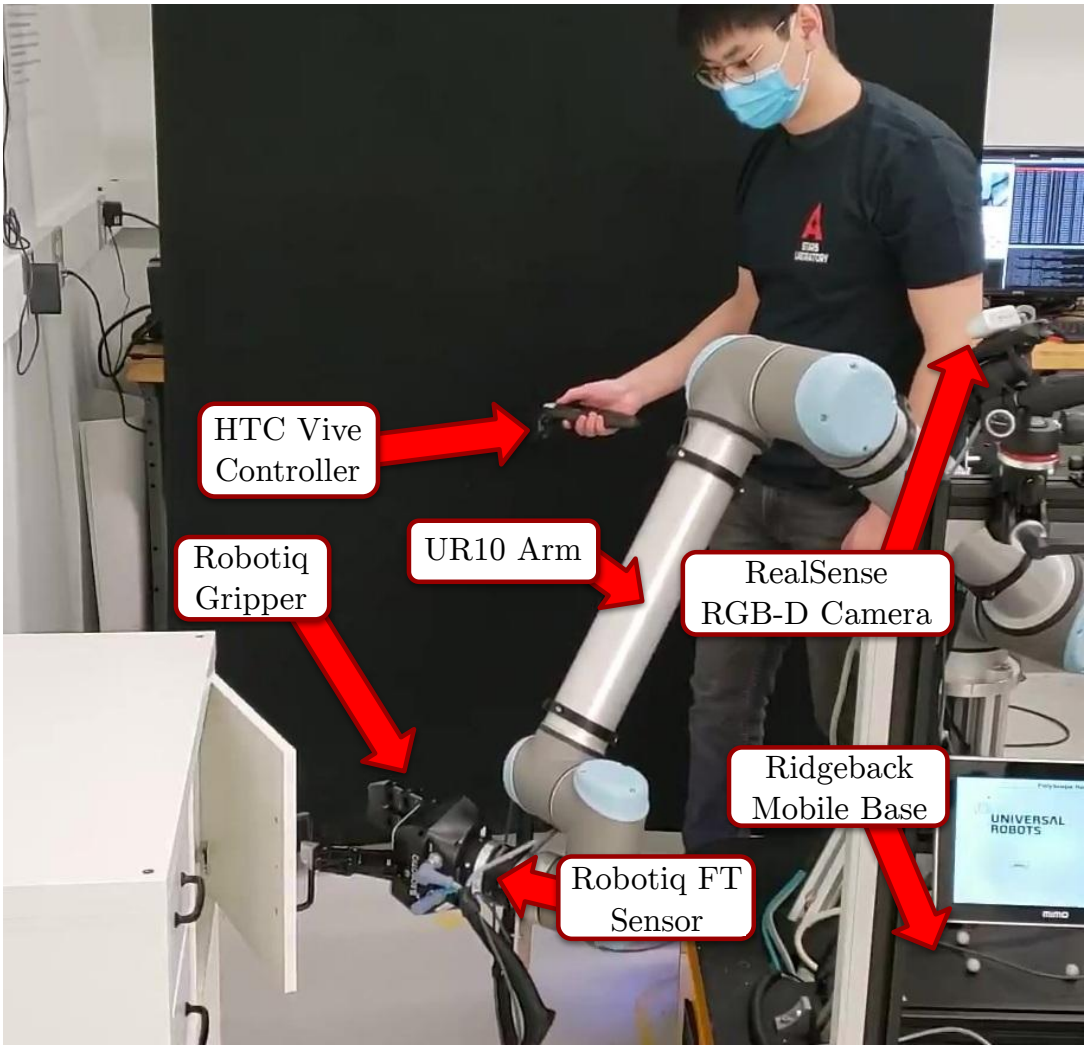


Figure 3.4: Our experimental setup in the real world. Pictured is our mobile manipulation platform as described in Section 3.6.1, as well as a human expert in the process of collecting a demonstration for our `DoorReal` task. See attached video for example demonstrations.

random noise to the pose values (ensuring that task-relevant objects remain in view), with the aim of increasing the robustness of our learned policy. Importantly, our learned policies do not require direct knowledge of the base-to-camera transform or the workspace-to-base transform—this information is used only during the autonomous view generation process.

Our method also requires the collection of a dataset $\mathcal{D}_E$ of expert trajectories of observation-action pairs (o, a) acquired through teleoperation. We only require that the demonstrations are collected without an operator in view of the imaging sensor, though we note that this constraint exists for all visuomotor imitation learning methods.

Table 3.1: Environments considered in this work. Demo time is the cumulative real time of the 200 demonstrations in the multiview version of each expert dataset and the $p(o_0)$ params are the initial conditions of the environment that are randomized between episodes. The "base b_ϕ range" corresponds to $b_{\phi,\max} - b_{\phi,\min}$, as described in Section 3.5.

Environment	Objective	Demo time	$p(o_0)$ params	$p(o_0)$ (ranges)	Actions
`LiftSim`	Reach block and lift above 7.5cm	18m23s	base pose, block pose	base b_ϕ range: $45°$, block center in 25cm×25cm box	Trans vel, grip
`StackSim`	Stack blue block on green block	25m05s	base pose, blue block pose, green block pose	base b_ϕ range: $45°$, block centers in 15cm×15cm box	6-DOF vel, grip
`PickAnd InsertSim`	Grasp cylinder and insert in hole (<1mm tol.)	14m23s	base pose, cylinder pose	base b_ϕ range: $45°$, cylinder center in 2.5cm×2.5cm box	6-DOF vel, grip
`DoorSim`	Grasp door handle, open >90°	25m17s	base pose, initial gripper pose	base b_ϕ range: $45°$, gripper: in 12cm×5cm×5cm box	6-DOF vel, grip
`PickAnd InsertReal`	Grasp cylinder and insert in hole (<1mm tol.)	28m30s	base pose, cylinder pose	base b_ϕ range: $35°$, cylinder center in 2.5cm×2.5cm box	6-DOF vel, grip
`DoorReal`	Hook door handle, open >90°	30m56s	base pose, initial gripper pose	base b_ϕ range: $35°$, gripper: in 12cm×5cm×5cm box	6-DOF vel
`DrawerReal`	Hook drawer handle, open within 2cm of max	32m23s	base pose, initial gripper pose	base b_ϕ range: $35°$, gripper: in 12cm×5cm×5cm box	6-DOF vel

3.6 Experimental Setup

In this section, we describe our experimental design, including the hardware used, the parameters of our tasks, and how we train our policies.

3.6.1 Hardware

We carry out experiments on both simulated and real versions of our mobile manipulation platform, shown in Fig. 3.4. Our real platform has a Robotiq three-finger gripper, while our simulated platform in PyBullet (Coumans and Bai, 2019) uses either a PR2 gripper or a Franka Emika Panda gripper due to the difficulty of simulating the three-finger gripper.

On our physical platform, we employ a simple compliant controller that uses a Robotiq FT-300 force-torque sensor for feedback, allowing our policies to operate safely in our contact-rich tasks. Our robot is controlled using off-the-shelf ROS packages at the lower level and our own inverse kinematics library.

On both the real and the simulated platforms, we use, in addition to other data as detailed in Section 3.6.2, RGB images and depth images. On the real platform, images are captured from an Intel RealSense D435. The sensor is firmly mounted to the mobile base (see Fig. 3.4), ensuring that, when the base moves, the sensor moves with it (see Fig. 3.3).

We collect human demonstrations with a single HTC Vive hand controller (see Fig. 3.4) and custom-designed software. For visual feedback, we found that having the user observe the real robot during collection was sufficient. The force-torque sensor measurements provide proportional haptic feedback

to the demonstrator through the vibration motor in the controller—the vibration amplitude increases with the magnitude of the force and/or torque.

3.6.2 Environments

For a summary of our environments/tasks, see Table 3.1. Representative images from successful trajectories in each environment are shown in Fig. 3.5. All of the tasks' input data include, in addition to RGB and depth images, the current pose of the end-effector in the frame of the robot, provided by forward kinematics and represented as a seven-tuple with Cartesian coordinates for position and a unit quaternion for orientation. As well, each environment that requires actuating the gripper includes the current and previous two positions of each gripper finger. Finally, the real environments also include force-torque sensor data.

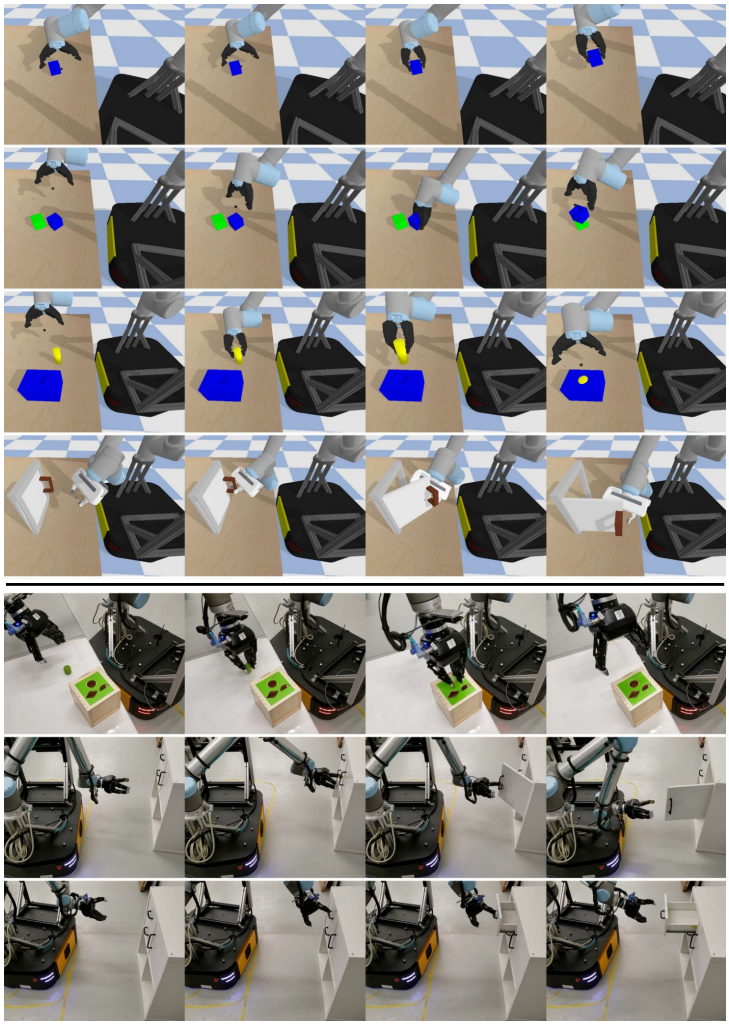


Figure 3.5: Successful trajectories for tasks studied in this work. From top to bottom: `LiftSim`, `StackSim`, `PickAndInsertSim`, `DoorSim`, `PickAndInsertReal`, `DoorReal`, and `DrawerReal`. See attached video for full examples.

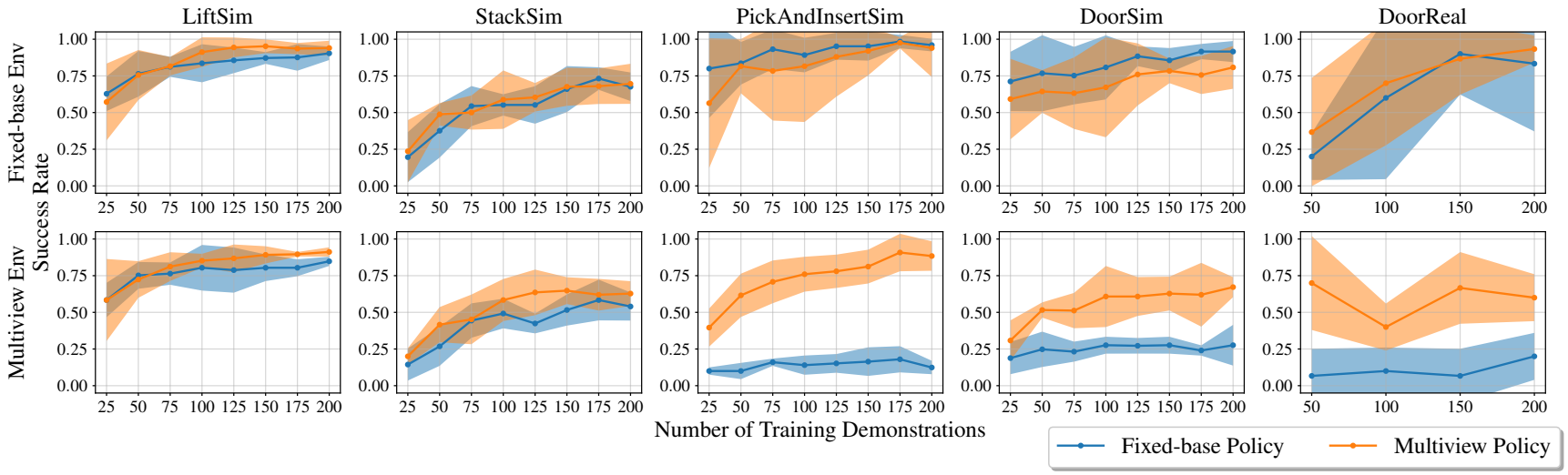


Figure 3.6: Performance of our policies for environments in which we compared fixed-base with multiview policies in fixed-base (top) and multiview (bottom) environments. The shaded region shows the two-sigma bounds across five policy seeds in simulation, and three in the `DoorReal` environment. The multiview policies, as expected, outperform fixed-base policies in multiview settings, often substantially so, with either no or only minor detriment compared with a fixed-base policy in a fixed-base environment.

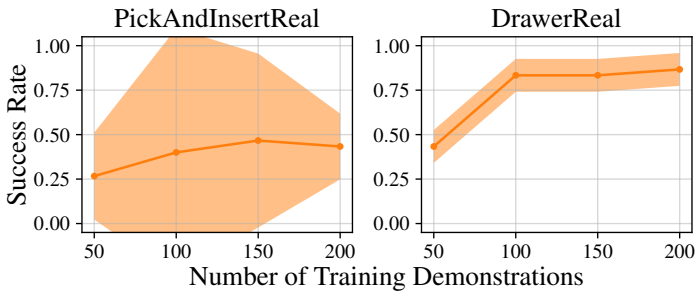


Figure 3.7: Performance results for the other real environments in which we only tested multiview policies. The shaded region shows the two-sigma bounds across three policy seeds.

3.6.3 Policy Architecture and Training

Our policy networks are inspired by (Zhang et al., 2018). Specifically, we use a mutli-layer convolutional neural network (CNN) to process the RGB and depth images, take the spatial soft-argmax (Levine et al., 2016) of the final CNN layer, and concatenate these points with other numerical state information before pushing them through a set of two fully-connected layers. Our CNN layers are the same as in (Zhang et al., 2018), but we use 512 neurons in each of our fully-connected, hidden layers. Crucially, all inputs are available from raw sensor data and our policy does not have access to any privileged state information, including object poses or the relative base pose. We reduce the resolution of our RGB and depth images to $64\times48\times3$ and 64×48, respectively, and initialize the weights of the RGB layer with weights from ResNet (He et al., 2016). Empirically, we found greatly reduced variance between the performance of differently seeded policies by training our policies as ensembles (Breiman, 1996), so each policy is a five-member ensemble, with the final output being the mean output. Each member policy is trained with the same data shuffled differently, initialized with different random orthogonal weights (apart from the pretrained weights).

We train the policies using Tensorflow (Abadi et al., 2015) and the Adam optimizer (Kingma and Ba, 2015) with early stopping, ending training when the validation error on a 20% holdout set has not improved for 30 epochs. We use a learning rate of 0.001, a mini-batch size of 64, and a maximum of 200 epochs. Our loss function is the mean squared error (Eq. (3.1)) between the expert and policy action. We did not do a hyperparameter search because our results given these parameters sufficiently answered the questions posed in Section 3.1, but presumably, a search could have marginally improved our success rates.

The `LiftSim` environment policies are trained using demonstrations generated from a policy learned with Soft-Actor Critic (Haarnoja et al., 2018) to encourage repeatable experiments. Given the high cost of generating autonomous policies for the other tasks, all other tasks use exclusively human-generated data.

3.7 Experiments

Our goal is to investigate the performance of multiview policies relative to fixed-view policies on a series of contact-rich manipulation tasks. To do so, we compare the performance of π_m and π_f in both $\mathcal{T}_m$ and $\mathcal{T}_f$ and on out-of-distribution data. To attempt to explain performance gaps, we additionally examine the spatial consistency of the visual features learned by π_m and π_f.

For each task, we collected 200 demonstrations and trained five policies, with different seeds and at multiple demonstration quantities, and finally ran a series of test episodes with held-out initial conditions to evaluate the success rate of each policy. We trained our policies with increments of 25 demonstrations per policy in simulation, and 50 per policy on the real robot. We tested our policies with 50 evaluation episodes per policy in simulation, and 10 evaluation episodes per policy on the real robot.

3.7.1 Multiview Versus Fixed-base

For `LiftSim`, `StackSim`, `PickAndInsertSim`, `DoorSim`, and `DoorReal`, we collected both multiview and fixed-base data on each task and compared performance under four conditions: π_m in $\mathcal{T}_m$, π_m in $\mathcal{T}_f$, π_f in $\mathcal{T}_m$, and π_f in $\mathcal{T}_f$ (see Fig. 3.6). Notably, as we predicted in Section 3.4.2, in the `Lift` and `Stack` environments, the multiview policy only provides a marginal benefit over a fixed-base policy in a multiview environment. For these two environments, π_m does not perform any worse than π_f in $\mathcal{T}_f$, indicating that a multiview policy can improve performance, and does not appear to cause any detriment.

The benefits of a multiview policy are much clearer in the `PickAndInsertSim`, `DoorSim`, and `DoorReal` environments, where the fixed-base policy fails often in the multiview case, while the multiview policy, as expected, increases in performance with the number of demonstrations. Compared with a fixed-base policy, the multiview policy does lose a small amount of performance in the $\mathcal{T}_f$ `DoorSim` task—an appealing direction for furture work is to consider whether a small amount of data from $\mathcal{T}_f$ provided to π_m, after pretraining on $\mathcal{T}_m$, could close this gap. Notably, we do not see the same effect in the real world version of the Door task. We suspect that in the real world, it is quite difficult to ensure that the base is in the *exact* same pose it was in during data collection. Of course, this small deviation is not an issue for the multiview policy and further motivates its use over a fixed-view policy.

The performance results of multiview policies for `PickAndInsertReal` and `DrawerReal` are shown in Fig. 3.7. As is the case for `PickAndInsertSim`, `DoorSim`, and `DoorReal`, we would reasonably expect that a fixed-base policy would not be able to complete these tasks successfully given differing views. It is worth noting that the performance variation between differently-seeded policies for `PickAndInsertReal` is relatively high. We suspect this is due to the difficulty of learning this task in a purely supervised framework. Empirically, many of the failures in this environment were "near-misses."

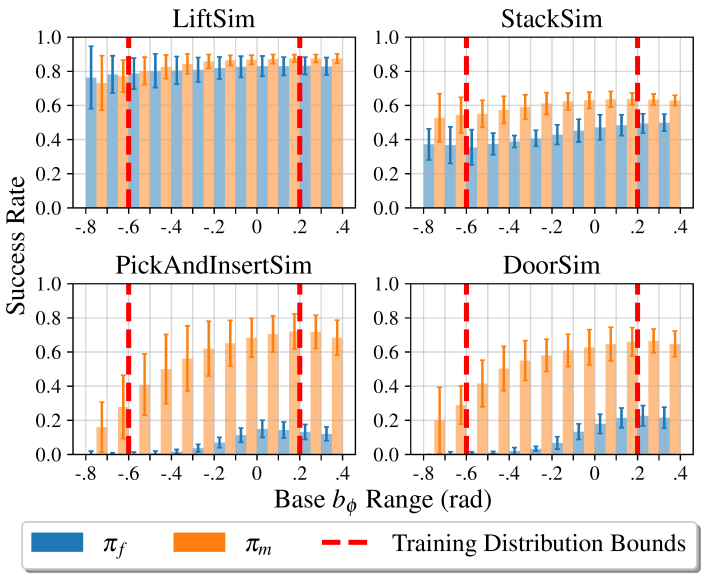
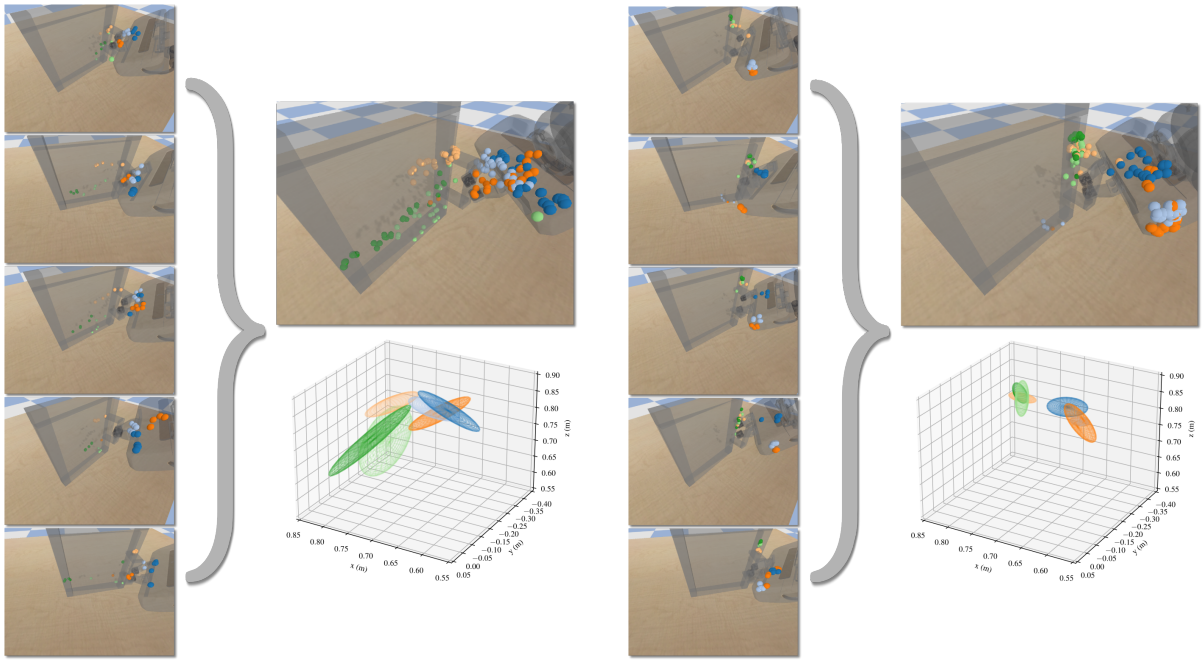


Figure 3.8: Results of testing multiview policies (π_m) and fixed-base policies (π_f), each trained with 200 demonstrations, in multiview environments at a range of angles. The whiskers show the two-sigma bounds across five policy seeds. Here, π_m outperforms π_f, and also shows some ability to perform adequately outside of the training distribution bounds.

3.7.2 Out-of-Distribution (OOD) Experiments

To investigate performance on OOD data, we compared the success rate of multiview policies with fixed-view policies given specific base angles, b_ϕ, as described in Section 3.5. Our multiview policies were trained in our simulated environments with $b_\phi \sim U[-0.6, 0.2]$ (radians), and our fixed-view policies were trained with $b_\phi = 0$. We tested both policies with 12 sets of initial conditions for b_ϕ: $b_{\phi,\text{range}} = \{[-0.8, -0.7], [-0.7, -0.6], \ldots, [0.3, 0.4]\}$. We drew 50 random values from each $b_{\phi,\text{range}}$, and recorded the success rate on these episodes for five seeds of multiview policies and fixed-base policies, each trained with 200 expert demonstrations. The results are shown in Fig. 3.8.

As predicted, both types of policies tend to perform roughly equally in each range of angles in `LiftSim`, and, as shown in Section 3.7.1, the `StackSim` multiview policy tends to perform better in general. For `PickAndInsertSim` and `DoorSim`, a clearer picture emerges to explain the performance difference shown in Fig. 3.6: the fixed-base policies were trained exclusively at $b_\phi = 0$ with no variation in b_x or b_y, so π_f performance, with even small variations in b_ϕ, falls dramatically compared with the performance of π_f on $\mathcal{T}_f$. The performance of π_f continues to deteriorate as $|b_\phi|$ increases, while the multiview policies do well throughout the training distribution, with a noticeable negative skew in performance towards the negative angles. This reduced performance may occur because our camera is already at an angle to the left of the scene (see Fig. 3.3), so moving it further to the left makes the task particularly challenging. The multiview policies show some degree of ability to generalize beyond their training distribution, indicating that the multiview policies learn about the geometric relationship between the arm and the objects in the scene.



(a) Spatially inconsistent SSAM points from π_f.

(b) More spatially consistent SSAM points from π_m.

Figure 3.9: A comparison of the three SSAM points with highest activation on the gripper and the door, reprojected from five different time steps for five different episodes with different viewpoints. The five-image columns on the left side of each figure show the locations of the six features for each different viewpoint, with each SSAM output corresponding to a different colour. The images on the right show all SSAM points reprojected to a single view. The ellipsoids show the two-sigma bounds of the covariance of all of the object-consistent SSAM positions. The features from π_f display far less spatial consistency than the features from π_m.

3.7.3 Learned Feature Analysis

Expanding on our analysis in Section 3.4.2, we compare the visual features learned by policies π_f and π_m. The vision portion of our network terminates with a set of 32 spatial soft-argmax (SSAM) outputs per ensemble member, generated from each of the last convolutional filters, which can be interpreted as points in image space (we refer the reader to (Levine et al., 2016) for a more detailed explanation of SSAM). In this section, we refer to SSAM outputs/points interchangeably as features, but unlike traditional features in computer vision (i.e., those used for feature matching), they do not specifically encode a descriptor that can consistently identify exactly the same parts of different images.

The use of SSAM outputs allows us to interpret where the network directs its attention. We can therefore observe whether individual SSAM outputs are spatially consistent, which would imply that a view-independent geometric representation has been learned and potentially (partially) explain the generalization capability. As noted in Section 3.4.2, the reuse of information between views would lead to spatially consistent (correlated) features.

For this analysis, we use five random episodes (i.e., with five different views) of each of π_m and π_f acting in $\mathcal{T}_m$ in our `DoorSim` environment. In each episode, starting from one time step before the policy initially closed the gripper (attempting to grasp the door handle—arguably the most challenging

part of the task), we record all of the SSAM points and activation magnitudes from each policy for five time steps. These SSAM points are then projected into Cartesian space using the known camera intrinsic parameters, depth image data, and the world gripper and door poses. Given five episodes and five time steps per episode, each SSAM output produces 25 3D feature points in the world frame. We sort the SSAM points by their activation magnitudes, and take the three SSAM points with the highest average activation magnitudes that also show up at least 20 out of 25 times on either the door or the gripper (as determined using ground-truth information from PyBullet), yielding six representative SSAM outputs in total (three for the door, three for the gripper) for each of π_m and π_f. Each of these six SSAM features has between 20 and 25 positions on either the gripper or the door. We plot the reprojected locations of these features in Fig. 3.9.

The features learned by π_m clearly show a smaller degree of spread, and higher spatial correlation, than the features from π_f. The spatial correlation of the SSAM layer activations indicates that a degree of view-invariance has been learned without the need to explicitly train using a view-invariance loss or architecture—the policy has learned a visual representation of the task-relevant objects (including the arm) in terms of features that are robust to viewpoint changes. Our interpretation that π_m has learned a degree of true view-invariance is also supported by our results in Section 3.7.2: π_m generalizes to viewpoint shifts *beyond* its training distribution.

3.8 Limitations

In this section, we discuss some limitations of our work. While we experimentally showed that our policies generalize to multiple views, we did not actually test them in tandem with a separate mobile base policy, so it is possible that our choice for generating multiple views may not be representative of a true mobile base policy. Furthermore, we train policies end-to-end on raw image and depth data, meaning that slight changes to the background, or to lighting, could potentially have highly detrimental effects on our policies. Finally, our policies still fail in many cases, and do not have any means for addressing these failures or recovering.

3.9 Summary

In this chapter, we learned end-to-end policies for challenging, contact-rich tasks involving multiple views, effectively resolving the distribution shift that occurs when a mobile manipulation base position changes between task attempts. We demonstrated the benefits of multiview policies through extensive experiments on a mobile manipulation platform in both simulation and in the real world. Specifically, given the same amount of training data, a multiview policy can be learned with very little, if any, detriment to performance compared with a fixed-base policy and a corresponding fixed-base task.

Multiview data and the corresponding policies leverage knowledge of the task to more broadly cover the expert distribution, thereby reducing the likelihood of encountering out-of-distribution states. Referring to the theoretical gap of the worst-case error between BC and DAgger from Section 2.4.3,

this increased spread provides a partial approximation to the on-policy learning enabled by DAgger. This indicates that, while we cannot expect to achieve the worst-case error of $\mathcal{O}(T\epsilon)$ provided by DAgger, we can do substantially better than regular BC with fixed-base data, with a worst-case error rate of $\mathcal{O}(T^2\epsilon)$. As we see in our results, the longer horizon tasks, such as opening a door or picking and inserting a peg, tend to have lower success rates overall, indicating that we likely do have a superlinear relationship between cost and T.

Since multiview policies are considerably more flexible than their fixed-based counterparts, we assert that multiview data is always desirable. Possible directions for future work include further investigation into methods for reducing the data required to learn effective policies through the use of traditional and learning-based view synthesis techniques or multiview representation learning. Training a policy with multiview data can be interpreted as an attempt to generate a policy that is agnostic to the view of task objects. Ideally, this would be done at the level of objects, meaning that a view-agnostic *representation* of an object could be shared between tasks, and potentially even between similar objects (e.g. (Florence et al., 2018; Zeng et al., 2022)). Although these representations can help with visual recognition and state estimation, transforming robotic trajectories based on new object states is still an open problem, although newer work has shown some progress (Johns, 2021; Mandlekar et al., 2023).

Chapter 4

Force-Matched Demonstrations

In this chapter, we present another approach to mitigate distribution shift in behavioural cloning by directly modifying the training dataset.[1] Instead of simply adding more data, we identify that the use of raw kinesthetic teaching for demonstrations results in a change of the applied force between training and testing, and present an algorithm to resolve the problem during data collection. We further identify that the complex contact involved in many tasks (e.g. roll, slip, shear forces) may benefit from controllable multimodal visuotactile sensing, the control of which we integrate into our modified approach to data collection, simplifying the application of the sensor. Compared with Chapter 3, this approach can be seen as also immediately identifying a potential shift between the training and testing distribution, but instead of increasing the variety of data collected, we directly modify individual demonstration trajectories themselves to better match the testing distribution.

We introduce two algorithmic contributions, *tactile force matching* and *learned mode switching*, as complimentary methods for improving IL via kinesthetic teaching. Tactile force matching enhances kinesthetic teaching by reading approximate forces during the demonstration with a visuotactile sensor and generating an adapted robot trajectory that recreates the recorded forces, reducing the detrimental effects of the distribution shift from training to testing data. Learned mode switching uses IL to couple *visual* and *tactile* sensor modes with the learned motion policy, simplifying the transition from reaching to contacting. We perform robotic manipulation experiments on four door opening tasks with a variety of observation and algorithm configurations to study the utility of our proposed improvements and multimodal visuotactile sensing. Our results show that the inclusion of force matching raises average policy success rates by 62.5%, visuotactile mode switching by 30.3%, and visuotactile data as a policy input by 42.5%, emphasizing the value of see-through tactile sensing for IL, both for data collection to allow force matching, and for policy execution to allow accurate task feedback.

[1] Project website: https://papers.starslab.ca/sts-il.

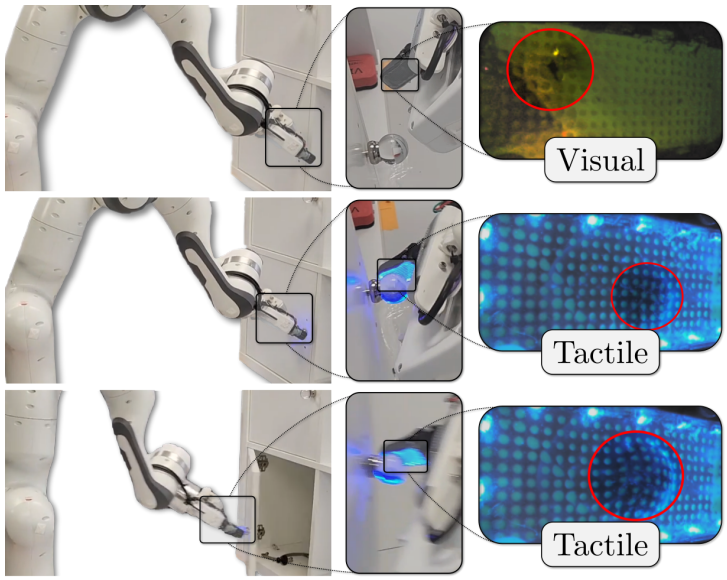


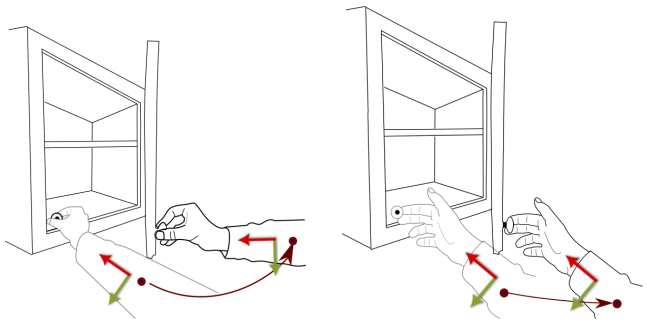
Figure 4.1: Our STS sensor before and during contact (right column) with a cabinet knob (middle column) during a door opening task (left column). In *visual* mode, the camera sees through the gel membrane, allowing the knob to be found, while *tactile* mode provides contact-based feedback, via gel deformation and resultant dot displacement, upon initial contact and during opening. Red circles highlight the knob in sensor view.

4.1 Motivation

The conventional approach to manipulating articulated objects such as doors and drawers with robots relies on a firm, stable grasp of the handle followed by a large arm motion to complete the opening/closing task. In contrast, humans are capable of opening and closing doors with minimal arm motions, by relaxing their grasp on the handle and allowing for relative motion between their fingers and the handle (see Section 4.1). In this chapter, we aim to learn robot policies for door opening that are more in line with human manipulation, by leveraging high-resolution visual and tactile feedback to control the contact interactions between the robot end-effector and the handle.

Optical tactile sensors (Chi et al., 2018) combine a gel-based material with an internal camera to yield rich tactile information (Yuan et al., 2017) and are able to provide the feedback needed for dexterous manipulation (Padmanabha et al., 2020; Ma et al., 2019). A recently-introduced see-through-your-skin (STS) multimodal optical sensor variant combines visual sensing with tactile sensing by leveraging a transparent membrane and controllable lighting (Hogan et al., 2021, 2022). This sensor enables perception of the full interaction, from approach, through initial contact, to grasping and pulling or pushing.

In this chapter, we investigate how to leverage visuotactile sensing for imitation learning (IL) on a real robotic platform for contact-rich manipulation tasks. We focus on the tasks of opening and closing cabinet doors with challenging handle geometries (e.g., flat and spherical knobs) that are difficult to grasp with a parallel jaw gripper and that require fine motor control and tactile feedback. We complete these tasks with a 7-DOF robotic system that integrates a single robotic finger outfitted with an STS visuotactile sensor (see Fig. 4.1), evaluating on four tasks in total.



(a) Grasp and rotate. **(b) Press and pull.**

Figure 4.2: Various human approaches to opening a cabinet. The "Press" approach on the right requires far less arm rotation, but also generates relative motion between the knob and the hand, motivating the use of high-resolution tactile sensing to replicate.

Human-based expert demonstrations for IL can be generated in a variety of ways, though most methods fall generally into the kinesthetic teaching (in which a person directly moves and pushes the arm to complete a task) or teleoperation (in which a person remotely controls the robot through a secondary apparatus) categories (Billard et al., 2016). While neither method is the definitive choice in all cases, kinesthetic teaching offers two specific major advantages over teleoperation: 1) a degree of haptic feedback is provided to the demonstrator, since they indirectly feel contact between the end-effector and the environment (similar to tactile feedback that humans feel during tool use), and 2) no extra devices beyond the arm itself are required. Teleoperation requires a proxy (i.e., a separate sensor, actuator and system) to provide a substitute for true haptic feedback (Ablett et al., 2021b; Li et al., 2023b), and can be costly or inaccurate. Additionally, prior work has found that kinesthetic teaching is preferred over teleoperation for its ease of use and speed of providing demonstrations (Pervez et al., 2017; Fischer et al., 2016; Akgun and Subramanian, 2011).

Unfortunately, kinesthetic teaching methods typically only measure robot motion, without considering the robot-environment contact force (and torque). To match the force profile[2] of the demonstration, we require a means of measuring robot-environment forces, in addition to a mechanism for reproducing those forces. Our first contribution is a *tactile force matching* method that uses the readings from an STS sensor to modify the poses recorded from kinesthetic teaching. Our method generates a new trajectory that, when used as input to a Cartesian impedance controller, recovers the recorded forces *and* poses to generate a *force-matched replay* (see steps 2, 3 and 4 from Fig. 4.3). Assuming a linear relationship between surface deformation and force, we measure approximate force in x, y, z, and torque τ_z using tracked dot motion. This affords multiple advantages compared with typical approaches to force-torque sensing: (i) standard methods for measuring robot-environment force are corrupted by human-robot force (see Fig. 4.4), (ii) an STS is an order of magnitude less expensive than a similarly-mounted force-torque sensor, and (iii) the STS can be additionally used, in both visual and tactile modes, to provide raw sensor data for learning policies.

[2]We refer to forces in this section, but the method also applies to wrenches. When necessary, we specify individual dimensions of force and torque.

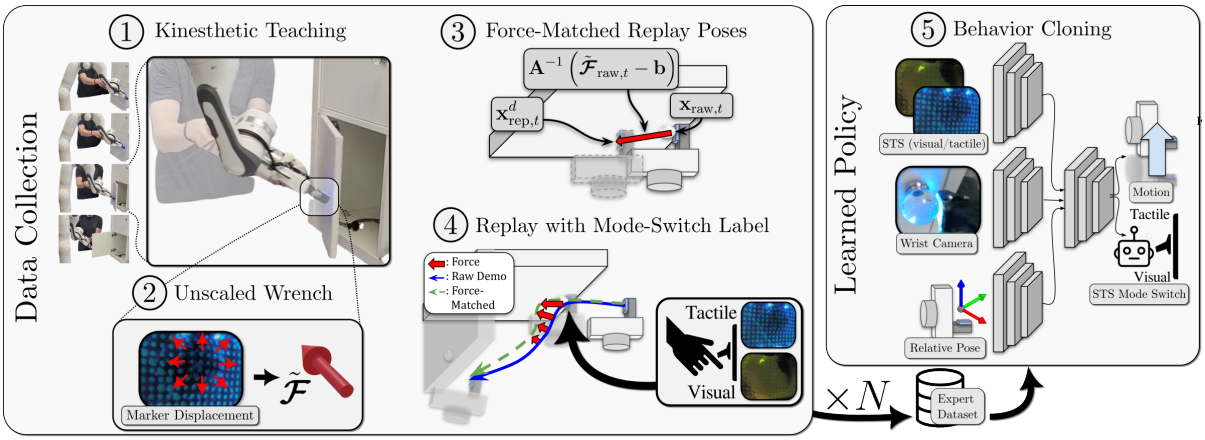


Figure 4.3: Visual representations of each component of our system: (1) Raw, human demonstrations are generated via kinesthetic teaching. (2) During the demonstration, an STS sensor in tactile mode allows us to read a four dimensions of an unscaled wrench in x, y, z, and rotationally about z. (3) For each timestep t from the demonstration trajectory from (1), each raw demonstration pose $\mathbf{x}_{\text{raw},t}$ uses the linear calibration parameters $\mathbf{A}$ and $\mathbf{b}$ (relating unscaled $\tilde{\boldsymbol{\mathcal{F}}}$ from (2) to control error $\mathbf{e}$) and the measured wrench $\tilde{\boldsymbol{\mathcal{F}}}_{\text{raw},t}$ from (2) to generate a *force-matched* replay pose $\mathbf{x}^d_{\text{rep},t}$. (4) The new, modified replay poses are used to replay the demonstration while a human provides an STS mode switch label. These replayed, force-matched demonstrations are stored in an expert dataset containing STS, wrist camera, and relative pose data as observations, as well as robot motion and STS mode labels as actions. (5) We train policies using some or all of STS, wrist camera, and relative pose data with behavioural cloning.

To take advantage of both visual and tactile modes of an STS sensor, a method is required to decide when to switch modes. Previous work (Hogan et al., 2022) accomplished this using known object information and a hand-crafted rule. Our second contribution is a novel method for switching between the visual and tactile modes of an STS sensor. We include mode switching as a policy output, allowing the human to set the sensor mode during the demonstration replay, which acts as a label for the expert dataset (see Step 4 of Fig. 4.3). We find that this approach effectively learns to switch the sensor at the point of contact, significantly improving policy performance compared with single-mode sensing. Our third contribution in this chapter is an extensive experimental evaluation of the benefits of including STS visual and tactile data (depending on the active mode) as inputs to a multimodal control policy. We compare the use of an STS sensor (both with and without mode switching) with the use of an eye-in-hand camera. Together, our contributions exhibit a system for significantly improving performance on contact-rich manipulation tasks by leveraging an STS sensor.

4.2 Related Work

Our work can be situated within the literature at the intersection of imitation learning, tactile sensing, and impedance control.

Impedance control is an approach to robotic control in which force and position are related by the dynamics of a theoretical mass-spring-damper system (Hogan, 1984). Impedance control can be easier to employ in robotic manipulation than standard force control (Siciliano, 2009) because it allows for

position control and because the contact dynamics between the robot and the environment are often difficult to model. It is not possible to apply desired forces in this scheme, and hybrid position/force control must be used instead (Raibert and Craig, 1981).

The development of gel-based optical visuotactile sensors has led to a range of research on tactile feedback (Yuan et al., 2017; Padmanabha et al., 2020; Ma et al., 2019). Using a semi-transparent polymer on top of these sensors allows for both visual and tactile sensing (Yamaguchi and Atkeson, 2017), and is further improved through the addition of controllable lighting (Hogan et al., 2021). These sensors can be used to map normal and shear forces to displacements by tracking printed dots embedded in the sensor membrane (Yuan, 2014; Ma et al., 2019; Kim et al., 2022). In this work, we determine the approximate applied wrench based on tracked marker motion. For example, in the z dimension, we use the average surface depth change, using an algorithm that estimates the depth via dot displacement (Jilani, 2024).

Imitation learning (IL) is an approach for training a control policy given a set of expert demonstrations of the desired behaviour. IL can generally be separated into methods based on behavioural cloning (Bain and Sammut, 1996), in which supervised learning is carried out on the expert demonstration set, or inverse reinforcement learning (Abbeel and Ng, 2004), where the expert's reward function is inferred from the demonstration set. Both behavioural cloning (Ablett et al., 2021b; Mandlekar et al., 2022; Zhang et al., 2018) and inverse reinforcement learning (Ablett et al., 2023; Chang et al., 2024; Orsini et al., 2021) have been used successfully for many robotic manipulation tasks. Recent applications tend to avoid the use of kinesthetic teaching (Billard et al., 2016) in favour of teleoperation, despite the ability of the former to provide a degree of haptic feedback to the demonstrator. Teleoperation requires a proxy system to provide haptic feedback (Ablett et al., 2021b; Li et al., 2023b) that may be inaccurate or expensive. Although we do not compare teleoperation to kinesthetic teaching, previous work has shown kinesthetic teaching to be preferred for many tasks for its ease of use and speed of demonstration (Pervez et al., 2017; Fischer et al., 2016; Akgun and Subramanian, 2011).

Prior work that combines kinesthetic teaching with force profile reproduction does not externally measure forces (Lee et al., 2015; Abu-Dakka et al., 2018), or requires one demonstration for positions and a separate demonstration for forces (Kormushev et al., 2011), which can be inconvenient and difficult to provide. In our work, a single demonstration provides both the desired poses and forces.

Learning-based manipulation has benefited from the use of force-torque sensing (Chebotar et al., 2014; Limoyo et al., 2023), and visuotactile sensing for both reinforcement learning (Hansen et al., 2022) and imitation learning (Huang and Bajcsy, 2020; Li et al., 2023a). Our learning architecture is similar to (Li et al., 2023a), in which tactile and visual data are fed to a single neural network that is trained with behavioural cloning. In (Li et al., 2023a), demonstrations are generated primarily with task-specific scripts, while we use kinesthetic teaching without any task-based assumptions or scripted policies. Our work is closely related to (Hogan et al., 2022), in which a multimodal tactile sensor is used to complete a bead-maze task with a robotic manipulator. Unlike (Hogan et al., 2022), we learn a single, unified policy for motion and sensor mode-switching, instead of relying on two separate, hand-crafted policies.

4.3 Methodology

In this section, we introduce each component of our system. We first provide a brief background on imitation learning (Section 4.3.1), Cartesian impedance control (Section 4.3.2), and kinesthetic teaching (Section 4.3.3). We then present our methods for (i) *force matching*, to match expert demonstrator wrenches (Section 4.3.4), (ii) *measuring unscaled forces*, where we explain how we use tactile displacement fields to provide wrench estimates (Section 4.3.5), (iii) *tactile force matching*, where we implement force matching using tactile wrench estimates (Section 4.3.6), and (iv) *contact mode labelling*, to supervise the visuotactile modality during data collection (Section 4.3.7). Finally, we provide our training objective in Section 4.3.8.

4.3.1 Markov Decision Processes and Imitation Learning

A Markov decision process (MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, R, \mathcal{P}, \rho_0 \rangle$, where the sets $\mathcal{S}$ and $\mathcal{A}$ are respectively the state and action space, $R : S \times A \to \mathbb{R}$ is a reward function, $\mathcal{P}$ is the state-transition environment dynamics distribution and ρ_0 is the initial state distribution. A deterministic policy $\pi(s)$ generates actions a. The policy π interacts with the environment to yield the experience (s_t, a_t, r_t, s_{t+1}) for $t = 0, \dots, T$ where $s_0 \sim \rho_0(\cdot), a_t = \pi(s_t), s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t), r_t = R(s_t, a_t)$, T is the finite horizon length, and $\tau_{t:T} = \{(s_t, a_t), \dots, (s_T, a_T)\}$ is the trajectory starting with (s_t, a_t).

We focus on imitation learning (IL), where R is unknown during training and instead we are given a finite set of task-specific expert demonstration pairs (s, a), $\mathcal{D}_E = \{(s, a), \dots\}$. Our goal is to learn a policy that maximizes task performance on the same evaluation function used to generate the demonstrations, for exmaple, whether a door is fully opened.

4.3.2 Impedance Control

During data collection and policy execution, we control the robot arm using Cartesian impedance control. This control strategy enables us to apply predictable forces on the environment by controlling the stiffness, damping, and desired position of the robot end effector in Cartesian space. By adjusting these parameters, we can regulate the interaction forces between the robot and its environment while maintaining the desired position of the end effector (Hogan, 1984).

Consider a robot arm with the motion equation

$$\boldsymbol{\mathcal{F}} = \boldsymbol{\Lambda}(\mathbf{q})\ddot{\mathbf{x}} + \boldsymbol{\mu}(\mathbf{x}, \dot{\mathbf{x}}) + \boldsymbol{\gamma}(\mathbf{q}) + \boldsymbol{\eta}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\mathcal{F}}_{\text{ext}}, \tag{4.1}$$

where $\dim \boldsymbol{\mathcal{F}} = \dim \mathbf{x} = 6$, $\boldsymbol{\mathcal{F}}$ is task-space wrench, $\mathbf{q}$ is joint position, $\mathbf{x}$ is task-space pose (where rotations are treated as rotation vectors), $\boldsymbol{\Lambda}$ is the task-space inertia matrix, $\boldsymbol{\mu}$ is the generalized Coriolis and centrifugal force, $\boldsymbol{\gamma}$ is the gravitation, $\boldsymbol{\eta}$ represents further non-linear terms, and $\boldsymbol{\mathcal{F}}_{\text{ext}}$ are environmental contacts. An impedance control law can be defined by

$$\boldsymbol{\mathcal{F}} = \mathbf{K}\mathbf{e} + \mathbf{D}\dot{\mathbf{e}} + \hat{\boldsymbol{\Lambda}}(\mathbf{q})\ddot{\mathbf{x}}^d + \hat{\boldsymbol{\mu}}(\mathbf{x}, \dot{\mathbf{x}}) + \hat{\boldsymbol{\gamma}}(\mathbf{q}) + \hat{\boldsymbol{\eta}}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\mathcal{F}}_{\text{ext}}, \tag{4.2}$$

where $\mathbf{x}^d$ is desired task-space pose, $\mathbf{e} = \mathbf{x}^d - \mathbf{x}$ is the task-space error, $\mathbf{K}$ and $\mathbf{D}$ are the selected task-space stiffness and damping matrices, and $\hat{\mathbf{\Lambda}}, \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\gamma}}$ and $\hat{\boldsymbol{\eta}}$ are the internal models of corresponding terms from Eq. (4.1). Substituting Eq. (4.2) into Eq. (4.1), we arrive at the closed-loop dynamics given by

$$\boldsymbol{\mathcal{F}}_{\text{ext}} = \mathbf{K}\mathbf{e} + \mathbf{D}\dot{\mathbf{e}} + \mathbf{\Lambda}\ddot{\mathbf{e}}. \tag{4.3}$$

4.3.3 Data Collection with Kinesthetic Teaching

In this section, we explain our method for generating raw demonstrations via kinesthetic teaching using impedance control, and why this method motivates force matching. We collect one expert dataset $\mathcal{D}_E$ for each task separately using kinesthetic teaching, where the expert physically pushes the robot to generate demonstrations (Billard et al., 2016) (see left side of Fig. 4.3 for an example). To allow for demonstrations via kinesthetic teaching, we set $\mathbf{K}$ and $\mathbf{D}$ from Eq. (4.3) very close to zero, ensuring the robot has full compliance with the environment. We then record end-effector poses $\mathbf{x}_E$ at a fixed rate as the robot is moved by the human, and use these poses, or, equivalently, the changes between poses, as expert actions (Billard et al., 2016). This recording process suffers from two limitations, however: (i) the recorded states and actions may not accurately reflect $\mathcal{S}$ and $\mathcal{A}$, and (ii) the recorded trajectory is unable to replicate the reference forces generated by the human during robot-environment contact. An example the first case above for $\mathcal{S}$ would be the presence of the human demonstrator, or even the shadow of the human demonstrator, in the frame of a camera being used to generate $\mathcal{S}$. For $\mathcal{A}$, it is difficult to guarantee that the controller can accurately reproduce the motion generated under full compliance while the human is pushing the arm.

This issue can be resolved with *replays*, in which the demonstrator generates a single demonstration trajectory $\tau_{x,\text{raw}} = \{\mathbf{x}_{\text{raw},0}, \ldots, \mathbf{x}_{\text{raw},T}\}$, resets the environment to the same s_0, and then uses a sufficiently accurate controller to reproduce each $\mathbf{x}_{\text{raw}}$ (Dasari et al., 2021). For trajectories in free space or where minimal force is exerted on the environment, this can be enough to learn effective policies (Dasari et al., 2021; Figueroa, 2023). The resolution of the second limitation requires additional sensory input, as we discuss in the next section.

4.3.4 Force Matching

In this section, we explain our method for generating *force-matched* replays of our raw kinesthetic teaching trajectories under the assumption that the true external end-effector contact wrench can be measured. As a reminder, force-matched replays recreate both the poses *and* the forces from the kinesthetic teaching trajectory, whereas standard kinesthetic teaching typically only recreates the poses.

Assuming static equilibrium with $\dot{\mathbf{e}} = \ddot{\mathbf{e}} = 0$, Eq. (4.3) simplifies to

$$\boldsymbol{\mathcal{F}} = \mathbf{K}\mathbf{e}^x, \tag{4.4}$$

where the position and external force are related as a spring and where we have dropped $(\cdot)_{\text{ext}}$ from external wrench $\boldsymbol{\mathcal{F}}$ for notational convenience. In our case, $\dim \boldsymbol{\mathcal{F}} = \dim \mathbf{e}^x = 6$, since we control

the robot in six degrees of freedom (three translational, three rotational). We treat the rotational components of $\mathbf{e}^x$ as a rotation vector. On our robot, we control joint torques [3] $\boldsymbol{\tau}$, such that

$$\boldsymbol{\tau} = \mathbf{J}^\top \boldsymbol{\mathcal{F}}, \tag{4.5}$$

where $\mathbf{J}$ is the the current manipulator Jacobian. Our per-timestep discrete control setpoints are

$$\boldsymbol{\mathcal{F}}_t = \mathbf{K}\left(\mathbf{x}_t^d - \mathbf{x}_t\right), \tag{4.6}$$

$$\boldsymbol{\tau}_t^d = \mathbf{J}^\top \boldsymbol{\mathcal{F}}_t. \tag{4.7}$$

Substituting Eq. (4.6) into Eq. (4.7) yields

$$\boldsymbol{\tau}_t^d = \mathbf{J}^\top \mathbf{K}\left(\mathbf{x}_t^d - \mathbf{x}_t\right), \tag{4.8}$$

illustrating that measuring external end-effector contact wrenches is not necessary for the indirect control scheme employed by a standard impedance controller (Villani and De Schutter, 2016).

Consider a raw demonstrator trajectory of recorded end-effector poses $\tau_{x,\text{raw}} = \{\mathbf{x}_{\text{raw},0}, \ldots, \mathbf{x}_{\text{raw},T}\}$ and wrenches $\tau_{f,\text{raw}} = \{\boldsymbol{\mathcal{F}}_{\text{raw},0}, \ldots, \boldsymbol{\mathcal{F}}_{\text{raw},T}\}$ as a set of poses and wrenches that we would like our controller to achieve. We invert the spring relationship in Eq. (4.6) to show how we can instead solve for desired (replay) positions $\mathbf{x}_{\text{rep},t}^d$ that would generate particular wrenches $\boldsymbol{\mathcal{F}}_{\text{raw},t}$ as

$$\mathbf{x}_{\text{rep},t}^d = \mathbf{K}^{-1}\boldsymbol{\mathcal{F}}_{\text{raw},t} + \mathbf{x}_{\text{raw},t}, \tag{4.9}$$

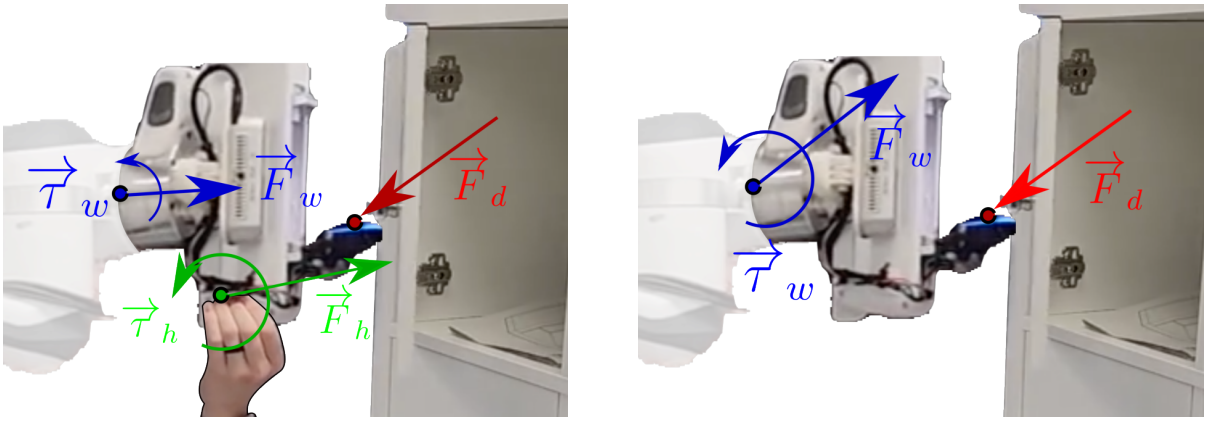
as illustrated in the replay pose generation step of Fig. 4.3. Considering the modified *replay* poses from Eq. (4.9) and substituting these into Eq. (4.8) as our new desired poses, we obtain

$$\begin{aligned} \boldsymbol{\tau}_t^d &= \mathbf{J}^\top \mathbf{K}\left(\mathbf{K}^{-1}\boldsymbol{\mathcal{F}}_{\text{raw},t} + \mathbf{x}_{\text{raw},t} - \mathbf{x}_t\right) \\ &= \mathbf{J}^\top \boldsymbol{\mathcal{F}}_{\text{raw},t} + \mathbf{J}^\top \mathbf{K}\left(\mathbf{x}_{\text{raw},t} - \mathbf{x}_t\right). \end{aligned} \tag{4.10}$$

Eq. (4.10) shows that we have modified the controller with an open-loop term to directly reproduce $\boldsymbol{\mathcal{F}}_{\text{raw},t}$, assuming static conditions, while maintaining the same original stiffness/impedance control term to reproduce $\mathbf{x}_{\text{raw},t}$. In cases where $\boldsymbol{\mathcal{F}}_{\text{raw},t} = \mathbf{0}$, our approach acts as a simple position-based controller.

We consider the stiffness $\mathbf{K}$ to be fixed (typically a diagonal matrix with one value for all translational components, and another for all rotational ones), selected to optimally trade off control accuracy and environmental compliance. Using Eq. (4.9), we can generate a trajectory of replay pose setpoints $\tau_{x,\text{rep}}^d = \{\mathbf{x}_{\text{rep},0}^d, \ldots, \mathbf{x}_{\text{rep},T}^d\}$, a new set of poses that, under static conditions, would reproduce the both the positions $\tau_{x,\text{raw}}$ and the wrenches $\tau_{f,\text{raw}}$ from the raw kinesthetic teaching trajectory.

[3] Note that our symbol for joint torques $\boldsymbol{\tau}$ (in bold) does not refer to a vector form of a trajectory τ. We choose these symbols to be consistent with existing literature on both force control and imitation learning.



(a) Kinesthetic Teaching. **(b) Autonomous Execution.**

Figure 4.4: With a human hand generating $\vec{F}_h$ and $\vec{\tau}_h$, wrenches measured at the wrist ($\vec{F}_w, \vec{\tau}_w$) via typical force-torque sensing modalities cannot isolate $\vec{F}_d$, as required for the force matching procedure outlined in Section 4.3.4. This notation applies only to this figure.

4.3.5 Measuring Unscaled Forces with A Tactile Sensor

Our force matching method requires access to robot-environment contact wrenches. Common approaches to measuring external end-effector wrenches include the use of a wrist-mounted force torque sensor, or kinematics and dynamics modelling combined with joint torque sensors. Fig. 4.4 shows estimated wrist wrenches on a robotic end effector, and illustrates that robot-environment contact wrenches cannot be decoupled from human-generated wrenches, making wrist-measured wrench values inadequate for our purposes. Our method requires measuring wrenches at the point of contact, making our visuotactile sensor a natural sensor choice. Below, we provide further detail on how we measure approximate, unscaled wrench signals $\tilde{\boldsymbol{\mathcal{F}}}$ with a visuotactile sensor. In the following section (Section 4.3.6), we describe how to calibrate these wrench signals for use in force matching.

Prior work has shown that the relationship between surface deformation and normal force is linear for membrane-based optical tactile sensors in their elastic region (Yuan, 2014), and as such, our approaches to approximating wrenches are based on measuring surface deformation via sensor surface marker tracking. We track the locations of these markers in the RGB camera using OpenCV's adaptive threshold (Bradski, 2000) as well as common filtering strategies, including low-pass filtering and a scheme for rejecting outlier marker displacements based on nearest neighbour distributions. Values for $\tilde{\mathcal{F}}_x, \tilde{\mathcal{F}}_y, \tilde{\mathcal{F}}_z$ and $\tilde{\tau}_z$ are then measured via separate methods, described below (we do not attempt to measure $\tilde{\tau}_x$ or $\tilde{\tau}_y$).

For $\tilde{\mathcal{F}}_z$ (i.e., perpendicular to the surface of the finger), we use a method for estimating membrane depth, where depth is inferred from marker movement on the surface of the membrane. The method, introduced in (Jilani, 2024), uses a perspective camera model to recover a relationship between the separation of markers (locally), and the amount of displacement of the surface towards the camera. Specifically, the Voronoi diagram between the markers and its corresponding medial axis are used to compute the changes in nearest-neighbour marker distance, giving a robust estimate of each

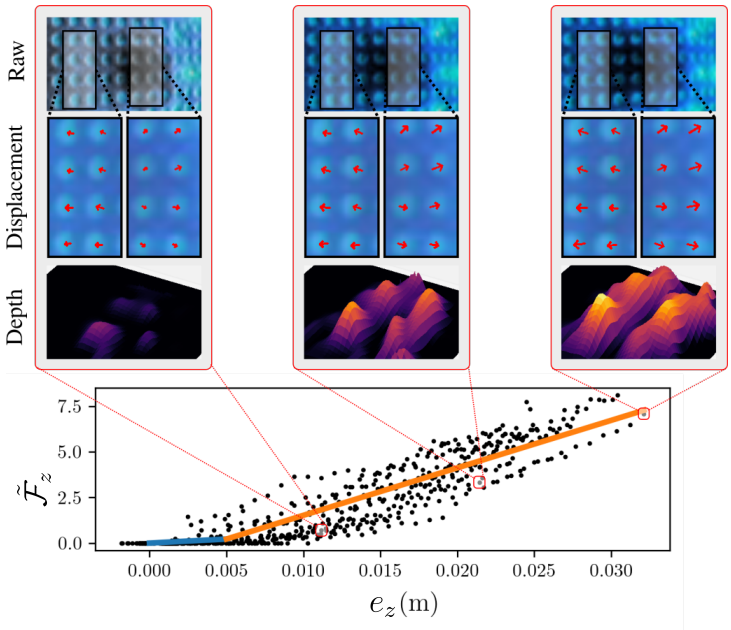


Figure 4.5: Example raw images along with corresponding marker displacements, inferred depths (Jilani, 2024), and e_z and $\tilde{\mathcal{F}}_z$ values, along with the piecewise linear relationship between e_z and $\tilde{\mathcal{F}}_z$. See supplementary materials for corresponding video.

marker's displacement towards the camera (Jilani, 2024). We use the average of each of these marker depths (displacements) for the normal force $\tilde{\mathcal{F}}_z$. Fig. 4.5 shows examples of both dot displacement and corresponding estimated depth at all points as the knob is pushed against the sensor, as well as our corresponding estimates for $\tilde{\mathcal{F}}_z$.

For approximate shear forces ($\tilde{\mathcal{F}}_x$ and $\tilde{\mathcal{F}}_y$), we use the average of the tracked dot movement in both the horizontal and vertical directions parallel to the sensor plane, For torque (only $\tilde{\tau}_z$), we use the average of the tracked dot movement that is perpendicular to an estimate of a center point of maximal normal force. Finally, while it would be possible to calibrate the transform between our robot and sensor frames using standard eye-in-hand manipulator calibration (Tsai and Lenz, 1989) or a form of touch-based eye-in-hand calibration (Limoyo et al., 2018), we find that assuming a fixed transformation is adequate.

4.3.6 Tactile Force Matching via Calibration

As a reminder, our goal is to generate replay trajectories $\tau^d_{x,\text{rep}}$ with Eq. (4.9) using a means of sensing robot-environment wrenches $\tau_{f,\text{raw}}$. Under the assumption that the deformation signals described in Section 4.3.5 are linear with respect to applied wrench, and the assumption that our impedance controller adequately models the spring relationship in Eq. (4.4), we can directly find the relationship between our unscaled wrench $\tilde{F}$ and control position error $\mathbf{e}^x = \mathbf{x}^d_t - \mathbf{x}_t$ by solving the linear least squares problem

$$\tilde{\boldsymbol{\mathcal{F}}} = \mathbf{A}\mathbf{e} + \mathbf{b}, \tag{4.11}$$

Figure 4.6: A front view of our finger-STS sensor in visual mode (left) and tactile mode (right), mounted on a Franka Emika Panda gripper.

where we drop the superscript x from $\mathbf{e}^x$ for brevity, and $\mathbf{A}$ and $\mathbf{b}$ are a matrix and vector, respectively, defining the linear relationship between $\tilde{\boldsymbol{\mathcal{F}}}$ and $\mathbf{e}$.

We generate values for $\tilde{\boldsymbol{\mathcal{F}}}$ and $\mathbf{e}$ through a short calibration procedure, where we incrementally push the sensor against the static environment numerous times, providing small initial perturbations to each trajectory to increase robustness. The calibration procedure is not object-specific and is only performed once before completing all of our experiments. Furthermore, we modify Eq. (4.11) to be piecewise linear, as we find that this fits our data more adequately. See the bottom of Fig. 4.5 for an example of collected $\tilde{\mathcal{F}}_z$ and e_z values, as well as the corresponding piecewise linear model. For further description of specific design choices related to this procedure in practice, see Section 4.4.1.

Returning to our original goal, we can rearrange Eq. (4.11) to resemble Eq. (4.9), yielding the individual poses in $\tau^d_{x,\text{rep}}$ as

$$\mathbf{x}^d_{\text{rep},t} = \mathbf{A}^{-1}\left(\tilde{\boldsymbol{\mathcal{F}}}_{\text{raw},t} - \mathbf{b}\right) + \mathbf{x}_{\text{raw},t}. \tag{4.12}$$

An advantage of this approach is that it is valid for any type of sensor that follows Hooke's law (e.g. visuotactile, pressure, or strain-gauge sensors). Furthermore, any repeatable control errors (where true joint torques do not match desired joint torques specified in Eq. (4.7)) or mild physical deformation of the sensor housing are accommodated, whereas simply using a finger-mounted force-torque sensor with Eq. (4.9) would not handle these issues. We identify three potential sources of error in this procedure: (i) measurement errors, where the same $\mathbf{e}^x$ value will result in different values for $\tilde{\boldsymbol{\mathcal{F}}}$, and (ii) hysteresis errors, which we do not attempt to mitigate, and (iii) aliasing errors, where different true six-dimensional $\boldsymbol{\mathcal{F}}$ values project to the same four-dimensional $\tilde{\boldsymbol{\mathcal{F}}}$.

4.3.7 STS Sensor Mode Labelling

Unlike a regular visuotactile sensor (Yuan, 2014; Ma et al., 2019), an STS sensor has a semi-transparent membrane and a ring of controllable LED lights, as shown in Fig. 4.6. The STS sensor can operate in two modes: *visual mode*, where the LEDs are off, allowing for pre-contact scene observation by the camera, and *tactile mode*, where the LEDs are on, enabling contact feedback.

An STS sensor requires a method to control the switching between visual and tactile modes. We treat the mode switching signal as an output from our control policy, and let the human demonstrator switch the mode of the sensor as part of their demonstration. During the replay phase, the controller autonomously generates $\tau_{x,\text{rep}}$, and the demonstrator observes and presses a button to change the sensor mode. This mode change is also used as an action label for training policy outputs. Our approach

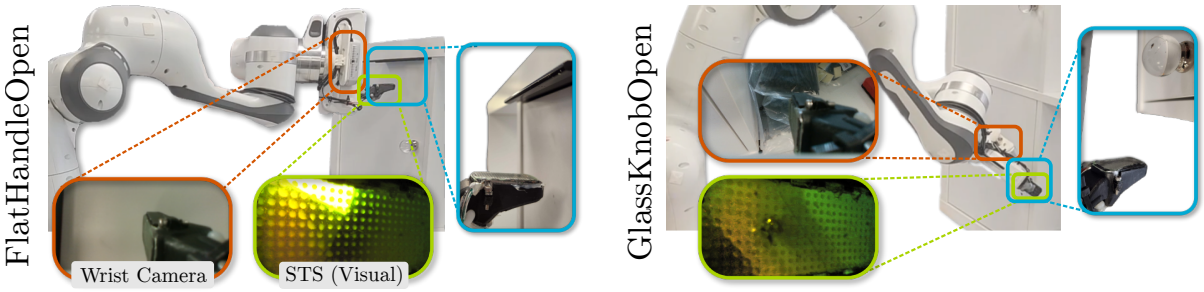


Figure 4.7: Our robot and sensor setup for `FlatHandleOpen` and `GlassKnobOpen`, showing example sensor data and a zoomed in view of STS and the handle/knob. Note that the glass knob on the top door is not used for experiments.

ensures that the visuotactile images that are added to the expert dataset $\mathcal{D}_E$ contain both visual and tactile data, since the initial demonstration of $\tau_{x,\text{raw}}$ had the sensor mode exclusively set to to tactile mode to read $\tilde{\mathcal{F}}$.

An advantage of this data collection method is that the demonstrator can choose on a task-by-task basis whether the sensor mode switch should occur before contact, at the point of contact, or after contact. Tasks that require scene tracking until the point of contact may benefit from having the mode switch occur post-contact, while tasks which require a delicate and less forceful touch or grasp may benefit from the opposite.

4.3.8 Policy Training

The previous sections detailed how we go from the human demonstration trajectories $\tau_{x,\text{raw}}$ to force-matched replay trajectories $\tau_{x,\text{rep}}$. The motion commands used to generate these replay trajectories (the *desired* replay trajectories, $\tau^{d}_{x,\text{rep}}$) comprise the motion components of our actions a used for training policies. Our policies are trained with a standard mean-squared-error behavioural cloning loss,

$$\mathcal{L}(\pi) = \sum_{(s,a)\in\mathcal{D}_E} \left(\pi(s) - a\right)^2, \tag{4.13}$$

where $s \in \mathcal{S}$ and $a \in \mathcal{A}$. $\mathcal{S}$ can include any or all of raw STS images, wrist camera images, and robot pose information, while $\mathcal{A}$ includes motion control and sensor mode commands (see Section 4.4.1 for more details).

4.4 Experiments

Our experiments are designed to answer the following questions:

1. What are the benefits of force matching and policy mode switching for imitation learning via kinesthetic teaching?

2. Is a see-through visuotactile sensor (STS) a required policy input for our door manipulation tasks, or is a wrist-mounted eye-in-hand camera sufficient?

Figure 4.8: Example trajectories for each of our four tasks, showing the motion of the robot from approach through contact.

3. Can an STS sensor alone (i.e., without another external sensor) provide sufficient visual and tactile information to complete our door manipulation tasks successfully, and if so, is there a benefit to including mode switching?

In Section 4.4.1, we describe our experimental environment and task parameters. Next, we report the performance results of our system as a whole in Section 4.4.3, the benefits of using force matching in Section 4.4.4, and the benefits of our policy mode-switching output in Section 4.4.5. We follow with details from our observational space study (i.e., of whether success is possible with the eye-in-hand camera alone) in Section 4.4.6. We then give results for training and testing policies with STS data exclusively in Section 4.4.7. Finally, we examine expert data scaling in Section 4.4.8.

4.4.1 Environment and Task Parameters

We study cabinet door opening and closing, using one door with a flat metal handle and one with a spherical glass knob (see Fig. 4.7), giving us four total experimental tasks, hereafter referred to as `FlatHandleOpen`, `FlatHandleClose`, `GlassKnobOpen`, and `GlassKnobClose`. All tasks include an initial reaching/approach component (see Fig. 4.7, Fig. 4.8, and Fig. 4.9), where the initial pose of the robot relative to the the knob or handle is randomized. See Fig. 4.9 for visual examples of the continuous slipping between the finger and the knobs/handles throughout demonstrations.

Door opening tasks are considered successful if the door fully opens within a given time limit. Door closing tasks are considered successful if the door closes without "slamming": if the finger fully slips off of the knob or handle before the door is closed, the spring hinges cause it to loudly slam shut. In either case, failure occurs because the finger loses contact with the handle or knob before the motion is complete.

Our robotic platform is a Franka Emika Panda, and we use the default controller that comes with Polymetis (Lin et al., 2021) as our Cartesian impedance controller. For all tasks, at the beginning of

each episode, the initial pose of the end-effector frame is randomized to be within a 3 cm $\times$ 3 cm $\times$ 3 cm cube in free space, with the rotation about the global z-axis (upwards facing) randomized between -0.15 and $+0.15$ radians. Our training data can include wrist camera 212 $\times$ 120 pixel RGB images, raw 212 $\times$ 120 STS images (in either visual or tactile mode), and the current and previous relative end-effector poses (position, quaternion). In this work, by *relative poses*, we mean that, for each episode, the initial pose is set to $\{0, 0, 0, 0, 0, 0, 1\}$, although the *global pose* is randomized for every episode, as previously described. Finally, our action space consists of 6-DOF relative position changes in the frame of the end-effector. These choices are meant to simulate the situation in which an approximate reach is performed with an existing policy and global pose information between episodes is inconsistent, as is often the case in mobile manipulation (Ablett et al., 2021b).

Our sensor is based on the one described in (Hogan et al., 2022, 2021), but in a smaller form factor. The finger housing is 3D-printed and mounted on the Franka Emika Panda gripper. Owing to the fragility of the top layer of the sensor, especially when subject to large shear forces, we use the sensor with this top layer, as well as the semi-reflective paint, fully removed. The LED values and camera parameters for the sensor are set using a simple tool to optimize scene visibility in visual mode, and marker visibility in tactile mode.

As detailed in Section 4.3.5, a short calibration procedure allows us to solve for the parameters in Eq. (4.12). This procedure takes about seven minutes in practice, and is fully autonomous. We use our glass knob as the calibration object, and move the desired pose of the end-effector 3.5 cm towards the glass knob while moving at 1 mm increments, with each trajectory starting between 1 mm and 3 mm away from the knob. We also generate shear by moving 1 cm laterally, and torque by moving 1 rad rotationally, after already having made initial contact with the knob.

4.4.2 Imitation Learning and Training Parameters

Kinesthetic teaching data are collected at 10 Hz once the robot has exceeded a minimum initial movement threshold of 0.5 mm. After a kinesthetic demonstration is completed, the robot is reset to the same initial (global) pose, and the demonstration is replayed with or without force matching, and with or without the human providing a mode-switch label. Notably, this environment setup does not preclude the possibility of using an actual mobile manipulator to collect data and execute policies, as in (Ablett et al., 2021b). We collected 20 raw kinesthetic teaching demonstrations with a human expert for each task (80 total). With these demonstrations, we complete replay trajectories in a variety of configurations (described in Sections 4.4.4, 4.4.5 and 4.4.8, depending on which algorithm is being tested. A benefit of this approach is that we were able to not only minimize the required human labour of completing demonstrations, but we were also able to more directly compare the benefits of including force matching and mode switching since the raw human trajectories were identical for each configuration. For every configuration, we train policies with three different random seeds, and complete 10 test episodes per seed.

We train our policies in PyTorch with the Adam optimizer and a learning rate of 0.0003, halving the learning rate halfway through training. We use a ResNet-18 (He et al., 2016) architecture pre-

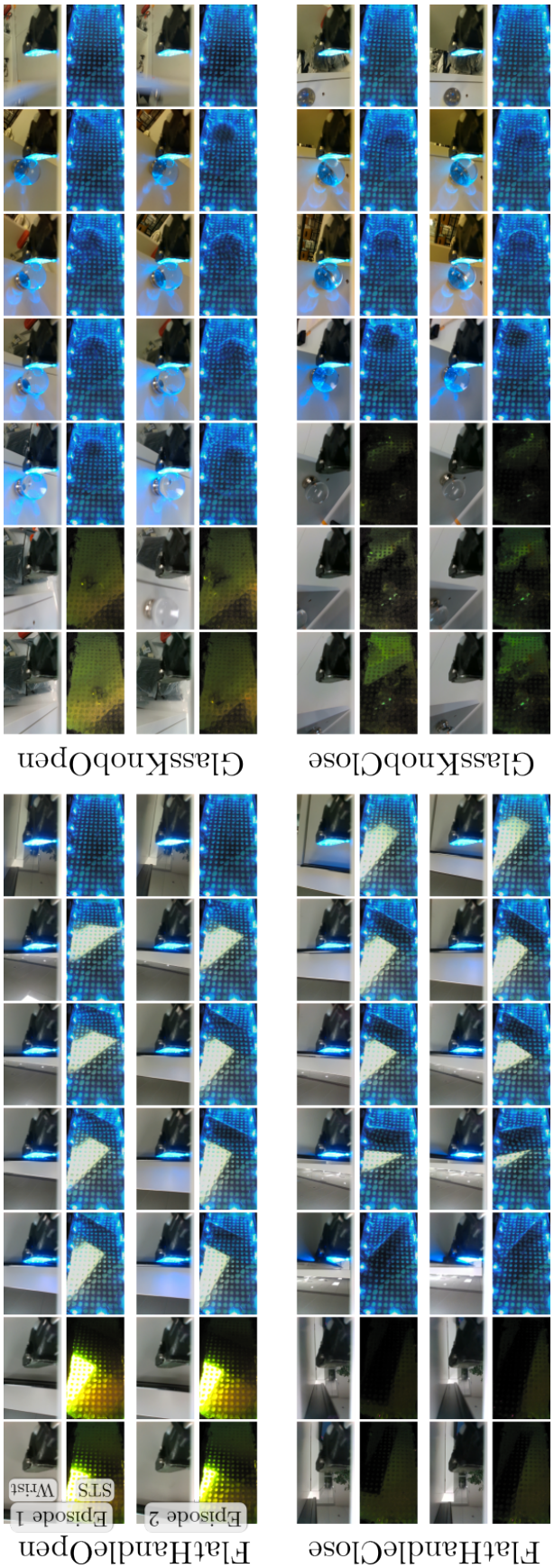


Figure 4.9: Wrist camera and STS data for two replayed demonstration trajectories per task (see Fig. 4.8 for corresponding examples showing the full scene). Notable features are differences in initial observations due to randomized initial poses, the change in appearance before and after the STS mode switches, the informative nature of the STS data compared with the wrist camera, and the amount of slip between the handle/knob and the sensor throughout each trajectory.

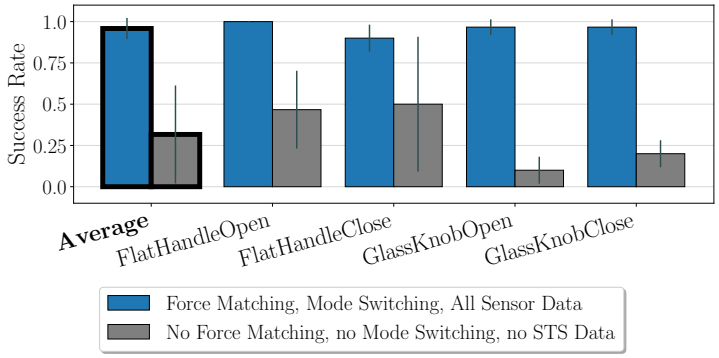


Figure 4.10: Performance results with and without each of the three novel additions presented in this work: force matching, STS mode switching, and STS as a policy input. There is a clear benefit across all tasks, with greater benefit for each of the `GlassKnob` tasks. For this and the following figures, the average across all tasks is shown in bold on the left and black lines indicate standard deviation of seeds.

trained on ImageNet and ending with spatial soft-argmax (Levine et al., 2016) for image data, and a small fully connected network for relative pose data. Features from each modality are concatenated and passed through another small fully connected network before outputting our seven-dimensional action: relative position change, orientation change as a rotation vector, and STS mode. All layers use ReLU non-linearities. We train each policy for 20k gradient steps using weight decay of 0.1 to avoid overfitting, as this has been shown to improve behavioural cloning results compared to early stopping (Mandlekar et al., 2022; Ablett et al., 2023).

4.4.3 System Performance

We evaluate the benefits of collectively including force matching, mode switching as a policy output, and STS data as a policy input by training policies with all three of these additions, as well as policies with none of them, and comparing their success rates. As stated in Section 4.4.2, we train three seeds per task with 10 test episodes per seed, that is, 30 episodes for each task and configuration, and 120 total test episodes per configuration. The comparison is visualized for each individual task, and with an overall average across all tasks, in Fig. 4.10. Including the three additions results in a 64.2% average absolute across-task performance gain over the baseline where none are included, demonstrating a clear benefit. There is a task-related correlation also, where the average improvement for the `FlatHandle` tasks is 46.7%, while the average improvement for `GlassKnob` tasks is 81.7%. The remaining sections ablate each of these additions individually to better understand each component's contribution.

4.4.4 Force Matching Performance

To evaluate the benefit of force matching, we complete replays of our raw expert datasets both with and without force matching and compare their performance. To isolate the benefits of force matching, and reduce the effect of including or excluding policy mode switching, we include policy mode switching in both configurations. We also include all three sensing modalities (wrist-mounted eye-in-hand RGB images, STS images, and relative positions) in both. The results of these tests are shown in Fig. 4.11.

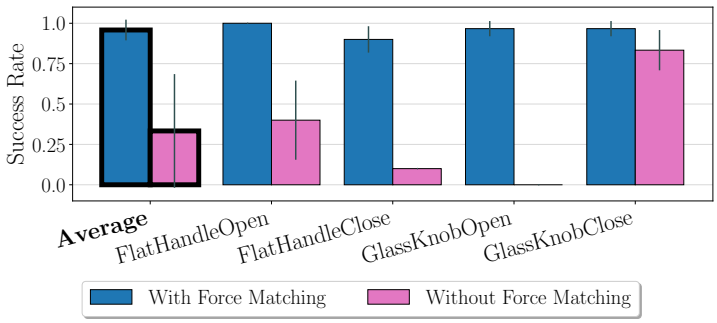


Figure 4.11: Performance results showing the effect of excluding force matching, while keeping mode switching and the STS as a policy input in both cases. Force matching improves performance in all tasks, with particularly large gains for `FlatHandleClose` and `GlassKnobOpen`.

The average absolute across-task performance increase with force matching is 62.5%, although the performance gain is greater for some tasks than for others. A notable case is the discrepancy between gains for the two `GlassKnob` tasks: the `GlassKnobOpen` policy always fails without force matching, while the `GlassKnobClose` policy sees only a slight benefit from force matching. This can partially be explained by the pose angle for much of the demonstration. This can partially be explained by the pose angle for much of the demonstration. For the `GlassKnobOpen` task, after contact is made with the knob, the door is almost exclusively pulled via a shear force. Poor initial contact caused by too little force often results in failure (see third column, third row of Fig. 4.12). In contrast, for the `GlassKnobClose` task, the angle of the robot finger relative to the knob ensures that normal force is still applied throughout much of the trajectory.

The force-matching results in Fig. 4.11 may appear contradictory to those from Fig. 4.10. In Fig. 4.11, the use of STS data without force matching actually performs *worse*, for `FlatHandleClose` and `GlassKnobOpen`, than excluding STS data and force matching (as shown in Fig. 4.10). We hypothesize that this is due to causal mismatch (de Haan et al., 2019): the highly suboptimal demonstration data generated without force matching may cause the policy to learn to switch the STS modality without firm contact. This then causes the policy to initiate the "open" or "close" phase (as opposed to the "reach" phase) of the task too early. We leave further investigation of this result to future work.

Force Matching Trajectory Comparison

While our policy learning experiments in Section 4.4.4 implicitly illustrate the value of force matching, Fig. 4.12 shows specific examples of how trajectories change with and without force matching. We label the initial timestep $t = 0$, final timestep $t = T$, and a single representative timestep for each trajectory, also including a cropped STS image at the representative timestep for each trajectory. Note that the No FM desired poses (bottom, green) are the same as the true demo poses (top, blue), while the difference between the desired poses and actual poses in the FM row lead to increased STS force (see Eq. (4.12)). In all four cases, the No FM replay causes the end-effector to slip off of the handle or knob, leading to failure.

We conclude that force matching generates $\tau_{x,\text{rep}}$ and $\tau_{\tilde{f},\text{rep}}$ that better match $\tau_{x,\text{raw}}$ and $\tau_{\tilde{f},\text{raw}}$.

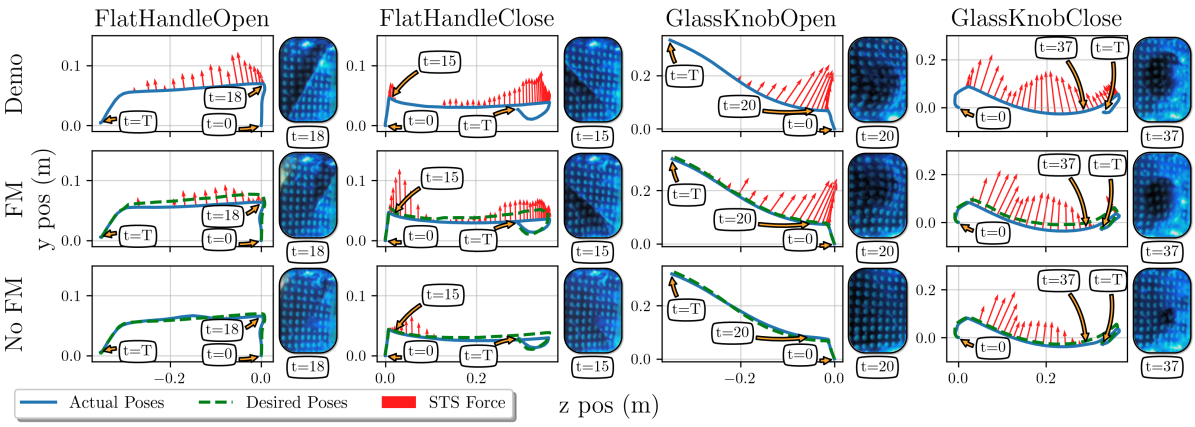


Figure 4.12: Top: a raw demonstration trajectory $\tau_{x,\text{raw}}$ (blue) as well as demonstration forces $\tau_{\tilde{f},\text{raw}}$ (red). Middle: a new set of desired poses that incorporate force matching $\tau^{d}_{x,\text{rep}}$ (green), the new set of replayed poses $\tau_{x,\text{rep}}$ given $\tau^{d}_{x,\text{rep}}$ (blue), and the actual forces with the modified trajectory $\tau_{\tilde{f},\text{rep}}$ (red). Bottom: a set of desired and actual poses, along with resulting forces, that use $\tau_{x,\text{raw}}$ directly (i.e., *without* force matching), while ignoring $\tau_{\tilde{f},\text{raw}}$, to generate a replay (green, blue, and red, respectively).

While $\tau_{\tilde{f},\text{rep}}$ occasionally has mismatches with $\tau_{\tilde{f},\text{raw}}$, most of these errors can be attributed to a combination of sensor reading errors (see noise in bottom of Fig. 4.5) and control errors.

4.4.5 Policy STS Mode Switching Performance

To better understand how mode switching can benefit performance on a real door-opening task, we collect replays of demonstrations both with and without mode switching. Specifically, we consider three configurations: mode switching enabled, tactile-only (the sensor lights are always on), and visual-only (the sensor lights are always off). Given the benefit of including force matching (as shown in Section 4.4.4) and to isolate the benefits of mode switching, we included force matching in all three configurations. We also include all three sensing modalities. Our results are shown in Fig. 4.13, using the same training and testing parameters as described in Section 4.4.2.

Averaging the success rates of the visual-only and tactile-only results together, the use of mode-

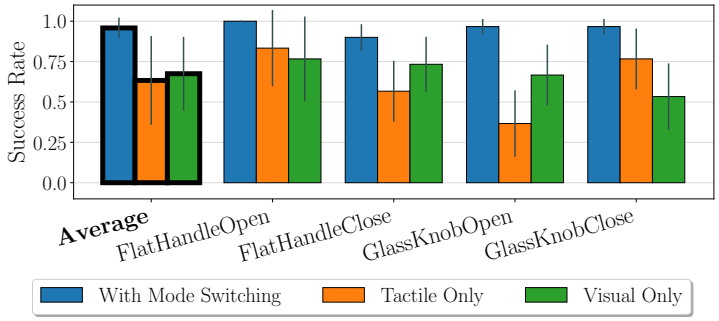


Figure 4.13: Performance results comparing the use of mode switching with setting the STS sensor to visual-only and tactile-only. Mode switching provides a clear benefit over keeping the sensor in a single mode, but there is no obvious pattern between whether a tactile-only or visual-only sensor would be preferred without mode switching.

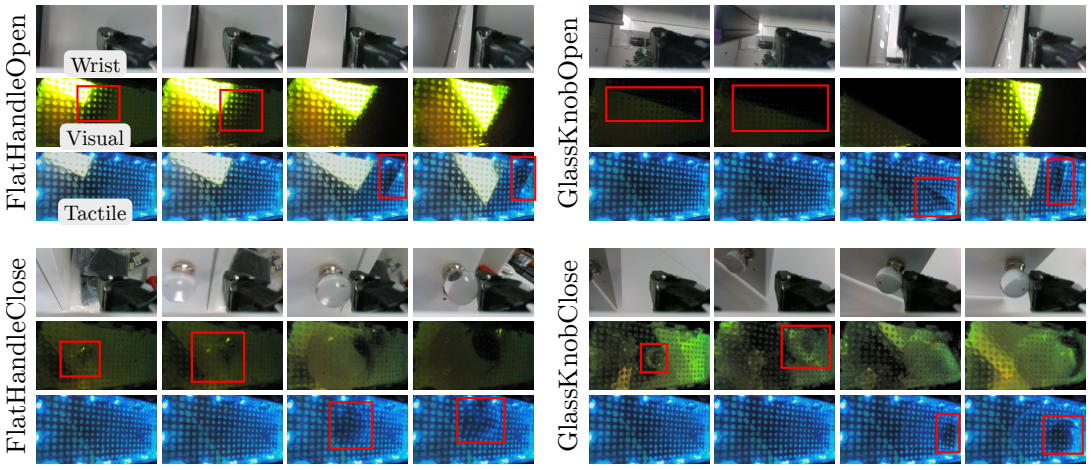


Figure 4.14: Partial trajectories from the same replay in visual-only and tactile-only for all four tasks. Red boxes highlight parts of the scene (knob, handle) that are significantly clearer in the respective sensor mode. The corresponding wrist camera images are provided on the top rows for reference.

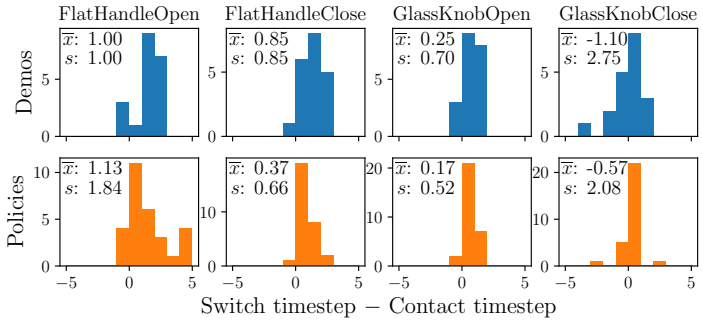


Figure 4.15: Histograms of the difference between mode switch action timestep and the true first contact timestep, for both our demonstrations and our learned policies with force matching, mode switching, and all sensing modalities.

switching results in an a task-average performance gain of 30.4%. As with force matching, there are some task-specific patterns. Task-specific correlations are similar to Section 4.4.3: the performance gain for `GlassKnob` tasks is 38.3%, while the performance gain for `FlatHandle` tasks is only 22.5%. This is perhaps unsurprising, given that the handle requires less precision to maintain contact. Qualitatively, Fig. 4.14 shows the value of including sensor mode switching: tactile-only data provides no information before contact, while visual-only data provides poor tactile information during contact.

Mode Switching Timing Analysis

While Section 4.4.5 helps verify the efficacy of mode switching in general, its evaluation of our specific mode switching policy output is only implicit. To evaluate the quality of our learned mode switching policy output, we completed an analysis of the timing of the learned mode switch action compared with the expert labels. An expert demonstrator may have a preference for having the sensor switch modes slightly before or after contact occurs, but for this analysis, we will consider that an optimal switch occurs at the moment of contact.

For each dataset with force matching, mode switching, and all three sensing modalities, we manually annotated each episode with the timestep at which contact was made between the handle/knob and the surface of the STS, and compared that to the timestep that the demonstrator provided a mode switch action label. The histograms of these timestep differences are shown in the top row of Fig. 4.15. While there are certain task-specific patterns, such as a slightly greater timestep difference average for the handle tasks, the clearest pattern is that the mode switch label usually occurs within one timestep (0.1s) of contact being made. The bottom row shows the same analysis, but for autonomous policies. With the exception of a few outliers (e.g., for `FlatHandleOpen`), the policies have converged to be close to an average of zero timesteps between contact and mode switching (i.e., smoothing out the reactive/predictive timing errors from the expert dataset). The outliers in Flat Handle are most likely due to a causal mismatch, where the policy learns to switch modes based on when the arm starts opening the handle, instead of at the moment of contact (de Haan et al., 2019).

4.4.6 Observation Space Study

The second experimental question we hope to answer is whether our door manipulation tasks can be complete with a wrist-mounted eye-in-hand camera alone, and whether the inclusion of STS data improves performance. To complete these tests, we do not collect any new human demonstration data or any new replay data, instead we selectively exclude some observations during training. All policies are trained with force matching enabled, and policies trained with STS data include mode switching. For policies that exclude STS data, we set the sensor to visual-only mode, since the wrist camera captures STS lighting changes that might provide a contact cue.

Results from these tests are shown in Fig. 4.16. Adding the STS sensor as an input yields in an average across-task increase in performance of 42.5%. It is worth reiterating that this performance gain only corresponds to the case where STS data is excluded as an input into the policy; the STS itself is still used indirectly for these policies through the use of force matching. Elaborating, for this baseline, the STS sensor is used exclusively in tactile mode and could be replaced with a regular visuotactile sensor or a finger-mounted force-torque sensor. Without force matching, the performance gain over the baseline is 64.2% (as shown in Section 4.4.3), indicating that force matching alone provides some

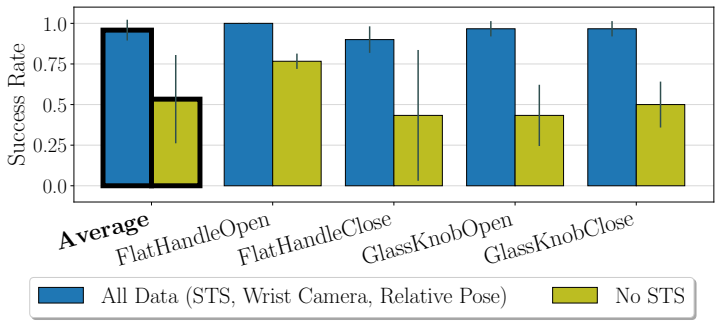


Figure 4.16: Performance when STS data are excluded as a policy input. Force matching is still included in this case. The data from the STS clearly provide substantial benefit for learned policies.

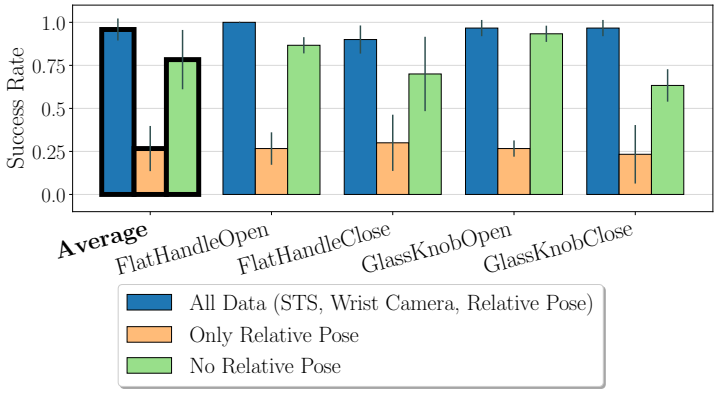


Figure 4.17: Performance results to illustrate the contribution of relative pose as a policy input. Relative pose alone is not capable of solving the tasks, but its inclusion along with each of our sensors does have a positive effect on performance.

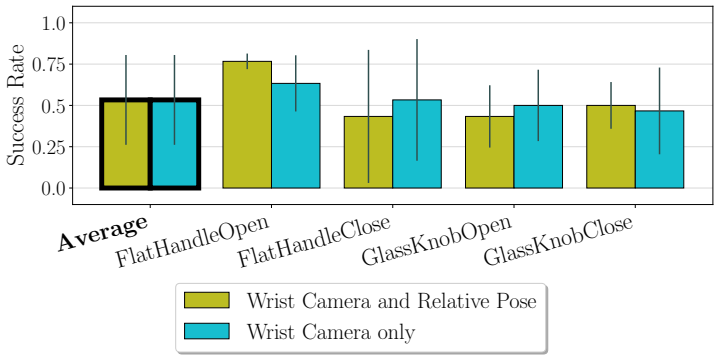


Figure 4.18: Performance results to evaluate the contribution of relative pose when excluding the STS. When only the eye-in-hand wrist-mounted camera is included, relative pose has a negligible effect on policy performance.

benefit, but that the STS as a policy input is still quite valuable. The average performance gain of 50% for the `GlassKnob` policies is again higher than 35% for the `FlatHandle` policies, indicating once more that the utility of the STS as a policy input is partially task-dependent.

Contribution of Relative Pose Input

To better understand the value of relative poses as inputs, we train policies with and without relative pose, as well as policies with *only* relative pose as an input (and no other changes). Results for training and testing in these three configurations are shown in Fig. 4.17. The relatively high performance of policies that exclude relative pose shows that the visual data alone often provides enough information to solve the task. However, including relative pose still yields an average across-task policy improvement of 17.5%.

We include relative pose alone as a baseline to show that, due to the initial randomization described in Section 4.4.1, the tasks require visual feedback to succeed consistently, and that a single "average" trajectory combined with impedance control is not, by itself, effective.

We also present the results of this comparison for policies excluding STS data in Fig. 4.18. The clear benefit of including relative pose shown in Fig. 4.17 is not exhibited when using eye-in-hand wrist camera data only. This result does not necessarily have a single explanation, but it does indicate that further study of the inclusion of numerical observation data is warranted in imitation learning, since prior work has shown it to have both a strictly positive (Zhang et al., 2018; Ablett et al., 2021b) and strictly negative (Mandlekar et al., 2022) effect.

4.4.7 STS-Only Policy Performance

Our third experimental question concerns whether an STS camera alone provides enough feedback to complete these challenging door manipulation tasks. Prior work has shown that simply adding more sensor data to learned models does not always lead to improved performance (Limoyo et al., 2020; Hansen et al., 2022; Mandlekar et al., 2022). In this section, we train and test several policies without using data from the wrist-mounted eye-in-hand camera.

The results in Fig. 4.19 show an average across-task absolute performance increase of 25.0% with the wrist camera data. The amount of improvement varies significantly by task, however, with no improvement for `FlatHandleClose` and `GlassKnobOpen`, but a dramatic improvement for `GlassKnobClose`. This is at least partially because the initial reach in `GlassKnobClose` is more difficult than for any of the other tasks, and the wrist camera provides a clear view of the approach. Excluding the wrist camera data (Fig. 4.19, red) from policy inputs results in a 17.5% higher success rate than excluding the STS data (Fig. 4.16, light green). This finding may indicate that contact-rich door tasks benefit more from a multimodal STS sensor alone than from a wrist camera alone.

Mode Switching Performance (STS-Only)

We also evaluate the benefits of our policy mode switching output, replicating the experiments in Section 4.4.5, but excluding the wrist camera data. Specifically, we train and test policies using STS data and in three configurations: with mode switching enabled, with the STS mode set to visual-only,

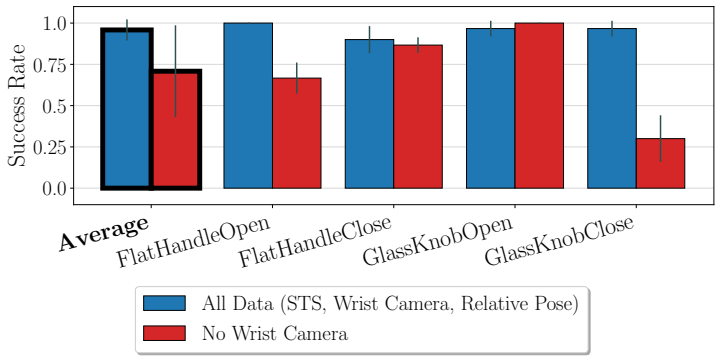


Figure 4.19: A performance comparison of the effect of excluding the eye-in-hand wrist-mounted RGB camera data while keeping both force matching and mode switching enabled. Performance deteriorates slightly in `FlatHandleOpen` and dramatically in `GlassKnobClose`, but an STS-only policy does perform adequately in certain cases.

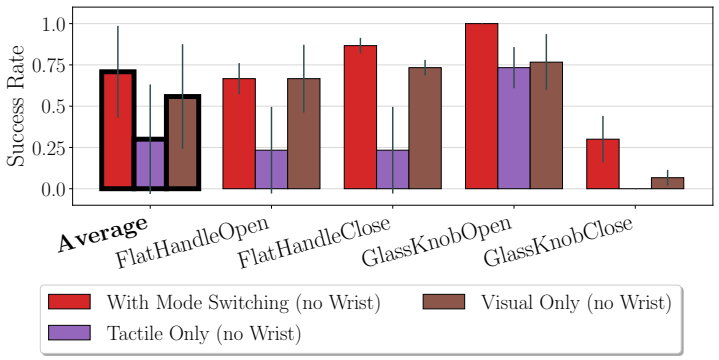


Figure 4.20: An STS-only variant of the mode switching performance results shown previously in Fig. 4.13. A similar pattern emerges, highlighting the value of mode switching. In the STS-only case, tactile-only policies are shown to be significantly poorer than visual-only policies.

and with the mode set to tactile-only. As in Section 4.4.5, we include both force matching and relative poses. Results are shown in Fig. 4.20. The average across-task performance gain, for visual-only and tactile-only combined, is 27.9%. This is comparable to the increase of 30.4% shown in Section 4.4.5 when wrist camera data is included.

One striking difference is the performance increase from mode switching over tactile-only policies of 40.8%, compared to only a 15.0% increase over visual-only policies. These tasks require reaching to make adequate contact with the handle or knob, and with both visual STS data and eye-in-hand wrist data absent, the tasks become much more difficult. Tactile-only performance matches visual-only performance for `GlassKnobOpen`, which may be because, even in tactile mode, the glass knob faintly shows up in the STS sensor images (see Fig. 4.21). However, even for `GlassKnobOpen`, mode switching is clearly the optimal configuration in which to use the STS sensor.

Contribution of Relative Pose Input (STS-only)

As a comparison with the effect of including relative positions as in input to policies with wrist camera data only (see Section 4.4.6), we also completed similar experiments for policies with only STS data. The results in Fig. 4.22 compare testing results for policies with both STS and relative position data, only

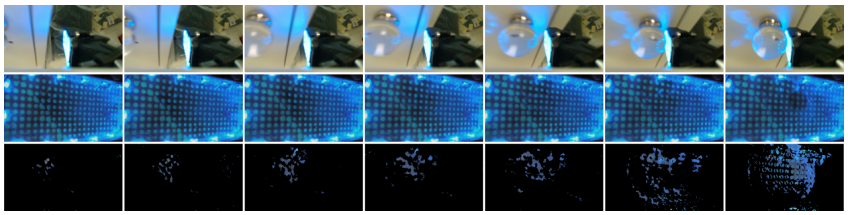


Figure 4.21: A short sub-trajectory of a `GlassKnobOpen` run showing the reaching phase with the STS sensor set to tactile-only mode. The top row shows the wrist camera view for reference, the middle row shows the (tactile) STS images, and the bottom row shows pixels that have changed since the previous timestep. The glass knob shows up faintly, explaining the surprisingly good performance of STS-only tactile-only policies in `GlassKnobOpen` from Fig. 4.20.

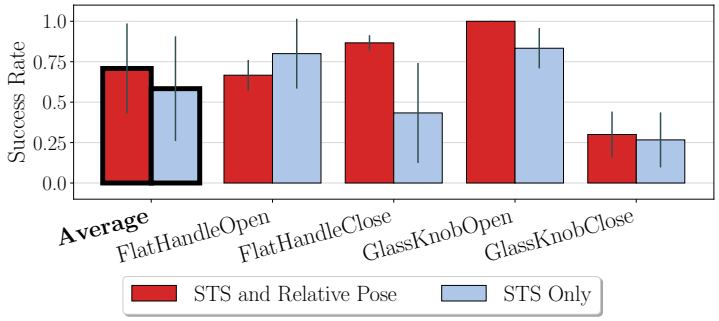


Figure 4.22: Performance results to evaluate the contribution of relative pose when excluding the wrist camera. The STS receives a small average bump in performance from adding relative pose as an input.

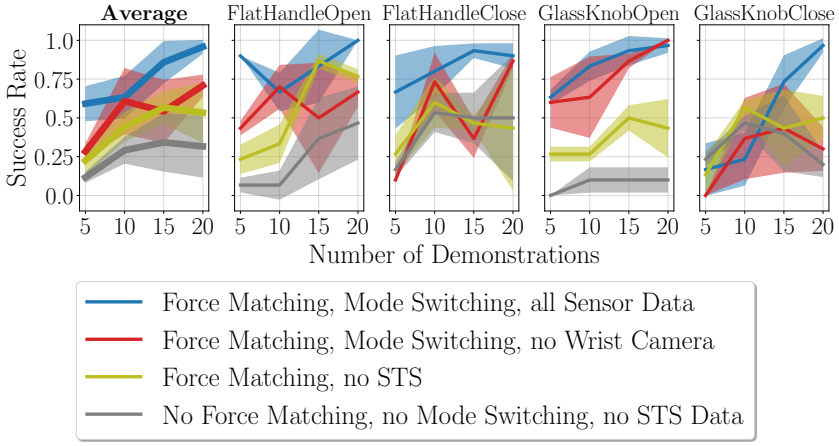


Figure 4.23: Performance with varying amounts of expert data for representative configurations from our prior experiments. The shading indicates one standard deviation. All modalities show increasing performance with increasing dataset sizes, but the two modalities without STS data reach a plateau at 15 demonstrations.

STS data, and only relative position data as inputs. In comparison with the results from Section 4.4.6, there is an average performance increase of 12.5% when using relative poses with STS compared to the STS alone. However, the gain is not as significant as one may expect, once again reiterating that deeper study on the benefits of including numerical observation data for policies learned with imitation learning is warranted.

4.4.8 Expert Data Scaling

In our final set of experiments, we choose representative configurations from the preceding sections and train and test policies with 5, 10, and 15 expert trajectories (compared with 20 for all of our previous experiments). Fig. 4.23 shows the results of experiments with less training data for these configurations. Most methods improve with more data, but there are exceptions. For example, both STS-free methods do not improve significantly for `FlatHandleClose`, `GlassKnobOpen`, or `GlassKnobClose`, indicating that the eye-in-hand wrist camera does not provide sufficient information to complete the task alone, regardless of data quantity. Conversely, eye-in-finger visual data, as well as tactile data, are sufficient for these three tasks, so their performance scales up with increasing data. A surprising find-

ing is that for `FlatHandleOpen`, `FlatHandleClose`, and `GlassKnobOpen`, performance is quite good with force matching, mode switching, and across all sensing modalities, even with only five demonstrations. Performance on the most difficult task, `GlassKnobClose`, clearly increases with more data, but the STS data alone are not sufficient to complete this task. Results for the two configurations that exclude STS input data plateau at 15 demonstrations, indicating that these tasks may be simply not possible to complete without the STS, even with an increasing amount of data.

4.4.9 Decoupling Human Forces from Contact Forces

As described in Section 4.3.5, common approaches to sensing end-effector force-torque, such as wrist-mounted force-torque sensors or joint-torque sensors with dynamics modelling, cause recorded wrenches $\tau_{f,\text{raw}}$ to be corrupted by the demonstrator's own force against the robot. To avoid this corruption, a sensor must be mounted at a point on the end-effector where human-applied forces are ignored during demonstrations, such as on the finger of a gripper.

In Fig. 4.24, we validate the presence of this corruption, and its resolution through the use of finger-mounted wrench sensing. The STS sensor correctly shows no readings in the circular trajectory, where there is no contact, and shows increasing and then decreasing force during contact in the trajectory where it is pushed against the knob. Conversely, the Panda force readings are primarily in response to the human pushing force against the robot, making it difficult to isolate end-effector to environment forces.

4.5 Limitations

In this section, we discuss some limitations of our work. Although we used a single sensor for all of our experiments, the gel-based contact surface of the STS physically degrades over time. This limitation partially motivates the use of learned models for gel-based sensors; if sensor data changes marginally

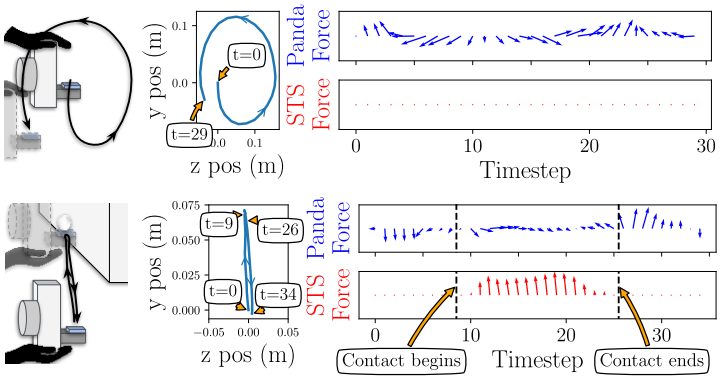


Figure 4.24: Example expert demonstrations showing measured forces (collapsed to 2-DOF) with Panda (the robot) joint-torque sensors and dynamics modelling (blue) and our own sensor (red). The top trajectory corresponds to pushing the end-effector in a circle without environmental contact, and the bottom trajectory corresponds to pushing the end-effector towards, against, and away from the door knob while fixed. Panda force readings are corrupted by human-robot interaction forces.

due to degradation, a practitioner can simply add more data to the dataset and retrain the policy. As well, the STS incorporates a standard camera, so policies trained on its data are susceptible to the same problems as other visual data paired with neural networks, such as overfitting to specific lighting conditions. Finally, our force-matched replayed demonstrations still occasionally fail because both our method for measuring forces as well as our Cartesian impedance controller can suffer from accuracy issues, but both of these limitations can be improved with further tuning.

4.6 Summary

In this chapter, we presented a robotic imitation learning system that leverages a see-through-your-skin (STS) visuotactile sensor as both a measurement device to improve demonstration quality and a multimodal source of raw data to learn from. Our first contribution was a method to use the STS in its tactile mode as a force-torque sensor to improve demonstration quality through *force-matched* replays that better recreate the demonstrator's force profile. Our second contribution was a learned approach to STS mode switching, in which a policy output is added to switch the sensor mode on the fly, and labels are provided by the demonstrator during demonstration replays. Our final contribution was an observational study in which we compared and contrasted the value of the STS visuotactile data (in mode-switching, visual-only, and tactile-only configurations) with the use of an eye-in-hand wrist-mounted camera on four challenging, contact-rich door opening and closing tasks on a real manipulator. We found that the inclusion of force matching, STS mode switching, and visuotactile data from an STS sensor as a policy input each increased policy success rates by 62.5%, 30.4%, and 42.5%, respectively.

The force matching method presented in this chapter exclusively uses behavioural cloning to generate a policy, which, as described in Section 2.4.3 and by Ross and Bagnell (2010), has a worst case cost of $\mathcal{O}(T^2\epsilon)$. Force matching does not directly reduce this worst-case dependency on T, but rather improves demonstration quality, which can be interpreted as lowering the error rate ϵ. Our results show that force matching clearly improves the success rate of policies, but it is likely that for longer-horizon tasks (i.e., increasing T), policies would still have higher cost, according to $\mathcal{O}(T^2\epsilon)$. An online component akin to DAgger (Ross et al., 2011) would be required to reduce the cost to $\mathcal{O}(T\epsilon)$, but accomplishing this with demonstrations based on replays, as presented in this chapter, might be challenging. As such, force matching is most beneficial for tasks with a relatively low T and low variance in the initial state.

Potential directions for future work include improving the accuracy of our force sensing approach and adding a means for self-improvement by automatically detecting execution failures.

Chapter 5

Failure Identification for Interventions

In this chapter, we present an approach to imitation learning (IL) that implicitly acknowledges the inherent distribution shift of behavioural cloning directly. If we can identify when an agent has entered an out-of-distribution region, we can stop execution and request a corrective demonstration from an expert to return the agent back in-distribution. These corrective data can be then appended to the expert dataset, gradually improving the policy. This can be viewed as an attempt to combine the convenience of offline behavioural cloning with the robustness of online reinforcement learning. The approach presented in this chapter is called **f**ailure **i**dentification to **r**educe **e**xpert burden (FIRE). Compared with true reinforcement learning, FIRE reduces algorithmic complexity and increases system safety. Compared with a standard *intervention-based* (or *interactive*) system (e.g., right side of Fig. 2.3), the failure prediction provided by FIRE reduces the burden on the expert.

Our system can predict when a running policy is likely to fail, halt its execution, and request a correction from the expert. Unlike existing approaches that learn only from expert data, our approach learns from both expert and non-expert data, akin to adversarial learning. We demonstrate experimentally for a series of manipulation tasks that our method is able to recognize state-action pairs that lead to failures. This permits seamless integration into an intervention-based learning system. We show an order-of-magnitude gain in sample efficiency compared with a state-of-the-art inverse reinforcement learning method and improved performance over an equivalent amount of data learned with behavioural cloning.

This chapter is associated with (Ablett et al., 2020), and includes details from that report and a previously unpublished expansion of the method to the multiview domain from Chapter 3. This chapter also includes a section on an unpublished improvement of the failure prediction method from (Ablett et al., 2020) based on Gaussian discriminant analysis.

5.1 Motivation

Imitation learning has proven to be an effective approach to overcome many of the limitations of reinforcement learning for robotic agents: it can significantly reduce the sample complexity of pure reinforcement learning (Sun et al., 2017) and it can eliminate the need for hand-designed rewards,

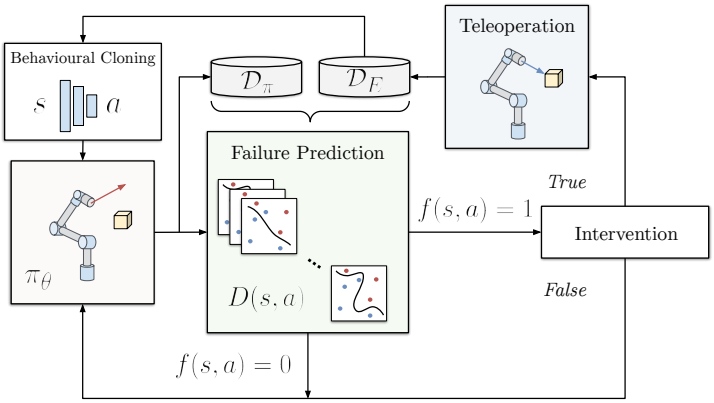


Figure 5.1: An overview of our proposed technique for learning policies through intervention-based learning with predicted failures. An initial policy is learned with a handful of demonstrations through behavioural cloning (top left), which the agent then executes. A failure predictor that uses a discriminator to classify expert and non-expert data predicts if and when a failure is likely to occur (center). When a failure is predicted ($f(s,a) = 1$), execution of the policy halts, and a human observer intervenes by either agreeing (True) or disagreeing (False) with the prediction. If they agree with the prediction, they teleoperate the robot to move it into a new state where they believe the existing policy can complete the task (top right). Periodically, the policy is retrained on both existing and new expert data (top left).

enabling agents to operate directly from observations without access to the state information often required by reward functions. In some cases, even behavioural cloning (BC), in which a model is trained directly on a dataset of expert state-action pairs, can achieve impressive results (Zhang et al., 2018; Pomerleau, 1989; Bojarski et al., 2016; Giusti et al., 2016). Unfortunately, in a phenomenon known as covariate shift, policies learned through BC will fail if they encounter data sufficiently outside the original expert distribution (Ross et al., 2011). In addition, policies learned with this paradigm may work in many cases, but may also fail in surprising ways (Zhang et al., 2018; Pomerleau, 1989), because they do not provide a means for *failure recognition*.

Covariate shift can be resolved by having an expert relabel every state that an agent encountered during execution in a process known as dataset aggregation, or DAgger (Ross et al., 2011). This solution and other related approaches are very effective when a programmatic expert is available to autonomously relabel states (Pan et al., 2018), but for many robotics tasks the only expert available is a human being. In such cases, except for trivially simple action spaces, it can be difficult or even impossible for a human to provide an adequate label: given the offline setting, the human has no feedback on whether the magnitudes of their action labels are correct. Existing work has shown that policies learned from offline human labels in a DAgger framework tend to be unstable (Kelly et al., 2019; Ross et al., 2013). Furthermore, in its standard form and with only a human expert available, DAgger requires a learned (or *novice*) policy to execute without intervention, meaning that the policy will reach many failure states or irrelevant states before a desirable policy is learned—this is problematic, and potentially catastrophic, for real robots interacting with the physical world.

Much of the prior work on failure recognition and prediction in imitation learning reduces to out-of-distribution detection (Menda et al., 2019; Laskey et al., 2016; Kelly et al., 2019; Kim and Pineau, 2013; Cui et al., 2019), that is, attempting to recognize state-action pairs that do not resemble those in the

expert dataset. These methods require setting a threshold for determining an allowable distance from the expert distribution, which can be nontrivial. A different method for predicting failures involves training a safety policy that attempts to predict whether an individual policy action deviates from the expert (Zhang and Cho, 2017). While this method is able empirically to reduce the expert relabelling burden in DAgger, it suffers from the limitation (noted in (Menda et al., 2019)) that it does not consider the policy's epistemic uncertainty—a single threshold may be too conservative in certain states, but too permissive in uncertain states.

We present **f**ailure **i**dentification to **r**educe **e**xpert burden (FIRE), a technique for attempting to resolve covariate shift and failure recognition simultaneously while learning a policy for completing a particular task. Compared to DAgger-based approaches for solving covariate shift, a human expert directly corrects the policy *during* execution and receives feedback on the effects of their actions immediately, making it possible for them to provide higher-quality labels than in DAgger with offline human labelling. Recent work has shown an online intervention-based approach can effectively learn a policy in self-driving car (Kelly et al., 2019) and drone landing (Goecks et al., 2019) domains.

Intervention-based learning suffers from high expert burden: an expert must be vigilant throughout execution of the novice policy for an indefinite number of trials. To address this, we incorporate failure prediction into our framework, so that when our system predicts a failure, motion can be halted and the human expert can add corrective data. Of course, this approach only works for tasks that are quasistatic, with relatively low speeds throughout the motion, but manipulation tasks that the community is interested in fall into this category. Our approach builds on adversarial inverse reinforcement learning methods that learn a discriminator between expert and non-expert data which doubles as a reward function (Ho and Ermon, 2016; Fu et al., 2018a). We use the learned discriminator and a threshold that is modified during policy execution based on human feedback to predict failures, effectively reducing the burden on the expert. Specifically, we make the following novel contributions:

1. We present a new method (FIRE) for predicting failures based on learning a discriminator and combining its output with a threshold value that is adjusted automatically based on human preferences during execution.

2. We demonstrate that a statistically grounded improvement based on Gaussian discriminant analysis (GDA) allows for significantly improved accuracy compared with competing methods.

3. We show that FIRE, an online-intervention-based approach with dataset aggregation, can learn effective policies in a variety of robotic manipulation domains, including a simulated view-agnostic domain for mobile manipulation policies, akin to Chapter 3. This learning occurs with significantly better sample complexity than a state-of-the-art inverse reinforcement learning-based technique and with greater stability than behavioural cloning.

4. We compare our failure prediction results with an existing method for identifying unsafe states.

5.2 Related Work

We use the notation for MDPs, imitation learning, and inverse reinforcement learning presented in Sections 2.2 and 2.4.

5.2.1 Behavioural Cloning and DAgger

A simple, but often very effective, approach for learning an agent given expert demonstrations is to simply treat $\mathcal{D}_E$ as a dataset and use supervised learning, often referred to as behavioural cloning (BC) (Pomerleau, 1989). We can model $\pi_\theta(a \mid s)$ using any function approximator and then learn its parameters θ by maximizing the likelihood, as shown in Eq. (2.32).

As previously discussed in Section 2.4.1, BC on its own will typically fail, in at least some cases, because the distribution of states encountered when executing π_θ does not necessarily match the distribution found in $\mathcal{D}_E$. This problem is resolved by DAgger (Ross et al., 2011), in which a BC-trained policy iteratively executes and relabels the states in the recorded dataset $\mathcal{D}_\pi$ with their corresponding expert actions. The new dataset is appended to the existing dataset, and the policy is re-trained. As stated, this strategy can be non-trivial to implement when the expert is a human. It is much easier for a human to provide labels by actually directly controlling the agent, allowing the (s, a) pairs to be recorded naturally as they occur.

5.2.2 Online, Intervention-Based Learning

To attempt to leverage the convergence properties of DAgger while allowing a human to provide high-quality labels, a human observer can predict when a policy will fail, intervene, return the agent to a state closer to the distribution $\mathcal{D}_E$, and then return control to π_θ. Strictly speaking, this approach is not the same as DAgger, since only the predicted failure states are relabelled with their corresponding expert actions, and the expert subsequently provides data to recover. This ensures that π_θ is not allowed to execute until failure, and does not require the human to provide labels offline.

Intervention-based approaches have been empirically shown to have varying degrees of success (Goecks et al., 2019; Cabi et al., 2020; Kelly et al., 2019). An obvious limitation of this approach is that it requires a vigilant human expert to perpetually watch the agent execute its policy, and be able to react quickly enough to take over control from the agent at appropriate times. This can be difficult due to an individual's inherent, delayed reaction time. For the general goal of finding an effective policy, the process may be exhausting for a human, since there is no guarantee of how many corrections the individual will have to provide before the policy reaches satisfactory performance.

5.2.3 Safe Intervention-Based Learning

Our work is most similar to (Kelly et al., 2019), in which the authors use expert feedback to find a threshold for determining when a policy may take an unsafe action. The authors of (Kelly et al., 2019) formulate their policy as an ensemble, following (Menda et al., 2019), which provides estimates of epistemic uncertainty for the policy actions. The mean of the measured uncertainty of state-action pairs

before human corrections is employed as the threshold for determining what degree of policy uncertainty is considered safe. In other words, policy uncertainty is considered to be inversely proportional to safety. The authors claim that their uncertainty method could be used to notify an expert when the agent has reached an unsafe state, but they do not actually incorporate this ability into their system.

Our method for predicting failures is similar to but distinct from another popular method for reducing expert burden (Zhang and Cho, 2017), in which a "safety" policy is learned that outputs whether an action is safe or not based on the mean squared error between it and an expert action. In (Zhang and Cho, 2017), an oracle policy must still be available to be queried arbitrarily to update the safety policy, however, as is the case in DAgger. We do not assume to have access to an oracle policy, making it impossible to use (Zhang and Cho, 2017) in our case.

5.2.4 Inverse Reinforcement Learning (IRL)

As originally discussed in Section 2.4.4, an alternative approach to imitation learning involves trying to learn a reward function $r(s, a) : S \times A \rightarrow \mathbb{R}$ that is maximized by π_E, as shown by Eq. (2.36). Our method for predicting failures is inspired by the discriminator trained in adversarial IRL methods, but we do not use the discriminator directly to improve our policy.

5.3 Failure Identification to Reduce Expert Burden (FIRE)

In Section 5.3.1 and Section 5.3.2 we present our original approach to intervention-based learning with failure identification and how we automatically adjust the prediction threshold. In Section 5.3.3, we describe an improvement to the original approach to FIRE based on time-conditional Gaussian discriminant analysis, and finally, in Section 5.3.4, we describe additional implementation details for FIRE, including how the policy is updated as new data is added by the expert. The full FIRE algorithm is shown in Algorithm 2.

5.3.1 Predicting Failures with a Discriminator

As shown in Algorithm 2, failures are predicted at every timestep of a trajectory when the human is not controlling the robot. As previously described in Section 2.4.4, a discriminator function $D(s_t, a_t)$ is a binary classifier which attempts to classify expert data from non-expert data. In other words, the output of $D(s_t, a_t)$ outputs the probability that (s_t, a_t) is from $\mathcal{D}_E$. Given $D(s_t, a_t)$, we can use the following rule to predict failures with a failure predictor $f : S \times A \rightarrow \{0, 1\}$, where $f(s, a) = 1$ indicates a failure:

$$f(s_t, a_t) = \begin{cases} D(s_t, a_t) < 0.5 & 1, \\ \text{else} & 0. \end{cases} \tag{5.1}$$

In this case, we have chosen 0.5 as the threshold point for choosing whether data should be classified as expert or not. While this may be reasonable in the general classification case, we will show in

Algorithm 2 Intervention-Based Learning with FIRE (GDA additions from Section 5.3.3 in teal)

Input: Expert dataset of state-action pairs $\mathcal{D}_E$, β, δ_{fn}, δ_{fp}, d, human observer H, environment Env, γ
Initialize D_w randomly.
Initialize policy $\pi_\theta \leftarrow$ *BehaviouralCloning*(π_θ, $\mathcal{D}_E$)
Initialize policy data $\mathcal{D}_\pi$ with initial episodes
Initialize `human_control` $\leftarrow 0$
Initialize $s_t \sim$ Env
while task performance is unsatisfactory **do**
  for $t = 1, \ldots, T$ **do**
    // 1. Get policy action, perform failure prediction, possibly get human action
    if not `human_control` **then**
      $a_t \leftarrow \pi(s_t)$
      Failure prediction $f(s_t, a_t) \leftarrow$ Eq. (5.2) or Eq. (5.5)
      if $f(s_t, a_t) = 1$ (predicted failure) **then**
        stop execution; query human: will agent fail?
        if yes **then**
          (*true positive*) `human_control` $\leftarrow 1$
        else
          (*false positive*) $\beta \leftarrow \beta + \delta_{\text{fp}}$ or $\gamma \leftarrow \gamma + \delta_{\text{fp}}$
        end if
      else if human predicts failure (*false negative*) **then**
        `human_control` $\leftarrow 1$
        $\beta \leftarrow \beta - \delta_{\text{fn}}$ or $\gamma \leftarrow \gamma - \delta_{\text{fn}}$
      end if
    end if
    if `human_control` **then**
      $a_t \leftarrow H(s_t)$
    end if

    // 2. Take step in environment, update corresponding dataset
    $s_{t+1} \leftarrow \text{Env}(s_t, a_t)$
    if `human_control` **then**
      $\mathcal{D}_E \leftarrow \mathcal{D}_E \cup (s_t, a_t, s_{t+1})$
      if human intervention complete **then**
        `human_control` $\leftarrow 0$
      end if
    else
      $\mathcal{D}_\pi \leftarrow \mathcal{D}_\pi \cup (s_t, a_t, s_{t+1})$
    end if
    $s_t \leftarrow s_{t+1}$
  end for
  Update D_w with Eq. (2.36) for T steps
  Update π_θ with T sampled batches from $\mathcal{D}_E$ and T gradient steps using Eq. (2.32)
  Update statistics for $p(D_t \mid \text{fail}) = \mathcal{N}(D_t \mid \mu_{\text{fail}}, \sigma_{\text{fail}})$ and $p(D_t \mid \text{success}) = \mathcal{N}(D_t \mid \mu_{\text{success}}, \sigma_{\text{success}})$
  if $|\mathcal{D}_E| \mod d = 0$ **then**
    $\pi_\theta \leftarrow$ *BehaviouralCloning*(π_θ, $\mathcal{D}_E$)
  end if
end while
Function BEHAVIOURALCLONING(π, $\mathcal{D}$)
  Train π with $\mathcal{D}$ and Eq. (2.32) by minimizing validation error
EndFunction

Section 5.3.3 that this choice may be highly suboptimal.

Individual values of $D(s_t, a_t)$ will fluctuate above and below 0.5 for data that is on the cusp of resembling expert data. To mitigate this uncertainty, we can also filter our failure prediction via a heuristic as

$$f(s_t, a_t) = \begin{cases} \{D(s_k, a_k) < 0.5\}_{k=t-\beta,\dots,t} & 1, \\ \text{else} & 0, \end{cases} \tag{5.2}$$

by adding a parameter β to only predict failures after β consecutive estimates below the threshold. The larger the value used for β, the less likely the system is to predict a failure.

Naturally, adding free parameters to any method is undesirable, and an ideal value for β would vary based on the environment, the tolerance for risk, the current accuracy of D, and the current performance of π_θ. In our case, similar to (Kelly et al., 2019), we use human feedback during execution to indirectly adjust the parameter, accounting for all of these factors simultaneously without forcing the human expert to directly supply β themselves. As is common in adversarial imitation learning approaches, we train our discriminator after every episode with a cross-entropy loss (Eq. (2.36)) by sampling random batches of data from $\mathcal{D}_\pi$ and $\mathcal{D}_E$ with a number of gradient steps equivalent to the episode length.

5.3.2 Automatic Adjustment of β

Adjustments to β are made based on three possible responses to the failure prediction system:

1. *Correct prediction (true positive)*: The human expert agrees with the prediction, meaning that they agree that the agent will fail if it continues to execute, in which case β remains the same.

2. *Incorrect prediction (false positive)*: The human expert disagrees with the prediction, meaning that the expert believes the agent will not fail if it continues to execute. β is increased by $\delta_{\text{fp}} \in \mathbb{N}^+$ to make the system more permissive.

3. *Missed prediction (false negative)*: The human expert manually stops the system when they believe a failure should have been predicted, but was not. β is decreased by $\delta_{\text{fn}} \in \mathbb{N}^+$ to make the system more conservative.

The modification hyperparameters δ_{fp} and δ_{fn} are easier to manually tune than β directly, and can be set based on an expert's own tolerance for false positives and false negatives (i.e., higher settings indicate a lower tolerance). While there are numerous other possibilities for determining how to modify β, we found that this simple scheme performed adequately. While the use of β can help in cases where $D(s_t, a_t)$ alone incorrectly classifies data, the use of 0.5 as a threshold in Eq. (5.1) can be quite suboptimal, which we will discuss in the foll section.

5.3.3 Gaussian Discriminant Analysis (GDA) for Failure Prediction

This section presents a modification to the prediction rule defined in Section 5.3.1 to use prior statistics and discriminant analysis. Ideally, raw discriminator outputs can be used to predict failures (e.g., predict a failure if $D(s, a) < 0.5$), but in practice, this is challenging for two reasons: (i) as D is being trained, and the relative size of $\mathcal{D}_\pi$ changes dramatically from one episode to the next, D values are noisy and inconsistent, and (ii) overall D values are strongly correlated with their frequency in $\mathcal{D}_E$ and $\mathcal{D}_\pi$. A concrete example of the second point occurs in many manipulation tasks, where the final, successful states of many trajectories are often quite similar to one another, while the initial state distribution, along with the trajectories leading to successful states, can be quite varied. For a further illustration of the second point, consider the average timestep-specific values of D shown in Fig. 5.6. For `PickAndPlaceXY` (shown in the middle plot), for both successful and unsuccessful trajectories, D is rarely above 0.5. Using Eq. (5.2), a failure could only be consistently correctly predicted with a very high value of β, since D values are, on average, below 0.5 for all early parts of trajectories.

If we assume to have access to labels for the true outcomes for each episode, and further assume that the timing of motions in episodes have a degree of between-episode consistency (e.g., the reach portion of a task happens consistently from timestep x to timestep y), we instead use two-class Gaussian discriminant analysis (GDA) on the outputs of D (Murphy, 2022), conditioned on each timestep. We periodically acquire statistics for D from past trajectories to obtain separate distributions at each timestep for success and failure of the task. In other words, given that a trajectory failed or succeeded, and assuming that D values at each timestep are normally distributed after being conditioned on success or failure, we generate per-timestep means and standard deviations for D_t data for both successful and failed trajectories.

We also maintain a prior estimate of failure using the final outcomes of all trajectories, which is simply $\frac{N_f}{N}$. In turn, we are able to use Bayes' rule to solve for the posterior $p(\text{fail} \mid D_t)$, where $D_t = D(s_t, a_t)$ is the discriminator output at time step t of any given episode.

$$p(\text{fail} \mid D_t) = \frac{p(D_t \mid \text{fail})p(\text{fail})}{p(D_t)} \tag{5.3}$$

$$= \frac{p(D_t \mid \text{fail})\frac{N_f}{N}}{p(D_t \mid \text{fail})\frac{N_f}{N} + p(D_t \mid \text{success})\frac{N_s}{N}}. \tag{5.4}$$

Here, $p(D_t \mid \text{fail}) = \mathcal{N}(D_t \mid \mu_{\text{fail}}, \sigma_{\text{fail}})$ and $p(D_t \mid \text{success}) = \mathcal{N}(D_t \mid \mu_{\text{success}}, \sigma_{\text{success}})$ are *class conditional densities*, the parameters for which are re-estimated after each episode based on all discriminator values at timestep t based on whether an episode ultimately failed or succeeded. Ultimately, our new failure prediction rule is

$$f(s_t, a_t) = \begin{cases} p(\text{fail} \mid D_t) < \gamma, D_t = D(s_t, a_t) & 1, \\ \text{else} & 0. \end{cases} \tag{5.5}$$

We update the threshold γ based on progressive user feedback, in the same fashion as we did with β in

Section 5.3.2. We can optionally choose whether or not to use the β parameter with the GDA scheme for predicting failures.

5.3.4 Intervention-Based Learning with Failure Prediction

As detailed in Algorithm 2, and as is done in (Goecks et al., 2019), although our policy is constantly updating with new samples generated from the expert, we retrain on the full batch of expert samples whenever d new expert samples are added by minimizing the loss of a random subset of validation expert data.

Since in our online intervention setting we have access to a human expert, we assume that we receive a binary sparse reward signal indicating whether each episode ends successfully. In practice, this is used to add (s_T, a_T) of a successful episode to $\mathcal{D}_E$, regardless of whether a_T was an expert action or not, to ensure that D always gives high value to successful states. We argue that this is a fair assumption, since most finite-horizon benchmark environments will output a "done" signal when a goal is reached.

To ensure that our initial model for D does not overfit, we allow our BC-initialized policy to complete a small number of episodes in the environment before we start execution of our method. This tactic is employed in (Fujimoto et al., 2018; Kostrikov et al., 2019), but because our policy is initialized with BC, the actions will not be random and it will be easier for a human to cut an episode short if the agent executes an unsafe action.

Given that our algorithm follows a reinforcement learning paradigm, and that we could easily learn Q in addition to learning D as is done in (Fujimoto et al., 2018; Kostrikov et al., 2019), a natural question is whether we could expand FIRE to include a self-improving reinforcement learning component, as is done in (Goecks et al., 2020; Jena et al., 2020; Rajeswaran et al., 2018). In each of these works, it was shown that attempting to learn a policy with an RL framework after initializing with BC either required careful tuning of weighting parameters for combined loss functions, or that initializing the policy with BC actually caused the final performance of the policy to deteriorate considerably (Jena et al., 2020). On top of that, attempting to learn using RL would require us to add exploration noise to our policy, decreasing the performance and potentially requiring even more expert interactions to learn a satisfactory policy. Owing to the varying results in existing literature, we choose to leave this extension as future work.

5.4 Fixed-Base Experiments

In our fixed-base experiments, we attempt to answer the following questions:

1. Does FIRE learn policies that can have strong performance with lower sample complexity than a state-of-the-art IRL-based method and with greater robustness than BC?

2. How accurately does our system predict failures? Does our failure prediction accuracy improve as more data is added?

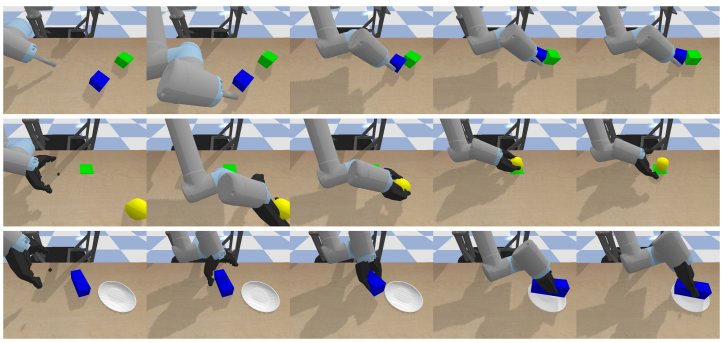


Figure 5.2: Examples of successful episodes in each of our original, fixed-base, environments, `PushingXY`, `PickAndPlaceXY`, and `PickAndPlace6Dof`, showing frames from $t = \{1, .25T, .5T, .75T, T\}$.

3. How does the failure prediction performance of FIRE compare to an existing, alternative method?

4. Is there empirical justification for this method of predicting failures? Do successful episodes tend to result in $D \geq 0.5$, while failures result in $D < 0.5$?

5.4.1 Environment Details

To demonstrate the efficacy of our method, we present experiments where we attempt to learn high-performing policies (measured by success rate) in a variety of challenging manipulation environments (see Fig. 5.2) described below:

1. `PushingXY`: A pushing environment, in which a narrow, cylindrical end-effector must push a block to be close to another block. The location and rotation of the first block are randomized between trials to positions within a 25 cm x 10 cm rectangle.

2. `PickAndPlaceXY`: A pick-and-place environment, in which a two-fingered gripper must grab a large cylinder and place it on a coaster. The location of the cylinder is randomized between trials to positions within a 25 cm x 10 cm rectangle.

3. `PickAndPlace6Dof`: A six degrees-of-freedom pick-and-place environment, where the end-effector can translate and rotate freely, and must grab a long block and place it on a plate. The location of the center of the block is randomized between trials to positions within a 5 cm x 5 cm square, while the rotation is randomized to all possible orientations. The gripper must rotate, often significantly, from its initial position to reach an orientation where it is possible for it to grab the block (see Fig. 5.2).

Environments 1 and 2 are constrained to only allow movement in two dimensions. As shown in Fig. 5.4, a state-of-the-art IRL method (Kostrikov et al., 2019) still struggles to solve even these two-dimensional environments adequately, failing altogether in the pick and place environment. The expert controls the agent in Environment 3 with an HTC Vive virtual reality hand tracker. Although we do not have access to a dense reward, we do have a binary success signal that determines if, for example,

the block is on the plate. The states in each of these environments consist of the poses of the task objects and the end-effector as well as the gripper link positions, while the actions are the end-effector velocity and, for the pick and place environments, a binary signal indicating whether to open or close the gripper. Our environments are simulated in PyBullet (Coumans and Bai, 2019) with a UR10 arm and a two-fingered gripper. A timestep in each environment is 0.1 s, while, at the level of simulation, each action is repeated 10 times, as is common in robotics learning environments (Plappert et al., 2018). Since we are exclusively dealing with the finite-horizon problem, for environments 1, 2, and 3, T is 70, 70, and 80 respectively.

5.4.2 Implementation Details

We use the discriminator gradient-penalty trick from (Gulrajani et al., 2017) to learn a discriminator D that avoids overfitting. We set our network and learning hyperparameters the same as in (Kostrikov et al., 2019): a 2-layer MLP policy with 256 hidden units, ReLU activations and a `tanh` acvitation for the output, and a 2-layer MLP discriminator with 256 hidden units and `tanh` activations. We train using the Adam optimizer (Kingma and Ba, 2015) with learning rates of 10^{-3} and 10^{-4} for the policy during behavioural cloning and episode training respectively. We set our periodic full batch retraining parameter d to 500, and we allow our initially BC-trained policy to collect 10,000 (s, a) pairs (or roughly 125 episodes, 15 minutes of wall clock time) before we start training D. When we rerun BC on all of our expert data, we always warm-start with the existing policy since we empirically found this to give better results. Intuitively, we expect that this is because the corrections that the human expert provides specifically address mistakes made by a particular policy, and restarting from scratch wastes this advantage.

In our experiments, we set δ_{fn} as 3 and δ_{fp} to be 1 because we consider the cost of a false negative to be higher than that of a false positive. A false negative causes the agent to fail or requires the human expert to react quickly themselves, while a false positive merely requires the expert to allow execution to continue, with no requirement for a fast reaction. For each run of FIRE, we initialize β to 20, and it quickly self-adjusts to the expert's preferences. The pushing and 3D pick-and-place environments are initialized with 50 expert trajectories, while the 2D pick-and-place environment is initialized with 15 expert trajectories.

5.4.3 Evaluating Policy Performance

In this section, we evaluate the performance of policies trained using FIRE. Because our method is inherently based on human decision-making, we validate the efficacy of our method by completing experiments in a variety of environments requiring different manipulation capabilities. Given the previously described difficulty of generating offline human labels in continuous domains, we do not attempt to benchmark against DAgger. In particular, we expect that it would be nearly impossible to supply human-generated offline expert labels to the six degrees-of-freedom environment.

We first compare FIRE with BC alone. Since FIRE requires the collection of progressively more expert data, a natural question is whether an expert could simply provide more demonstrations initially

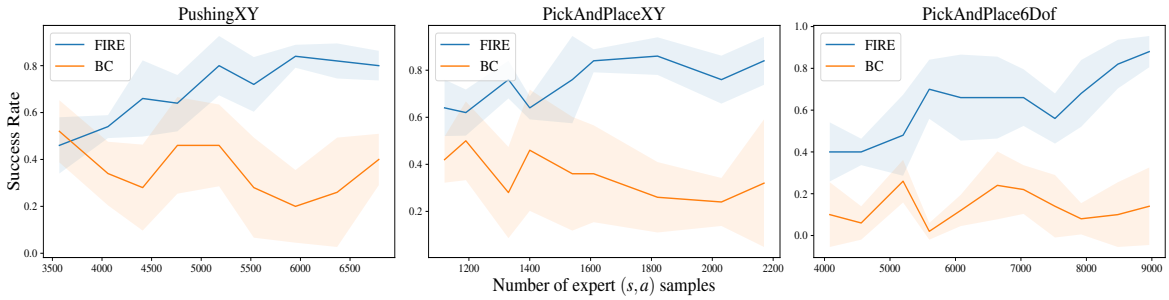


Figure 5.3: Performance of FIRE and BC in each of our test environments, compared with total number of expert (s, a) samples. Due to the suboptimality of human demonstrations, behavioural cloning has high variance, and adding more data does not necessarily improve performance. As well, the corrective expert data we collect throughout the execution of FIRE ensures that we address the distribution drift that occurs during standard BC, without requiring the collection of costly offline labels, as in (Ross et al., 2011).

and achieve the same performance as that found in FIRE. To answer this question, in Fig. 5.3, we compare the performance of FIRE with the performance of BC on an equivalent amount of data. The BC data are all collected from full episode rollouts, without any interactive component. The policies learned with data collected from FIRE clearly show substantial improvement over policies trained with behavioural cloning alone.

To illustrate the sample efficiency in terms of total execution time, we also compare against a state-of-the-art sample-efficient inverse reinforcement learning method (Kostrikov et al., 2019) in Fig. 5.4. It is clear that FIRE learns with considerably greater sample efficiency than (Kostrikov et al., 2019), and performs better than BC alone after only a small number of iterations. Our algorithm allows us to learn policies that complete each task with a success rate of over 80% with 50 minutes or less of wall clock execution time. We stopped execution of (Kostrikov et al., 2019) after it had converged (in the pushingXY environment) or after it was sufficiently clear that its sample efficiency, were it to converge, would be at least an order-of-magnitude worse than FIRE.

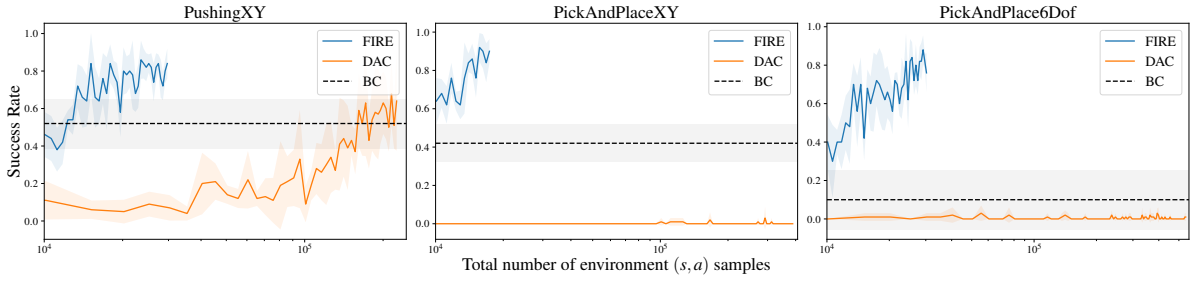


Figure 5.4: Performance of FIRE in our test environments, compared with the total number of environment samples. FIRE, Discriminator Actor Critic (DAC) (Kostrikov et al., 2019), and BC were all started with the same number of expert trajectories. Of course, in FIRE, more expert data was added throughout execution. The success rate of DAC starts off significantly lower than BC or FIRE because naively initializing RL-based methods with BC has been shown to be detrimental to learning (see end of Section 5.3.4).

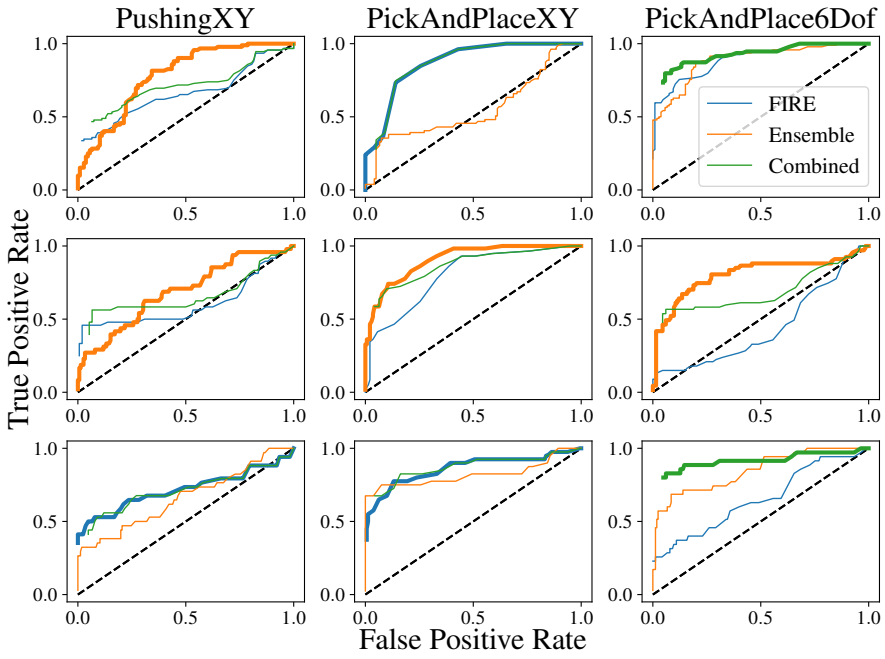


Figure 5.5: ROC curves for each environment throughout the execution of FIRE, compared with ensemble-based uncertainty (Kelly et al., 2019; Menda et al., 2019) and a method combining both. Columns correspond to environments, while rows correspond to the percentage of time between when we started and when we stopped executing FIRE, with the first row being 20%, the second being 50%, and the final being 70%. The method with the largest area under the curve is bolded in each case.

5.4.4 Evaluating Failure Prediction Performance

In addition to showing that FIRE is able to learn good policies, it is important to know if the failure prediction system is actually capable of predicting failures. To analyze our system's capability in this regard, we consider failure prediction as a binary classification problem, which allows us to use standard statistical measures. We show receiver operating characteristic (ROC) curves (Fig. 5.5) for our learned D at several points during the running of FIRE.

In existing similar work (Kelly et al., 2019; Cui et al., 2019), the authors have chosen to evaluate their binary predictor based on whether it correctly predicts infractions, which they define as hitting

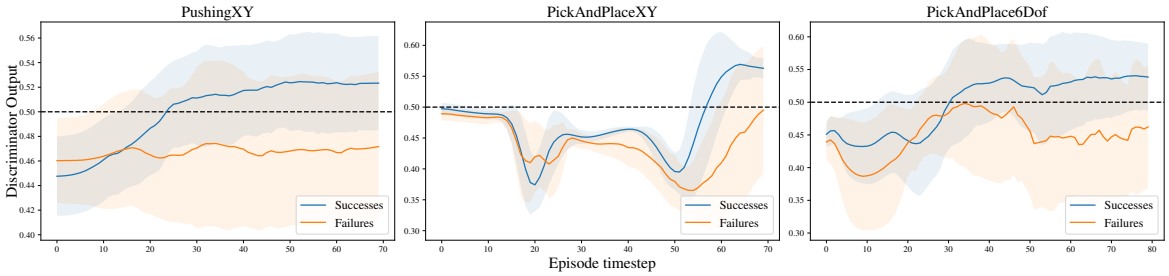


Figure 5.6: The average estimate from the discriminator for 200 episodes sampled using a fixed policy and discriminator approximately 70% of the way through the execution of FIRE. The average values for the successful and unsuccessful episodes are separated.

another car or driving off the road. Unfortunately, the concept of an infraction is not so cut-and-dry in manipulation environments. To generate the ROC curves, we used snapshots of our policy weights at various stages of executing FIRE and ran them for 200 episodes, allowing the episodes to fully run to failure or success. From these executions, along with the associated value of D for all steps of each episode, we generated true positive rates and false positive rates for different values of β, where we define a true positive to be a correct failure prediction.

It is clear from Fig. 5.5 that in the majority of cases, FIRE can predict failures at a level above random, though it is stronger in some cases than in others. To answer our original question, it is not clear whether our failure prediction improves as more data is added, as the results are not consistent between environments.

5.4.5 Comparing Failure Prediction Performance

To generate the curves based on the method from (Kelly et al., 2019; Menda et al., 2019), we trained an ensemble of 10 policies on all of the expert data up to various points during the execution of FIRE. Also, we used the ℓ_2-norm doubt metric $d(s_t) = \|\text{diag}(C_t)\|_2$, where C_t is the current covariance matrix of the output of the ensemble. We used the same 200 episode rollouts that we used for evaluating FIRE. We consider a prediction to be a true positive whenever $d(s_t)$ is greater than the test value of the parameter at any point during an unsuccessful trajectory, and a true negative when $d(s_t)$ is less than the test value for an entire successful trajectory.

It is clear from Fig. 5.5 that there are some cases where our method for predicting failures performs best, and some cases where the ensemble-based doubt metric performs best. However, in general, combining the two methods tends to provide at least reasonable performance even if the ensemble metric performs the best, and performs either the best or nearly equivalent to the FIRE-only metric in other cases. We acknowledge that our technique for combining the two methods is quite basic: we simply take the logical OR of the two predictions at every timestep. We expect that a more intelligent combination of the two methods would provide the best performance overall.

5.4.6 Further Analysis

In Fig. 5.6, we show the average predicted D values for 200 full episodes of each environment using the D and π_θ fixed from about 70% of the way through the execution of FIRE. We separate the curves into averages for successful and unsuccessful episodes. By setting β to a number of timesteps where D is predicted to be, on average, consistently above 0.5 for successful episodes and consistently below 0.5 for failed episodes, we can predict failures correctly at least some of the time. For example, for the `PushingXY`, `PickAndPlaceXY`, and `PickAndPlace6Dof` environments respectively, we can expect that values of $\beta \geq 30$, $\beta \geq 35$, and $\beta \geq 50$ should predict failures correctly more often than not. However, a clear limitation of this approach is that, should the agent begin to fail significantly before the timestep where it tends to fail most often, the predictor, limited by its higher value of β, will show a significant lag in predicting the failure. We acknowledge that setting the cutoff for D to 0.5, while intuitive, is somewhat arbitrary. In this work, we chose to demonstrate that the values of D *can* be

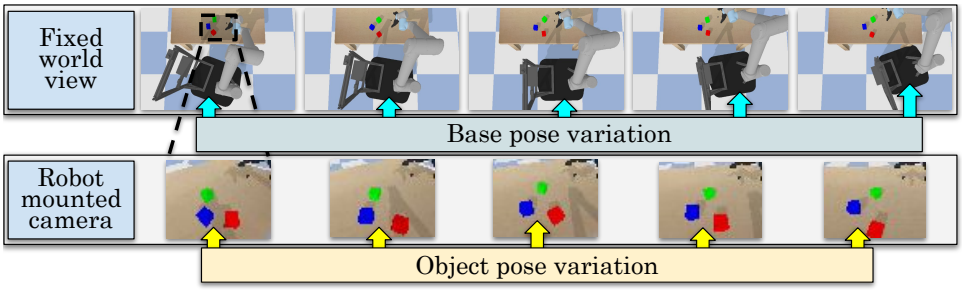


Figure 5.7: Example initial conditions for our `Stack2` task, showing both base pose variation and object pose variation at the beginning of episodes.

used to predict failures, even with a very simple decision rule. It is also worth noting that a value of β that approaches the horizon length T will become less and less useful. Our scheme, which allows the human expert to adjust β, indirectly ensures that β will only approach the horizon length once the policy is succeeding almost all of the time.

5.5 Multiview Experiments with GDA-FIRE

In followup work, we completed more experiments with FIRE in multiview environments, akin to the challenging environments originally presented in Chapter 3, and we also tested the Gaussian discriminant analysis (GDA) approach to predicting failures with FIRE. Specifically, compared with the environments from Section 5.4, these environments (i) vary the base pose at the beginning of each episode, and (ii) use 64×48 RGB image observations, instead of numerical state data. Our environments (see Fig. 5.7 and Fig. 5.8) are described below:

1. `PickAndInsert`: The manipulator must grasp a peg and insert it into a block with a hole within 12 seconds. The peg starts randomly in a 2.5 cm $\times$ 2.5 cm box, while the block's position is the same between episodes. The diameters of the peg and hole are 3.7 cm and 4 cm respectively.

2. `Stack2`: The manipulator must stack two blocks on top of a third to make a stack of three blocks within 15 seconds. The centers of all three blocks start in random positions and orientations in separate 5 cm $\times$ 5 cm boxes.

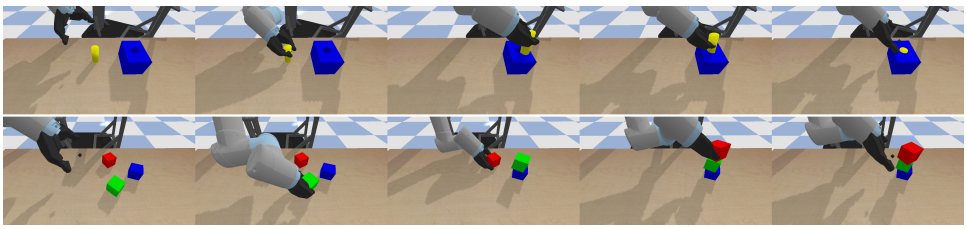


Figure 5.8: Examples of successful episodes in each of our multiview environments, `PickAndInsert` (originally presented in Chapter 3) and `Stack2`, showing frames from $t = \{1, .25T, .5T, .75T, T\}$.

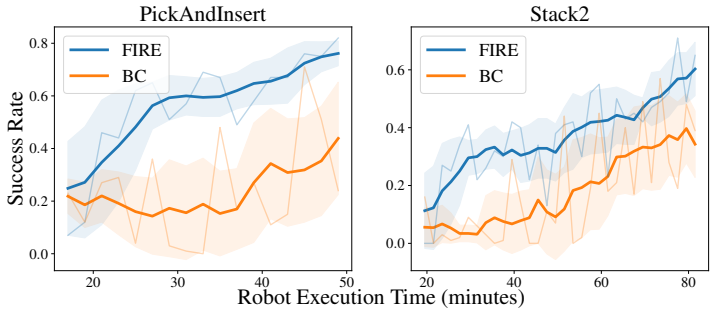


Figure 5.9: A comparison of our method (FIRE: Failure Identification for Reducing Expert burden) with behavioural cloning (BC) for equivalent amounts of robot execution time. The bolded line and surrounding fill are a 7-step moving average and standard deviation. Notably, the results compare total execution time, as opposed to the results from Fig. 5.3, which compare expert data quantity only.

5.5.1 Performance Results

In Fig. 5.9, we benchmark our method's performance by comparing its success rate against behavioural cloning (BC) for an equivalent number of executed episodes: demonstrations in BC, and a combination of demonstrations, corrections, and autonomous execution in our method. We compare against BC since, to the best of our knowledge, BC continues to be the most sample-efficient imitation learning method when one cannot query π_E. In particular, state-of-the-art methods based on inverse reinforcement learning still require hundreds of thousands, if not millions, of environment interactions before learning a good policy in manipulation environments (Kostrikov et al., 2019; Cabi et al., 2020), while our policies are learned with less than 50,000 environment interactions corresponding to less than 1.5 hours of robot execution time. With an equivalent amount of time, our method clearly performs better than BC in both of our tasks.

It is important to note that, compared with the results from Section 5.4.3 and Fig. 5.3, these results appear to show BC scaling well as data increases. The results from Fig. 5.3 only compare the actual number of expert examples used, and because many episodes of FIRE add no or only few new expert samples to the dataset, it can appear to be far more efficient than BC when comparing samples only. Comparing total execution time is, in most cases, a more useful baseline, since it signals to a practitioner how useful it is to collect intervention-based data, as opposed to simply collecting more full trajectories.

5.5.2 Failure Prediction Results

In Fig. 5.10, we compare our failure prediction method against the method shown in Section 5.3.1 and Section 5.4, as well as a method based on using the uncertainty provided by ensemble variance (Menda et al., 2019). We generated 100 episodes with π_θ using snapshots from two evaluation points in our training for both environments. We created receiver operating characteristic (ROC) curves by varying the threshold of each method for predicting failures, and comparing that threshold with the maximum output value from each episode. Our method based on discriminant analysis (DA) consistently has the best performance for both tasks, and particularly, strongly improved the performance of using a

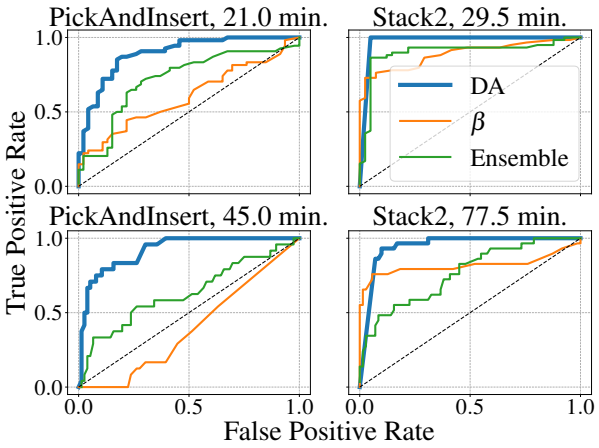


Figure 5.10: ROC curves showing the accuracy of our Discriminant Analysis(DA)-based failure predictor compared with the original β threshold from Section 5.3.1 and with ensemble doubt (Menda et al., 2019) at different points during training for both environments.

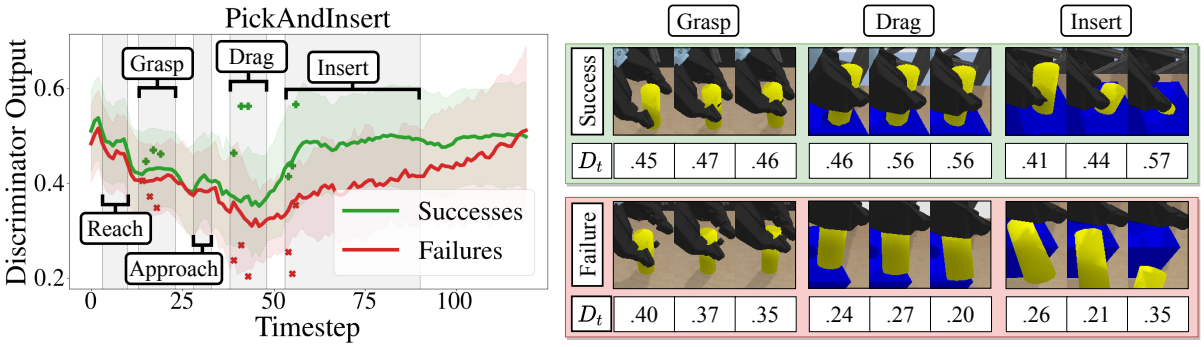


Figure 5.11: Left: Average and standard deviation of discriminator at each timestep of an episode, conditioned on successful episodes and failure episodes separately, used for predicting failures across 100 episodes in `PickAndInsert`. Individual approximate timing of sub-tasks (reach, grasp, approach, drag, insert) are also labelled, although these are not enforced in any way. **Right:** Examples of frames from successful and failed trajectories. Their corresponding D_t values, underneath each image, are also shown in the left plot.

discriminator compared to the results shown in Fig. 5.5.

5.5.3 Quantitative Analysis of GDA-FIRE

Fig. 5.11 shows a quantitative example of why Gaussian discriminate analysis (GDA) (see Section 5.3.3 for description) might be more useful for failure prediction than raw discriminator outputs alone. The mean and standard deviation for each timestep are shown for estimates of $p(D_t|\text{fail})$ and $p(D_t|\text{success})$ in red and green, respectively. Clearly, a fixed D value is unlikely to be accurate for predicting failures: the initial states of the trajectory, for both successful and failed episodes, tend to have the highest D values overall, meaning that a fixed D value would most likely produce many false negatives or false positives. Instead, by maintaining statistics of D values at each timestep, a general pattern of D values becomes clear, and there is a gap between the average for $p(D_t|\text{success})$ and $p(D_t|\text{fail})$. The right side of Fig. 5.11 shows examples of both successes and failures and their corresponding D values, showing

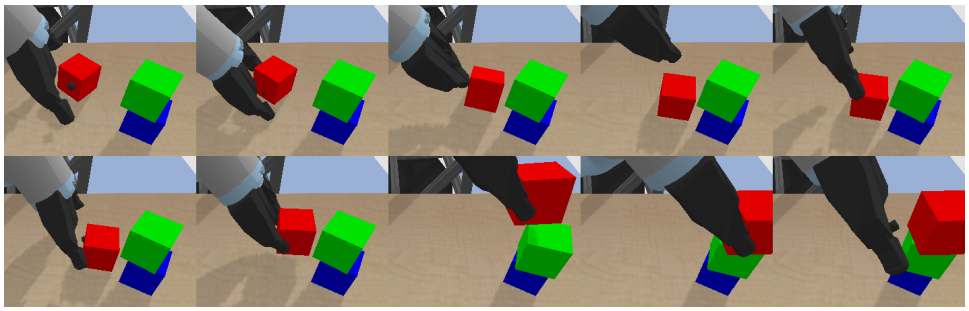


Figure 5.12: An example `Stack2` trajectory showing that our policy learns to recover from errors. The robot only successfully grasps and stacks the red block on its second attempt. Trajectory proceeds left to right, top to bottom.

that D values for failure or success can vary quite substantially, again indicating that GDA may be a more useful approach than raw D values for predicting failures.

5.5.4 Learning to Fix Errors

A major benefit of our method compared with pure behavioural cloning (BC) is the ability for our policies to learn strategies for recovering from potential failures (see Fig. 5.12). Of course, an ideal policy would be optimal and not make any mistakes, but a policy that can recover from potential failures will be more robust in the face of adversarial inputs. This robustness shows that an intervention-based framework can achieve the same consistent increase in performance as DAgger but without requiring costly offline relabelling. On top of that, our method outperforms BC significantly (with an equivalent amount of data) on two challenging view-invariant manipulation tasks, demonstrating that interventions with failure prediction can assist users in providing higher-quality data to a learning-based manipulation system.

5.6 Limitations

In this section, we discuss some limitations of our work. We only train policies in simulation; real-world conditions that may be affected by the presence of an expert demonstrator, such as shadows or errors due to poor views of the system, may cause performance to deteriorate. Furthermore, our system inherently adds data in which the robot has failed or nearly failed to an expert dataset. By adding the failed data, we can inadvertently end up training suboptimal policies that make mistakes, such as that shown in Fig. 5.12. Finally, predicting failures in manipulation is potentially an ill-defined task. A policy may make many slightly suboptimal movements before reaching an unrecoverable failure, or it might still succeed. An intervention-based system will always need to be tuned to individual user preferences to decide what level of suboptimality is acceptable.

5.7 Summary

In this chapter, we presented failure identification for reducing expert burden (FIRE), a method for learning a policy from expert demonstrations followed by expert interventions based on failure prediction. Our method predicts failures with a discriminator, and optionally with Gaussian discriminant analysis, and an adjustable threshold indicating the tolerable number of non-expert (s, a) pairs in a row. We showed that this technique can be used to learn high-performance policies in several challenging manipulation environments with an order of magnitude better sample efficiency than a state-of-the-art inverse reinforcement learning method. As well, policies learned using our method exhibit significantly better final performance than then those learned using an equivalent amount of data and pure behavioural cloning.

To allow policies learned from data and stored in deep neural networks to execute in the real world, we need to better understand when and if such policies are going to fail. Most state-of-the-art methods for failure prediction are based on uncertainty estimated from ensembles or pseudo-ensembles. As we showed in this work, these methods can be partially effective, but are limited by the fact that they do not directly learn from non-expert data.

A fruitful direction for future research would be to attempt to formally prove that on-the-fly interventions benefit policy performance, since at the moment, this approach is only justified empirically. It is possible that following the analysis from Ross and Bagnell (2010); Ross et al. (2011) could help in this regard; FIRE is meant to be a more realistic alternative to DAgger and other schemes (e.g., (Laskey et al., 2016)) that relabel offline data with expert actions. The corrective demonstrations provided in an interactive scheme, such as FIRE, will eventually reduce worst case error to $\mathcal{O}(T\epsilon)$ by removing distribution shift, as described in Section 2.4.3 and by Ross et al. (2011). Practically, however, the number of interactive demonstrations required to achieve this goal will likely be higher than $\mathcal{O}(T \log T)$ (one estimate of the amount required for DAgger by Ross et al. (2011)), because corrective demonstrations provided by interventions will be strictly of equal or lower quality than the perfect labels provided by DAgger.

Chapter 6

Learning from Guided Play

In practice, the interactive approach from Chapter 5 can still be costly to implement, requiring an unknown amount of expert time and data to acquire a proficient policy. In this chapter, we switch from approaches to imitation learning based on behavioural cloning to an approach based on inverse reinforcement learning (IRL).[1] In IRL, the expert data is provided once at the beginning of training, and, in principle, the agent learns to stay in-distribution through its own exploration and distribution matching. In practice, modern approaches to IRL can suffer poor exploration coupled with deceptive rewards, leading to a local maximum where the distribution visited by the agent only partially matches the expert distribution, and ultimately fails to complete tasks.

We present Learning from Guided Play (LfGP), a framework in which we leverage expert demonstrations of multiple exploratory, auxiliary tasks in addition to a main task. The addition of these auxiliary tasks forces the agent to explore states and actions that standard adversarial imitation learning (AIL) methods may learn to ignore, allowing for full matching between the expert distribution and the policy distribution. In addition, this particular formulation allows for the reusability of expert data between main tasks. Our experimental results in a challenging multitask robotic manipulation domain indicate that LfGP significantly outperforms both AIL and behavioural cloning, while also being more expert sample efficient than these baselines. To explain this performance gap, we provide further analysis of a toy problem that highlights the coupling between a local maximum and poor exploration, and also visualize the differences between the learned models from AIL and LfGP.

6.1 Motivation

Exploration is a crucial part of effective reinforcement learning (RL). A variety of methods have attempted to optimize the exploration-exploitation trade-off of RL agents (Sutton and Barto, 2018; Bellemare et al., 2016; Nair et al., 2018), but the development of a technique that generalizes across domains remains an open research problem. A simple, well-known approach to reduce the need for random exploration is to provide a dense, or "shaped," reward to learn from, but this can be very challenging to

[1] Project website: https://papers.starslab.ca/lfgp.

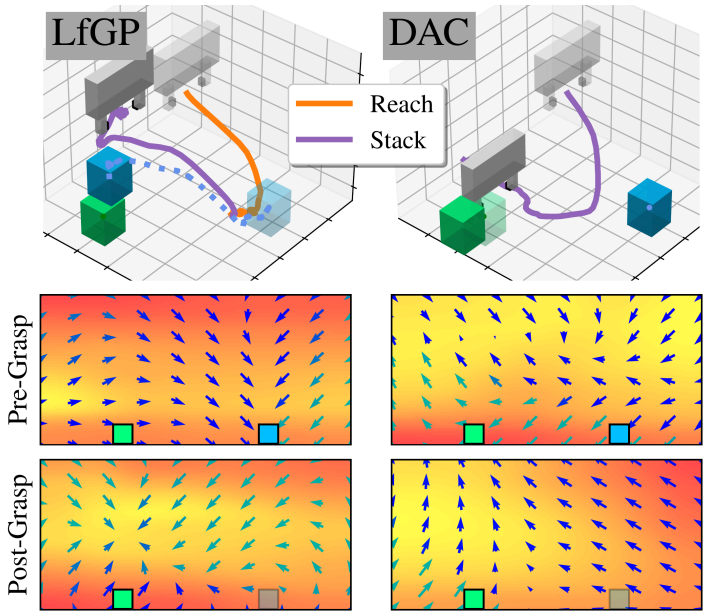


Figure 6.1: Learning from Guided Play (LfGP) finds an effective stacking policy by learning to compose multiple simple auxiliary tasks (only Reach is shown, for this episode) along with stacking. Discriminator Actor-Critic (DAC) (Kostrikov et al., 2019), or off-policy AIL, reaches a local maximum action-value function and policy, failing to solve the task. Arrow direction indicates mean policy velocity action, red-to-yellow (background) indicates low-to-high learned value, while arrow colour indicates probability of closing (green) or opening (blue) the gripper.

design appropriately (Ng and Jordan, 2003). Furthermore, the environment may not directly provide the low-level state information required for such a reward. An alternative to providing a dense reward is to learn a reward function from expert demonstrations of a task, in a process known as inverse RL (IRL) (Ng and Russell, 2000). Many modern approaches to IRL are part of the adversarial imitation learning (AIL) family (Ho and Ermon, 2016). In AIL, rather than learning a reward function directly, the policy and a learned discriminator form a two-player min-max optimization problem, where the policy aims to confuse the discriminator by producing expert-like data, while the discriminator attempts to classify expert and non-expert data.

Although AIL has been shown to be more *expert* sample efficient than behavioural cloning in continuous-control environments (Ho and Ermon, 2016; Fu et al., 2018a; Kostrikov et al., 2019), its application to long-horizon robotic manipulation tasks with a wide distribution of possible initial configurations remains challenging (Kostrikov et al., 2019; Orsini et al., 2021).

In this chapter, we investigate the use of AIL in a multitask robotic manipulation domain. We find that a state-of-the-art AIL method, in which off-policy learning is used to maximize *environment* sample efficiency (Kostrikov et al., 2019) (i.e., reduce the quantity of environment interaction required from the online RL portion of AIL), is outperformed by BC with an equivalent amount of expert data, contradicting previous results (Ho and Ermon, 2016; Fu et al., 2018a; Kostrikov et al., 2019). Through a simplified example, simulated robotic experiments, and learned model analysis, we show that this outcome occurs because a model learned with expert data and a discriminator is susceptible to the

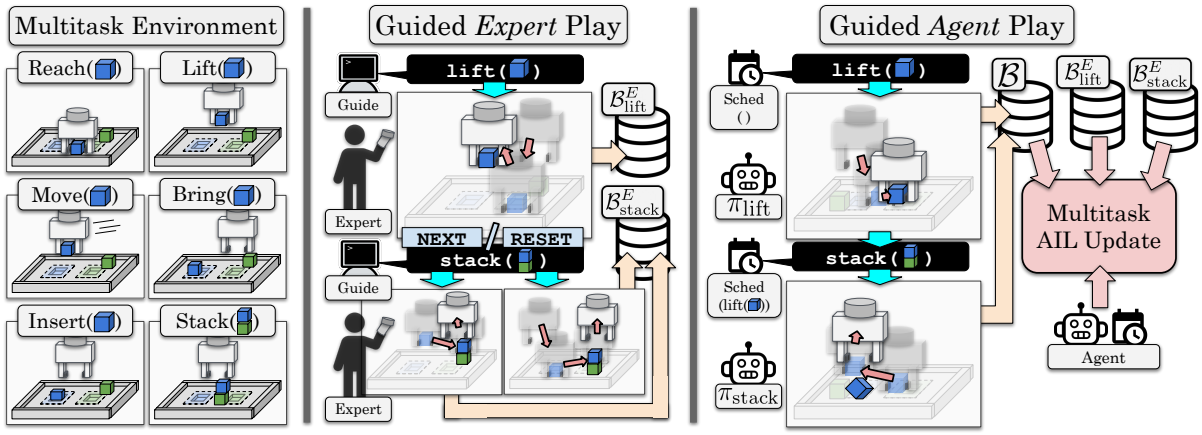


Figure 6.2: The main components of our system for learning from guided play. In a multitask environment, a guide prompts an expert for a mix of multitask demonstrations, after which we learn a multitask policy through scheduled hierarchical AIL.

deceptive reward problem (Ecoffet et al., 2021). In other words, while AIL, and more generally IRL, can provide something akin to a dense reward, this reward is not necessarily optimal for teaching, and AIL alone does not enforce sufficiently diverse exploration to escape locally optimal but globally poor models. A locally optimal policy has converged to match a subset of the expert data, but in doing so, avoids crucial states and actions (e.g., in Fig. 6.1, grasping the blue block) required to globally match the full expert set.

To overcome this limitation of AIL, we present Learning from Guided Play (LfGP), in which we combine AIL with a scheduled approach to hierarchical RL (HRL) (Riedmiller et al., 2018), allowing an agent to 'play' in the environment with an expert guide. Using expert demonstrations of multiple relevant auxiliary tasks (e.g., Reach, Lift, Move-Object), along with a main task (e.g., Stack, Bring, Insert), our scheduled hierarchical agent is able to learn tasks where AIL alone fails. The use of a multitask agent affords other benefits as well: expert data for auxiliary tasks can be reused from one main task to another, and the possibility of reusing previously-learned auxiliary models for transfer learning is opened. Crucially, our formulation also allows auxiliary expert data to be reused between main tasks, further emphasizing the expert sample efficiency of our method.

We use the word *play* to describe an agent that simultaneously attempts and learns numerous tasks at once, freely composing them together, inspired by the playful (as opposed to goal-directed) phase of learning experienced by children (Riedmiller et al., 2018). In our case, *guided* represents two separate but related ideas: first, that the expert guides this play, as opposed to requiring hand-crafted sparse rewards as in (Riedmiller et al., 2018) (right side of Fig. 6.2), and second, that the expert gathering of multitask, semi-structured demonstrations is *guided* by uniform-random task selection (middle of Fig. 6.2), rather than requiring the expert to choose transitions between goals, as in (Lynch et al., 2019; Gupta et al., 2019). Our specific contributions are the following:

1. A novel application of a hierarchical framework (Riedmiller et al., 2018) to AIL that learns a reward and policy for a challenging main task by simultaneously learning rewards and policies for auxiliary tasks.

2. Manipulation experiments in which we demonstrate that AIL fails, while LfGP significantly outperforms both AIL and BC.

3. An extension of our method to transfer learning.

4. A thorough ablation study to examine the effects of various design choices for LfGP and our baselines.

5. Empirical analysis, including a simplified representative example and visualization of the learned models of LfGP and AIL, to better understand why AIL fails and how LfGP improves upon it.

6.2 Related Work

Imitation learning is often divided into two main categories: behavioural cloning (BC) (Ross et al., 2011; Ablett et al., 2021b) and inverse reinforcement learning (IRL) (Ng and Russell, 2000; Abbeel and Ng, 2004). BC recovers the expert policy via supervised learning, but it suffers from compounding errors due to covariate shift (Ross et al., 2011; Ablett et al., 2020). Alternatively, IRL partially alleviates the covariate shift problem by estimating the reward function and then applying RL using the learned reward. A popular approach to IRL is adversarial imitation learning (AIL) (Ho and Ermon, 2016; Kostrikov et al., 2019; Hausman et al., 2017), in which the expert policy is recovered by matching the occupancy measure between the generated data and the demonstration data. Our proposed method enhances existing AIL algorithms by enabling exploration of key auxiliary tasks via the use of a scheduled multitask model, simultaneously resolving the susceptibility of AIL to deceptive rewards.

Agents learned via hierarchical reinforcement learning (HRL), which act over multiple levels of temporal abstractions in long-horizon tasks, are shown to provide more effective exploration than agents operating over only a single level of abstraction (Riedmiller et al., 2018; Sutton et al., 1999; Nachum et al., 2019). Our approach for learning agents most closely resembles hierarchical AIL methods that attempt to combine AIL with HRL (Hausman et al., 2017; Henderson et al., 2018; Sharma et al., 2019; Jing et al., 2021). Existing work (Henderson et al., 2018; Sharma et al., 2019; Jing et al., 2021) often formulates the hierarchical agent using the Options framework (Sutton et al., 1999) and learns the reward function with AIL (Ho and Ermon, 2016). Both (Henderson et al., 2018) and (Jing et al., 2021) leverage task-specific expert demonstrations to learn options using mixture-of-experts and expectation-maximization strategies, respectively. In contrast, our work focuses on expert demonstrations that include multiple reusable auxiliary tasks, each of which has clear semantic meaning.

In the multitask setting, (Hausman et al., 2017) and (Sharma et al., 2019) leverage unsegmented, multitask expert demonstrations to learn low-level policies via a latent variable model. Other work has used a large corpus of unsegmented but semantically meaningful "play" expert data to bootstrap policy learning (Lynch et al., 2019; Gupta et al., 2019). We define our expert dataset as being derived from *guided* play, in that the expert completes semantically meaningful auxiliary tasks with provided transitions, reducing the burden on the expert to generate these data arbitrarily and simultaneously providing auxiliary task labels. Compared with learning from unsegmented demonstrations, the use of

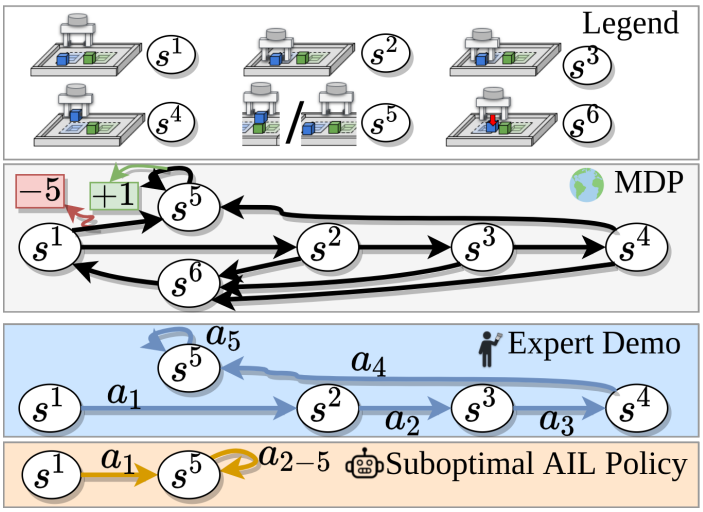


Figure 6.3: An MDP, analogous to stacking, with an expert demonstration. Poor exploration can lead AIL to learn a suboptimal policy.

segmented demonstrations, as in (Codevilla et al., 2018), ensures that we know which auxiliary tasks our model will be learning, and opens up the possibility of expert data reuse and also transfer learning. Finally, we deviate from the Options framework and build upon Scheduled Auxiliary Control (SAC-X) to train our hierarchical agent, since SAC-X has been shown to work well for challenging manipulation tasks (Riedmiller et al., 2018).

6.3 Problem Formulation

For a background on Markov decision processes (MDPs) and the notation used in this section, see Section 2.2. In this chapter, to accommodate hierarchical learning, we augment an MDP $\mathcal{M}$ with auxiliary tasks, where $\mathcal{T}_{\text{aux}} = \{\mathcal{T}_1, \ldots, \mathcal{T}_K\}$ are separate MDPs that share $\mathcal{S}, \mathcal{A}, \mathcal{P}, \rho_0$ and γ with the main task $\mathcal{T}_{\text{main}}$ but have their own reward functions, R_k. With this modification, we refer to entities in our model that are specific to task $\mathcal{T} \in \mathcal{T}_{\text{all}}$, $\mathcal{T}_{\text{all}} = \mathcal{T}_{\text{aux}} \cup \{\mathcal{T}_{\text{main}}\}$, as $(\cdot)_{\mathcal{T}}$. We assume that we have a set of expert data $\mathcal{B}^E_{\mathcal{T}}$ for each task.

6.4 Local Maximum with Off-Policy AIL

In this section, we provide a representative example of how AIL can fail by reaching a locally maximum policy due to a learned deceptive reward (Ecoffet et al., 2021) coupled with poor exploration. A simple six-state MDP is shown in Fig. 6.3, with ten state-conditional actions. We refer to actions as $a_t = a^{nm}$ and states as $s_t = s^n$ where t, n and m refer to the current timestep, current state, and next state, respectively. The reward function is $R(s^5, a^{55}) = +1$, $R(s^1, a^{15}) = -5$ and 0 for all other state-action pairs. The initial state s_1 is always s^1, the fixed horizon length is 5, and no discounting is used.

The MDP is meant to be roughly analogous to a stacking manipulation task: s^2, s^3, s^4 and s^6 represent the first block being reached, grasped, lifted, and dropped respectively. State s^5 represents

the gripper hovering over the second block (whether the first block has been stacked or not), while s^1 is the reset state, and a^{15} represents reaching s^5 without grasping the first block. Taking action a^{15} results in a total return of -1 (because $R(s^1, a^{15}) = -5$), since the first block has not actually been grasped. In our case, the agent does not receive any reward, and instead an expert demonstration of the optimal trajectory is provided. We will assume access to a learned (perfect) discriminator, and will use the AIRL (Fu et al., 2018a) reward, so state-action pairs in the expert set receive +1 reward and all others receive -1.

We define the action-value $Q(s_t, a_t)$ as the expected value of taking action a_t in state s_t, and initialize it to zero for all (s, a) pairs. We define our update rule as the standard Q-Learning update (Sutton and Barto, 2018), $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left(R(s_t, a_t) + \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$, with $\alpha = 0.1$. The agent uses ϵ-greedy exploration, storing each (s_t, a_t, s_{t+1}) tuple into a buffer. After each episode, all Q values are updated to convergence using the whole buffer.

After the first complete episode of $\{a^{15}, a^{55}, a^{55}, a^{55}, a^{55}\}$, $Q(s^1, a^{15}) = 2.7$, and $Q(s^1, a^{12}) = 0$. In the second ($\{a^{12}, a^{26}, a^{61}, a^{15}, a^{55}\}$) and third ($\{a^{12}, a^{23}, a^{36}, a^{61}, a^{15}\}$) episodes, the agent initially moves in the correct direction, but ultimately still fails. The final Q values in s^1 are $Q(s^1, a^{15}) = 0.49$ and $Q(s^1, a^{12}) = 0.13$.[5]

A policy maximizing Q, having simultaneously learned to avoid s^6 (by avoiding s^2 and s^3) and exploiting the (s^5, a^{55}) expert pair, will choose $a_1 = a^{15}$, giving a final return of -1 in the real MDP. This behaviour matches what we see in Fig. 6.1: due to the large negative reward from dropping the block, AIL learns a policy that avoids stacking altogether and merely reaches the second block, just as AIL here learns to skip s^2 and s^3 and exploit a^{55}. In both cases, poor initial exploration leads to a deceptive reward, which exacerbates poor exploration.

6.5 Learning from Guided Play (LfGP)

We now introduce Learning from Guided Play (LfGP). Our primary goal is to learn a policy $\pi_{\mathcal{T}_{\text{main}}}$ that can solve the main task $\mathcal{T}_{\text{main}}$, with a secondary goal of also learning auxiliary task policies $\pi_{\mathcal{T}_1}, \ldots, \pi_{\mathcal{T}_K}$ that are used for improved exploration. More specifically, we derive a hierarchical learning objective that is decomposed into three parts: (i) recovering the reward function of each task with expert demonstrations, (ii) training all policies to achieve their respective goals, and (iii) using all policies for effective exploration in $\mathcal{T}_{\text{main}}$. The complete algorithm is shown in Algorithm 3.

6.5.1 Learning the Reward Function

We first describe how to recover the reward functions from expert demonstrations. For each task $\mathcal{T} \in \mathcal{T}_{\text{all}}$, we learn a discriminator $D_{\mathcal{T}}(s, a)$ that is used to define the reward function for policy optimization. We construct the joint discriminator loss following (Kostrikov et al., 2019) to train each

[5] See `six_state_mdp.py` from open source code to reproduce.

Algorithm 3 Learning from Guided Play (LfGP)

Input: Expert replay buffers $\mathcal{B}^E_{\text{main}}, \mathcal{B}^E_1, \ldots, \mathcal{B}^E_K$, scheduler period ξ, sample batch size N
Parameters: Intentions $\pi_{\mathcal{T}}$ with corresponding Q-functions $Q_{\mathcal{T}}$ and discriminators $D_{\mathcal{T}}$, and scheduler π_S (e.g. with Q-table Q_S)

Initialize replay buffer $\mathcal{B}$
for $t = 1, \ldots,$ **do**
  # Interact with environment
  For every ξ steps, select intention $\pi_{\mathcal{T}}$ using π_S
  Select action a_t using $\pi_{\mathcal{T}}$
  Execute action a_t and observe next state s'_t
  Store transition $\langle s_t, a_t, s'_t \rangle$ in $\mathcal{B}$

  # Update discriminator $D_{\mathcal{T}'}$ for each task $\mathcal{T}'$
  Sample $\{(s_i, a_i)\}_{i=1}^N \sim \mathcal{B}$
  for each task $\mathcal{T}'$ **do**
    Sample $\{(s'_i, a'_i)\}_{i=1}^B \sim \mathcal{B}^E_k$
    Update $D_{\mathcal{T}'}$ following Eq. (6.1) + Gradient Penalty
  end for

  # Update intentions $\pi_{\mathcal{T}'}$ and Q-functions $Q_{\mathcal{T}'}$ for each task $\mathcal{T}'$
  Sample $\{(s_i, a_i)\}_{i=1}^N \sim \mathcal{B}$
  Compute reward $D_{\mathcal{T}'}(s_i, a_i)$ for each task $\mathcal{T}'$
  Update π and Q following Eq. (6.4) and Eq. (6.5)

  # *Optional* Update learned scheduler π_S
  if at the end of effective horizon **then**
    Compute main task return $G_{\mathcal{T}_{\text{main}}}$ using reward estimate from D_{main}
    Update π_S (e.g. update Q-table Q_S following Eq. (6.11) and recompute Boltzmann distribution)
  end if
end for

discriminator in an off-policy manner:

$$\mathcal{L}(D) = -\sum_{\mathcal{T} \in \mathcal{T}_{\text{all}}} \mathbb{E}_{\mathcal{B}} \left[\log\left(1 - D_{\mathcal{T}}(s,a)\right)\right] + \mathbb{E}_{\mathcal{B}^E_{\mathcal{T}}} \left[\log\left(D_{\mathcal{T}}(s,a)\right)\right]. \tag{6.1}$$

Each resulting discriminator $D_{\mathcal{T}}$ attempts to differentiate the occupancy measure between the distributions induced by $\mathcal{B}^E_{\mathcal{T}}$ and $\mathcal{B}$. We can use $D_{\mathcal{T}}$ to define various reward functions (Kostrikov et al., 2019); following (Fu et al., 2018a), we define the reward function for each task $\mathcal{T}$ to be $R_{\mathcal{T}}(s_t, a_t) = \log\left(D_{\mathcal{T}}(s_t, a_t)\right) - \log\left(1 - D_{\mathcal{T}}(s_t, a_t)\right)$.

6.5.2 Learning the Hierarchical Agent

We adapt Scheduled Auxiliary Control (SAC-X) (Riedmiller et al., 2018) to learn the hierarchical agent. The agent includes low-level intention policies (equivalently referred to as intentions), a high-level

scheduler policy, as well as the Q-functions and the discriminators. The intentions aim to solve their corresponding tasks (i.e., the intention $\pi_{\mathcal{T}}$ aims to maximize the task return $J(\pi_{\mathcal{T}})$), whereas the scheduler aims to maximize the expected return for $\mathcal{T}_{\text{main}}$ by selecting a sequence of intentions to interact with the environment. For the remainder of the paper, when we refer to a policy, we are referring to an intention policy, as opposed to the scheduler, unless otherwise specified.

Learning the Intentions

We learn each intention using Soft Actor-Critic (SAC) (Haarnoja et al., 2018), an actor-critic algorithm that maximizes the entropy-regularized objective, though any off-policy RL algorithm would suffice. The objective is

$$J(\pi_{\mathcal{T}}) = \mathbb{E}_{\pi_{\mathcal{T}}}\left[\sum_{t=0}^{\infty}\gamma^t\left(R_{\mathcal{T}}(s_t,a_t) + \alpha\mathcal{H}(\pi_{\mathcal{T}}(\cdot|s_t))\right)\right], \tag{6.2}$$

where the learned temperature α determines the importance of the entropy term and $\mathcal{H}(\pi_{\mathcal{T}}(\cdot|s_t))$ is the entropy of the intention $\pi_{\mathcal{T}}$ at state s_t. The soft Q-function is

$$Q_{\mathcal{T}}(s_t,a_t) = R_{\mathcal{T}}(s_t,a_t) + \mathbb{E}_{\pi_{\mathcal{T}}}\left[\sum_{t=0}^{\infty}\gamma^t(R_{\mathcal{T}}(s_{t+1},a_{t+1}) + \alpha\mathcal{H}(\pi_{\mathcal{T}}(\cdot|s_{t+1})))\right]. \tag{6.3}$$

The intentions maximize the joint policy objective

$$\mathcal{L}(\pi_{\text{int}}) = \sum_{\mathcal{T}\in\mathcal{T}_{\text{all}}}\mathbb{E}_{s\sim\mathcal{B}_{\text{all}},a\sim\pi_{\mathcal{T}}(\cdot|s)}\left[Q_{\mathcal{T}}(s,a) - \alpha\log\pi_{\mathcal{T}}(a|s)\right], \tag{6.4}$$

where π_{int} refers to the set of intentions $\{\pi_{\mathcal{T}_{\text{main}}},\pi_{\mathcal{T}_1},\ldots,\pi_{\mathcal{T}_K}\}$ and $\mathcal{B}_{\text{all}}$ refers to buffer containing every transition from interactions and demonstrations, as is done in (Vecerik et al., 2018; Kalashnikov et al., 2018). Notice that the replay buffer $\mathcal{B}$ contains transitions generated using every intention. This allows each intention $\pi_{\mathcal{T}}$ to learn optimal actions for its respective task starting at any state visited by any intention. In other words, other intentions $\pi_{\mathcal{T}'\neq\mathcal{T}}$ interacting with the environment provide exploration samples for $\pi_{\mathcal{T}}$.

For policy evaluation, the soft Q-functions $Q_{\mathcal{T}}$ for each $\pi_{\mathcal{T}}$ minimize the joint soft Bellman residual

$$\mathcal{L}(Q) = \sum_{\mathcal{T}\in\mathcal{T}_{\text{all}}}\mathbb{E}_{(s,a,s')\sim\mathcal{B}_{\text{all}},a'\sim\pi_{\mathcal{T}}(\cdot|s')}\left[\left(Q_{\mathcal{T}}(s,a) - \delta_{\mathcal{T}}\right)^2\right], \tag{6.5}$$

$$\delta_{\mathcal{T}} = R_{\mathcal{T}}(s,a) + \gamma\left(Q_{\mathcal{T}}(s',a') - \alpha\log\pi_{\mathcal{T}}(a'|s')\right). \tag{6.6}$$

Crucially, because each task shares the common $\mathcal{S},\mathcal{A},\mathcal{P},\rho_0$, and γ, and we are using off-policy learning, all tasks can learn from all data, as in (Riedmiller et al., 2018).

The Scheduler

SAC-X formulates learning the scheduler by maximizing the expected return of the main task (Riedmiller et al., 2018). In particular, let H be the number of possible intention switches within an episode and let each chosen intention execute for ξ timesteps. The H intention choices made within the episode are defined as $\mathcal{T}^{0:H-1} = \{\mathcal{T}^{(0)}, \ldots, \mathcal{T}^{(H-1)}\}$, where $\mathcal{T}^{(h)} \in \mathcal{T}_{\text{all}}$. The return of the main task, given chosen intentions, is then defined as

$$G_{\mathcal{T}_{\text{main}}}(\mathcal{T}^{0:H-1}) = \sum_{h=0}^{H-1} \sum_{t=h\xi}^{(h+1)\xi-1} \gamma^t R_{\mathcal{T}_{\text{main}}}(s_t, a_t), \tag{6.7}$$

where $a_t \sim \pi_{\mathcal{T}^{(h)}}(\cdot|s_t)$ is the action taken at timestep t, sampled from the chosen intention $\mathcal{T}^{(h)}$ in the h^{th} scheduler period. The scheduler for the h^{th} period P_S^h aims to maximize the expected main task return: $\mathbb{E}\left[G_{\mathcal{T}_{\text{main}}}(\mathcal{T}^{h:H-1})|P_S^h\right]$. In this work, we consider three options for a scheduler: learned, weighted random, and weighted random with additional handcrafted trajectories, each of which are described in the following paragraphs.

Learned Scheduler. We define the Q-function for the scheduler as

$$Q_S(\mathcal{T}^{0:h-1}, \mathcal{T}^{(h)}) = \mathbb{E}_{\mathcal{T}^{h:H-1} \sim P_S^{h:H-1}}\left[G_{\mathcal{T}_{\text{main}}}(\mathcal{T}^{h:H-1})|\mathcal{T}^{0:h-1}\right] \tag{6.8}$$

and represent the scheduler for the h^{th} period as a softmax distribution P_S^h over

$$\{Q_S(\mathcal{T}^{0:h-1}, \mathcal{T}_{\text{main}}), Q_S(\mathcal{T}^{0:h-1}, \mathcal{T}_1), \ldots, Q_S(\mathcal{T}^{0:h-1}, \mathcal{T}_K)\}. \tag{6.9}$$

The scheduler maximizes the expected return of the main task following the scheduler:

$$\mathcal{L}(S) = \mathbb{E}_{\mathcal{T}^{(0)} \sim P_S^0}\left[Q_S(\emptyset, \mathcal{T}^{(0)})\right]. \tag{6.10}$$

We use Monte Carlo returns to estimate Q_S, estimating the expected return using the exponential moving average:

$$Q_S(\mathcal{T}^{0:h-1}, \mathcal{T}^{(h)}) = (1-\phi)Q_S(\mathcal{T}^{0:h-1}, \mathcal{T}^{(h)}) + \phi\, G_{\mathcal{T}_{\text{main}}}(\mathcal{T}^{h:H}), \tag{6.11}$$

where $\phi \in [0, 1]$ represents the amount of discounting on older returns and $G_{\mathcal{T}_{\text{main}}}(\mathcal{T}^{h:H})$ is the cumulative discounted return of the trajectory starting at timestep $h\xi$.

Weighted Random Scheduler. The weighted random scheduler (WRS) forms a prior categorical distribution over the set of tasks, with a higher probability mass $p_{\mathcal{T}_{\text{main}}}$ for the main task and $\frac{p_{\mathcal{T}_{\text{main}}}}{K}$ for all other tasks. This approach is comparable to the uniform scheduler from (Riedmiller et al., 2018), with a bias towards the main task.

WRS with Handcrafted Trajectories. We add a small set of handcrafted trajectories of tasks that are sampled some of the time, forcing the scheduler to explore trajectories that would clearly be beneficial for completing the main task. For example, for a *Stack* task, we can provide *Reach, Lift, Stack, Open-Gripper*.

6.5.3 Breaking Out of Local Maxima with LfGP

Returning to the discussion in Section 6.4, resolving the local maximum problem with LfGP is straightforward. Suppose we include a *go-right* auxiliary task with $\mathcal{B}^E_{\text{go-right}} = \{(s^1, a^{12}), (s^2, a^{23}), (s^3, a^{34})\}$. When the scheduler chooses the go-right intention, the agent does not exploit the a^{55} action because the go-right discriminator learns that $R(s^5, a^{55}) = -1$. Since the transitions are stored in the shared buffer that the main intention also samples from, the agent can quickly obtain the correct, optimal value.

6.5.4 Expert Data Collection

We assume that each $\mathcal{T} \in \mathcal{T}_{\text{all}}$ has, for evaluation purposes only, a binary indicator of success. In single-task imitation learning where this assumption is valid, expert data is typically collected by allowing the expert to control the agent until success conditions are met. At that point, the environment is reset following ρ_0 and collection is repeated for a fixed number of episodes or (s, a) pairs. We collect our expert data in this way for each $\mathcal{T}$ separately. In LfGP, and more generally for multitask IL, expert data can also be collected this way for each $\mathcal{T}$ separately: we refer to this strategy, which we use for all of our main experiments, as *reset-based* expert data collection.

A limitation of reset-based expert data collection is that, when training LfGP, the initial state of each individual policy can be any state that $\mathcal{M}$ has been left in by a previous policy, which may include states not in the distribution of ρ_0. This "transition" initial state distribution, which we call $\rho_0(s|\mathcal{T}')$, where $\mathcal{T}' \in \mathcal{T}_{\text{all}}$ corresponds to the previously running policy $\pi_{\mathcal{T}'}$, would be challenging to sample from—it relies on the policies, and it may include states which are impractical to manually reset to (e.g. objects may start off as grasped or in mid-air). Consequently, an alternative data collection strategy exists, where we alternate between uniformly sampling the next task for an expert to complete and having the expert execute that task until success. In our implementation, we also reset the environment following ρ_0 periodically.

6.6 Experiments

In this work, we are interested in answering the following questions about LfGP:

1. How does the performance of LfGP compare with BC and AIL in challenging manipulation tasks, in terms of success rate and expert sample efficiency?
2. What parts of LfGP are necessary for success?
3. How do the policies and action value functions differ between AIL and LfGP?

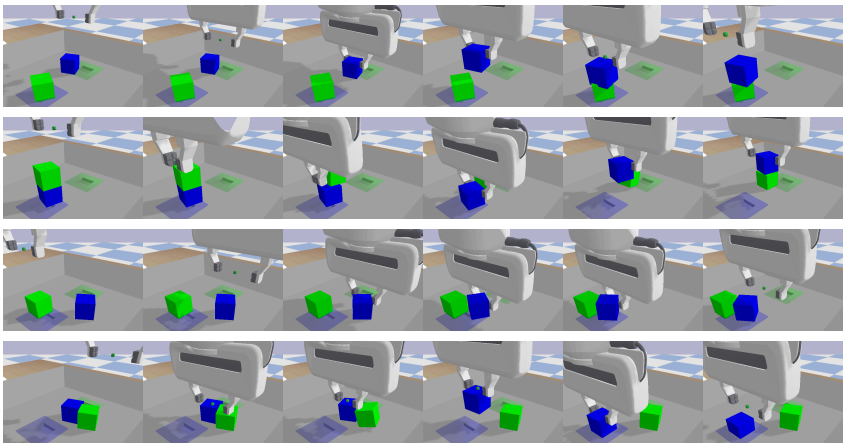


Figure 6.4: Example successful runs of our four main tasks. Top to bottom: `Stack`, `Unstack-Stack`, `Bring`, `Insert`.

6.6.1 Experimental Setup

We complete experiments in a simulation environment containing a Franka Emika Panda manipulator, one green and one blue block in a tray, fixed zones corresponding to the green and blue blocks, and one slot in each zone with $<$ 1mm tolerance for fitting the blocks (see bottom right of Fig. 6.4). The robot is controlled via delta-position commands, and the blocks and end-effector can both be reset anywhere above the tray. The environment is designed such that several different challenging tasks can be completed within a common observation and action space. The main tasks that we investigate are `Stack`, `Unstack-Stack`, `Bring`, and `Insert` (see Fig. 6.4). We also define a set of auxiliary tasks: `Open-Gripper`, `Close-Gripper`, `Reach`, `Lift`, `Move-Object`, and `Bring` (`Bring` is both a main task and an auxiliary task for `Insert`), all of which are reusable between main tasks.

We compare our method to several standard multitask and single-task baselines. A multitask algorithm simultaneously learns to complete a main task as well as auxiliary tasks, while the single-task algorithms only learn to complete the main task. In general, we consider a multitask algorithm to be more useful than a single-task algorithm, given the potential to reuse expert data and trained models for learning new tasks. To ensure a fair comparison, we provide single-task algorithms with an equivalent amount of *total* expert data as our multitask methods, as shown in Table 6.1.

In our main experiments, we compare LfGP to a multitask variant of behavioural cloning (BC), single-task BC, and discriminator-actor-critic (DAC) (Kostrikov et al., 2019), a state-of-the-art approach to AIL. We train multitask BC with a multitask mean squared error objective,

$$\mathcal{L}(\pi_{\text{int}}) = \sum_{\mathcal{T} \in \mathcal{T}_{\text{all}}} \sum_{(s,a) \in \mathcal{B}^E_{\mathcal{T}}} \left(\pi_{\mathcal{T}}(s) - a\right)^2, \tag{6.12}$$

while BC is trained with the corresponding single task version. Following recent trends in improving BC performance, we train our BC baselines with the same number of gradient updates as LfGP and DAC, evaluating the policies at the same frequency. This adjustment has been shown to dramatically

	Task	Dataset Sizes	Reuse	Single	Total
Multitask	`Stack`	S**OCRLM**: 1k/task	**5k**	1k	6k
	`U-Stack`	U**OCRLM**: 1k/task	**5k**	1k	6k
	`Bring`	**BOCRLM**: 1k/task	**6k**	0	6k
	`Insert`	I**BOCRLM**: 1k/task	**6k**	1k	7k
Single Task	`Stack`	S: 6k	0	6k	6k
	`U-Stack`	U: 6k	0	6k	6k
	`Bring`	B: 6k	0	6k	6k
	`Insert`	I: 6k	0	7k	7k

Table 6.1: The number of (s, a) pairs used for each main and auxiliary task. The table illustrates the reusability of the expert data used to generate the performance results described in Section 6.6.3. Each letter under "Dataset Sizes" is the first letter of a single (auxiliary) task, and bolded letters indicate that a dataset was reused for more than one main task (e.g., **O**pen-Gripper was used for all four main tasks). Multitask methods (e.g., LfGP) are able to reuse a large portion of the expert data, while single-task methods (e.g., single-task BC) cannot.

increase the performance of BC (Mandlekar et al., 2022; Hussenot et al., 2021), particularly compared to the more common practice of using early stopping, as is done in (Kostrikov et al., 2019; Ho and Ermon, 2016). We validate that this change significantly improves BC performance in our ablation study (see Section 6.6.4).

We gather expert data by first training an expert policy using scheduled auxiliary control (SAC-X) (Riedmiller et al., 2018). We then run the expert policies to collect various amounts of expert data as described in Section 6.5.4 and Table 6.1. We also collect an extra 200 expert $(s_T, \mathbf{0})$ pairs per auxiliary task, where T refers to the final timestep of an individual episode and $\mathbf{0}$ is an action of all zeros. This is equivalent to adding example data, as is done in example-based RL (Fu et al., 2018b). This addition improved final task performance, likely because it biases the reward towards completing the final task. It is worth noting that, in the real world, final states are easier to collect than full demonstrations, and LfGP does not require any modifications to accommodate these extra examples. Finally, even without this addition, LfGP still outperforms the baselines (see Section 6.6.4).

6.6.2 Scheduler Implementation Details

In early empirical results, we found that a weighted random scheduler combined with handcrafted trajectories performed comparably to a learned scheduler, while being considerably easier and more computationally efficient to implement. Our main experimental results from Section 6.6.3, therefore, were generated using a weighted random scheduler with handcrafted trajectories, as described in Section 6.5.2. This section provides more details about those handcrafted trajectories, which are reusable between main tasks.

Given a chosen proportion hyperparameter (0.5 in our experiments), we randomly sampled full trajectories from the lists below at the beginning of training episodes, and otherwise sampled from the regular WRS. For all four tasks *Main* = {*Stack, Unstack-Stack, Bring, Insert*}, we provided the following set of trajectories:

1. *Reach, Lift, Main, Open-Gripper, Reach, Lift, Main, Open-Gripper.*

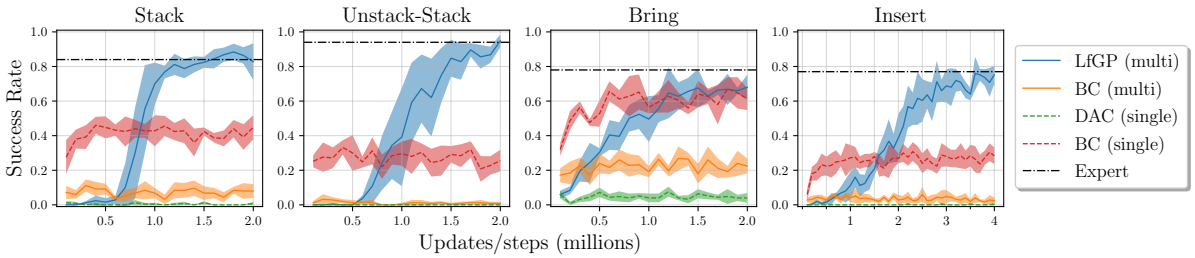


Figure 6.5: Performance results for LfGP, multitask BC, single-task BC, and DAC on all four tasks considered in this work. The x-axis corresponds to both gradient updates and environments steps for LfGP and DAC, and gradient updates only for both versions of BC. The shaded area corresponds to standard deviation across five seeds. LfGP significantly outperforms the baselines on all tasks, and even in Bring where it is matched by single-task BC, it is far more expert sample efficient.

2. *Reach, Lift, Move-Object, Main, Open-Gripper, Reach, Lift, Move-Object.*
3. *Lift, Main, Open-Gripper, Lift, Main, Open-Gripper, Lift, Main.*
4. *Main, Open-Gripper, Main, Open-Gripper, Main, Open-Gripper, Main, Open-Gripper.*
5. *Move-Object, Main, Open-Gripper, Move-Object, Main, Open-Gripper, Move-Object, Main.*

For insert, in addition to the trajectories listed above, we added two more trajectories to specifically accommodate *Bring* as an auxiliary task:

1. *Bring, Insert, Open-Gripper, Bring, Insert, Open-Gripper, Bring, Insert.*
2. *Reach, Lift, Bring, Insert, Open-Gripper, Reach, Lift, Bring.*

6.6.3 Performance Results

Performance results for all methods and main tasks are shown in Fig. 6.5. We freeze the policies every 100k steps and evaluate those policies for 50 randomized episodes, using only the mean action outputs for stochastic policies. For all algorithms, we test across five seeds and report the mean and standard deviation of all seeds.

In `Stack`, `Unstack-Stack`, and `Insert`, LfGP achieves expert performance, while the baselines all perform significantly worse. In Bring, LfGP does not quite achieve expert performance, and is matched by single-task BC. However, we note that LfGP is much more expert data efficient than single-task BC because it reuses auxiliary task data (see Table 6.1). A more direct comparison is multitask BC, which performs much more poorly than LfGP across all tasks, including Bring. Intriguingly, DAC also performs very poorly on all tasks, a phenomenon that we further explore in Section 6.8.

6.6.4 Ablation Study

While the fundamental idea of LfGP is relatively straightforward, it is worth considering alternatives to some of the specific choices made for our experiments. In this section, we complete an ablation study

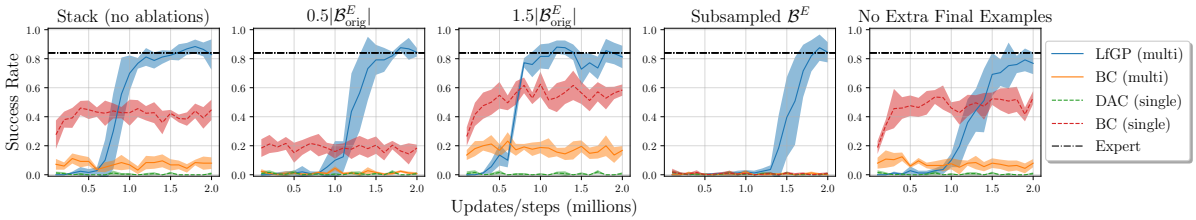


Figure 6.6: Various dataset ablations for LfGP and all baselines, including dataset size, subsampling of expert dataset, and replacement of extra $(s_T, \mathbf{0})$ pairs with an equivalent amount of regular trajectory (s, a) pairs. In all cases, LfGP still significantly outperforms all baselines.

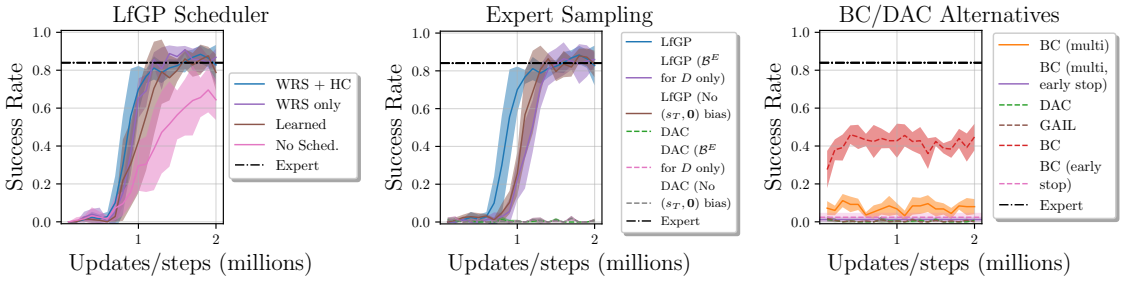
where we vary (a) the expert dataset, including size, subsampling, and inclusion of extra $(s_T, \mathbf{0})$ pairs; (b) the type of scheduler used for LfGP (see Section 6.5.2); (c) the sampling strategy used for expert data; and (d) the method for training our baselines. To reduce the computational load of completing these experiments, all of these variations were carried out exclusively for our `Stack` task.

Dataset Ablations

In Fig. 6.6, we show the results of the following dataset variations: (a) half and one and a half times the original expert dataset size; (b) subsampling $\mathcal{B}^E$, taking only every 20th timestep, as is done in (Kostrikov et al., 2019; Ho and Ermon, 2016); and (c) replacing the 200 extra $(s_T, \mathbf{0})$ pairs in each buffer with 200 regular trajectory (s, a) pairs. Notably, even in the challenging regimes of halving and subsampling the dataset, LfGP still learns an expert-level policy (albeit more slowly).

Scheduler Ablations

Fig. 6.7a shows the results of testing the following scheduler variations: (a) Weighted Random Scheduler (WRS) only, removing the Handcrafted (HC) addition; (b) a learned scheduler, as is used in (Riedmiller et al., 2018); and (c) no scheduler, in which only the main task is attempted, akin to the Intentional Unintentional Agent (Riedmiller et al., 2018; Cabi et al., 2017). Both WRS versions learn slightly faster than the learned scheduler, but all three methods outperform the No Scheduler ablation, replicating

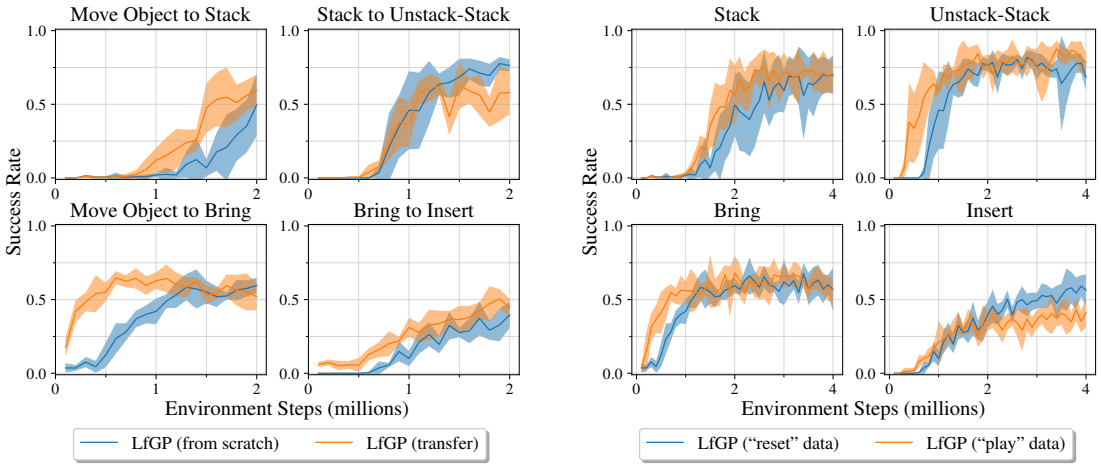


(a) Scheduler ablations for training LfGP.

(b) Expert sampling ablations for training LfGP/DAC.

(c) Baseline ablations for training BC/DAC.

Figure 6.7: Scheduler, sampling, and baseline ablation results.



(a) Transfer learning results.

(b) Play-based vs. reset-based expert data results.

Figure 6.8: Results on transfer learning and different schemes for collecting expert data.

results from (Riedmiller et al., 2018) demonstrating the importance of actually exploring all auxiliary tasks. Perhaps surprisingly, the HC modification made little difference compared with WRS only, but it is possible that for even more complex tasks, this could change.

Expert Sampling Ablations

For our main performance experiments, we modified standard AIL in two ways: (a) we added expert buffer sampling to π and Q updates, in addition to the D updates, as is done in (Vecerik et al., 2018; Kalashnikov et al., 2018); and (b) we biased the sampling of $\mathcal{B}^E$ when training D to be 95% final $(s_T, \mathbf{0})$ pairs. We tested both LfGP and DAC without these additions (see Fig. 6.7b). For LfGP, although these modifications improve learning speed, they are not required to generate an expert policy. For DAC, performance is quite poor regardless of these adjustments.

Baseline Ablations

To verify that we evaluated against fair baselines, we tested two alternatives to those used for our main performance experiments (see Fig. 6.7c): (a) an early stopping variation of BC, in which each expert buffer is divided into a 70%/30% train/validation split, taking the policy after validation error has not improved for 100 epochs; and (b) the on-policy variant of DAC, also known as Generative Adversarial Imitation Learning (GAIL) (Ho and Ermon, 2016). Notably, the early stopping variants of BC, commonly used as baselines in other AIL work (Ho and Ermon, 2016; Kostrikov et al., 2019; Zolna et al., 2021) perform dramatically more poorly than those used in our experiments, verifying recent trends (Mandlekar et al., 2022; Hussenot et al., 2021).

6.6.5 Transfer Learning Results

In this section, we show the results of experiments in which we attempt to reuse policies that learned to complete one main task to complete another main task.[2] Specifically, we are interested in reducing the learning time for a main task compared with learning from scratch. To complete these experiments, we chose a relatively simple formulation as a proof-of-concept. For each main task, we take a high-performing existing model that uses a subset of auxiliary tasks required for the new main task and save the parameters and the replay buffer of the existing model. We then train a new model on the new task by loading the saved parameters and replay buffer, adding new model outputs for the new main task. For `Stack` and `Bring`, we transfer from a model with $\mathcal{T}_{\text{main}}$ = Move-Object and $\mathcal{T}_{\text{aux}}$ = {Open-Gripper, Close-Gripper, Reach, Lift}. For `Unstack-Stack` and `Insert`, we use $\mathcal{T}_{\text{main}}$ = Stack and $\mathcal{T}_{\text{main}}$ = Bring models respectively, with the same $\mathcal{T}_{\text{aux}}$ as in Section 6.6.3.

The results of these experiments are shown in Figure 6.8a. It is clear that in all cases except for `Unstack-Stack`, transferring trained models from one main task to another is more sample efficient than from scratch. This result is particularly clear in the case of `Bring`. For `Unstack-Stack`, training speed remains roughly consistent, but final performance is actually considerably lower with the transfer model. Given that this experiment was only meant to prove that transfer learning would be possible and could provide benefits in some cases, we leave investigating this issue for future work.

6.6.6 Expert Data Collection Schemes

As described in Section 6.5.4, there are (at least) two strategies for collecting expert data to be used with LfGP. In this section, we compare the training performance of LfGP between reset-based and play-based expert data. We show the comparison between the results of training LfGP with reset-based and with play-based data in Figure 6.8b.[3] For `Stack`, `Unstack-Stack`, and `Bring`, play-based data appears to generally increase the learning speed of LfGP, implying that matching the transition distribution does appear to be beneficial for learning, although there does not appear to be any significant effect on final performance. Conversely, for `Insert`, play-based data appears to have only a marginal effect on learning speed, while having fairly significant negative impact on final performance. This could be because the Insert task is the least forgiving in terms of the required final state of the object, and reset-based data may actually contain more transitions between near-insertions and complete insertions than play-based data.

Compared with reset-based data, while play-based data assures that the expert distribution better matches the learning distribution, it also has the downside of making it harder to reuse. In the case of reset-based data, one can easily add a new dataset corresponding to a new task, while keeping existing datasets the same. In play-based, and in our experiments, each individual main task has its own dataset, given that the "transition" initial state distribution should contain data from $\mathcal{T}_{\text{all}}$, which

[2] These experiments were completed using an earlier version of LfGP before a variety of bug fixes and implementation changes had improved general performance. Therefore, the overall performance on each task is lower than what we found in our main results. Nonetheless, these results indicate that the LfGP framework allows for a degree of transfer learning.

[3] The same note from Footnote 2 applies to these experiments.

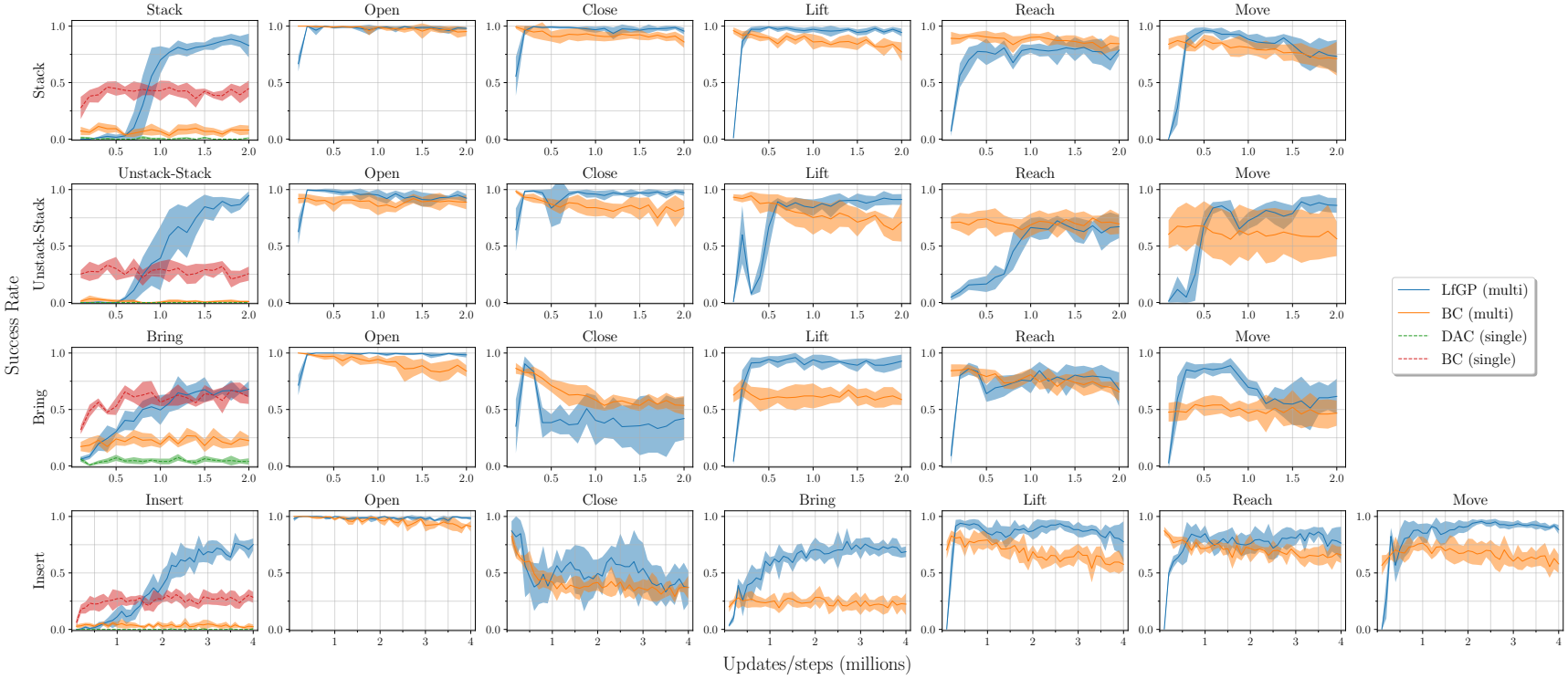


Figure 6.9: Performance for LfGP and the multitask baselines across all tasks, shaded area corresponds to standard deviation.

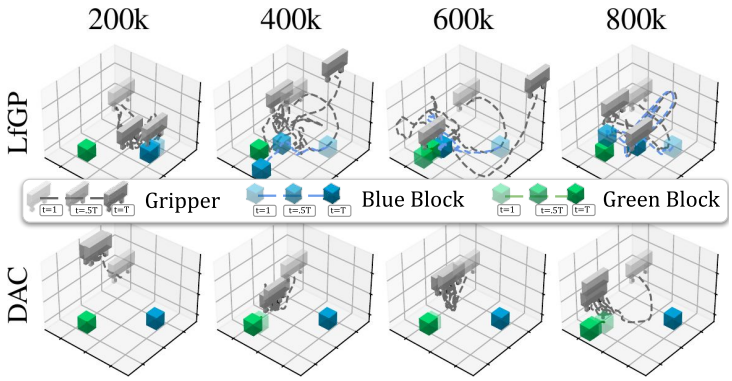


Figure 6.10: LfGP and DAC trajectories of the gripper, blue block, and green block for four stack episodes with consistent initial conditions throughout the learning process. The LfGP episodes, each including auxiliary task sub-trajectories, demonstrate significantly more variety than the DAC trajectories.

changes depending on $\mathcal{T}_{\text{main}}$. One could achieve reusability in expert data by creating a separate play-based dataset for each of the reusable tasks, and then use reset-based data for the non-reusable main tasks, but we did not investigate this option for this work.

6.7 Performance Results for Auxiliary Tasks

The performance results for all multitask methods and all auxiliary tasks are shown in Fig. 6.9. Multitask BC has gradually decreasing performance on many of the auxiliary tasks as the number of updates increases, which is consistent with mild overfitting. Intriguingly, however, multitask BC does achieve quite reasonable performance on many of the auxiliary tasks (such as `Lift`) without needing any of the extra environment interactions required by an online method such as LfGP or DAC. An interesting direction for future work is to determine whether pretraining via multitask BC could provide any improvements in environment sample efficiency. We did attempt to do this, but found that it resulted in poorer final performance than training from scratch.

6.8 Learned Model Analysis

In this section, we further examine the learned `Stack` models of LfGP and DAC. We take snapshots of the average performing models from LfGP and DAC at four points during learning: 200k, 400k, 600k, and 800k model updates and environment steps. Although the initial gripper and block positions are

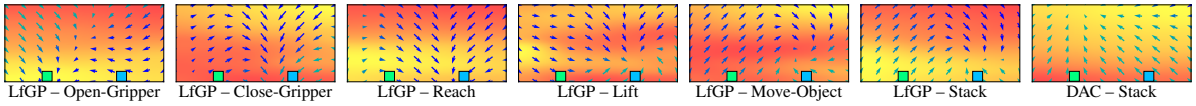


Figure 6.11: The policy outputs (arrows) and Q values (background) for each LfGP task and for DAC at 200k environment steps. The arrows show velocity direction/magnitude, blue → green indicates open-gripper → close-gripper. For Q values, red → yellow indicates low → high. The LfGP policies and Q functions are reasonable for all tasks, while DAC has only learned to reach toward and above the green block.

randomized between episodes during learning, for each snapshot, we reset the stacking environment to a single set of representative initial conditions. We then run the snapshot policies for a single exploratory trajectory, using the stochastic outputs of each policy as well as, for LfGP, the WRS+HC scheduler. Trajectories from these runs are shown in Fig. 6.10.

DAC is unable to learn to grasp or even reach the blue block and ultimately settles on a policy that learns to reach and hover near the green block. This is understandable—DAC learns a deceptive reward for hovering above the green block regardless of the position of the blue block, because it has not sufficiently explored the alternative of first grasping the blue block. Even if hovering above the green block does not fully match the expert data, the DAC policy receives some reward for doing so, as evidenced by the learned Q value on the right side of Fig. 6.11.

In comparison, even after only 200k environment steps, LfGP learns to reach and push the blue block, and by 600k steps, grasp, move, and nearly stack it. By enforcing exploration of sub-tasks that are crucial to completing the main task, LfGP ensures that the distribution of expert stacking data is fully matched.

6.9 Limitations

Our approach is not without limitations. While we were able to use LfGP in six and seven-task settings, the number of tasks for which this method would become intractable is unclear. LfGP needs access to segmented expert data as well; in many cases, this is reasonable, and is also required to be able to reuse auxiliary task data between main tasks, but it does necessitate extra care during expert data collection. Also, LfGP requires pre-defined auxiliary tasks: while this is a common approach to hierarchical RL (see (Pateria et al., 2021), Section 3.1, for numerous examples), choosing these tasks may sometimes present a challenge. Finally, compared with methods that use offline data exclusively (e.g., BC), for our tasks, LfGP requires many online environment steps to learn a high-quality policy. This data gathering could be costly if human supervision was necessary. It is worth noting that, because LfGP is already a multitask method, this final point could be partially resolved through the use of multitask reset-free RL (Gupta et al., 2021).

6.10 Summary

In this chapter, we have shown how adversarial imitation learning can fail at challenging manipulation tasks because it learns deceptive rewards. We demonstrated that this can be resolved with Learning from Guided Play (LfGP), in which we introduce auxiliary tasks and the corresponding expert data, *guiding* the agent to *playfully* explore parts of the state and action space that would have been avoided otherwise. We demonstrated that our method dramatically outperforms both BC and AIL baselines, particularly in the case of AIL. Furthermore, our method can leverage reusable expert data, making it significantly more expert sample efficient than the highest-performing baseline, and its learned auxiliary task models can be applied to transfer learning. In future work, we intend to investigate transfer

learning to determine if overall policy learning time can be reduced.

The practical benefit of IRL over BC, and therefore over methods presented in Chapters 3 to 5 is, generally, that performance can be continuously improved through autonomous exploration. This can be interpreted as having the same benefit as DAgger (Ross et al., 2011), where we remove the negative effect of distribution shift on policy performance. As stated, in this work, we found that modern approaches to IRL actually had a dramatically reduced success rate compared with BC for manipulation tasks. We showed that the performance of LfGP far exceeds both IRL and BC, which ultimately realizes the promise of IRL over BC by removing the inevitable distribution shift from BC.

Chapter 7

Auxiliary Control from Examples

Until this point, we have largely ignored an important question in imitation learning: what if the expert data is suboptimal? There are many reasons this can occur, such as the lack of a proficient expert, or due to an inherent disconnect between the expert's capabilities and the agent's (such as the one discussed in Chapter 4). In this chapter, we consider *examples of success*, in which only examples of completed tasks are provided to the agent, rather than full expert trajectories.[1] Examples of success are far less susceptible to being suboptimal, and they are much easier to provide than full expert trajectories.

The exclusion of full expert trajectories completely circumvents the distribution shift problem explored in Chapters 3 to 6. The trade-off is that learning from exclusively *sparse* expert data is, similar to having a sparse reward, a difficult exploration problem (see Section 2.3.3). The multitask exploration tools developed and applied to inverse reinforcement learning in learning from guided play (LfGP, Chapter 6) are a natural fit for improving exploration in the example-based setting. In this chapter, we show that the direct application of LfGP to the example-based problem can result in overestimated Q-values and poor performance, and we resolve the problem with a novel above-success-level value penalty. Combining this penalty, the multitask framework for improving exploration from Chapter 6, and expert data containing only examples of success, in this chapter we present **VPACE**: **v**alue-**p**enalized **a**uxiliary **c**ontrol from **e**xamples. Across both simulated and real robotic environments, we show that our approach substantially improves learning efficiency for challenging tasks, while maintaining bounded value estimates. We compare with existing approaches to example-based learning, inverse reinforcement learning, and an exploration bonus. Preliminary results also suggest that our multitask example-based approach may learn more efficiently than the more common approaches of using full trajectories or true sparse rewards.

Compared with Chapters 3 to 6, VPACE puts the lowest burden on an expert. As stated, collecting sets of successful examples is, in general, far less laborious for a demonstrator than collecting full trajectories, which were required for all of the previous chapters.

[1] Project website: https://papers.starslab.ca/vpace.

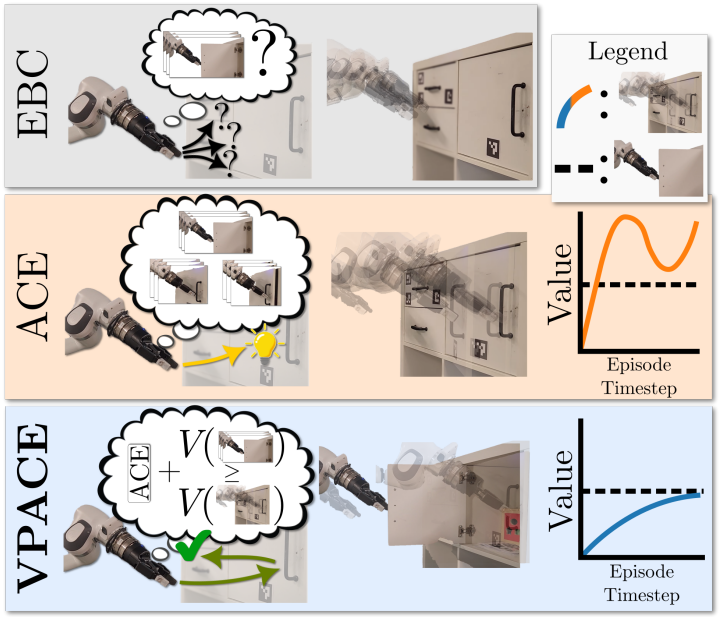


Figure 7.1: Example-based control (EBC) is inefficient due to poor exploration resulting from the inherently sparse nature of individual examples of success as a means for feedback. Auxiliary control from examples (ACE) remedies this by adding scheduled exploration of semantically meaningful auxiliary tasks, but can result in poor performance due to the unbounded value error of highly exploratory data. We combine ACE with value penalization (VP) as VPACE to efficiently learn from examples of success.

7.1 Motivation

Robotics presents a unique challenge to learning algorithms: because robotics data is costly to gather, ensuring that algorithms have high sample efficiency is crucial. In reinforcement learning and imitation learning, common approaches to improving sample efficiency incorporate a manually-defined *dense* reward function or demonstration trajectories of an expert completing the task, respectively. These additions, if they are available at all, can be highly biased or suboptimal. *Sparse* reward functions are less biased (Ng and Jordan, 2003), but require significant exploration (Vasan et al., 2024). Sparse reward functions may also be unavailable, because they require accurate state estimation and the tuning of various parameters (e.g., how high must an object be lifted?).

We consider another form of feedback—*example states* of completed tasks, sometimes referred to as example-based control (EBC) (Eysenbach et al., 2021). Obtaining example states can be far less laborious than designing a reward function or gathering trajectories; practitioners can gather states from a distribution that represents a completed task without consideration of *how* the states are reached. However, just as in the case of sparse rewards, excluding information on how goal states are reached can lead to highly inefficient learning (e.g., an example of a loaded dishwasher provides no information about the long sequence of actions required to complete the task). In this work, our primary goal is to improve the sample efficiency of example-based control.

To answer this question, we propose a new example-based control method, value-penalized auxiliary control from examples (VPACE). Since hierarchical RL (HRL) has shown success in improving exploration in robotics domains (Nachum et al., 2019), we leverage an HRL approach known as sched-

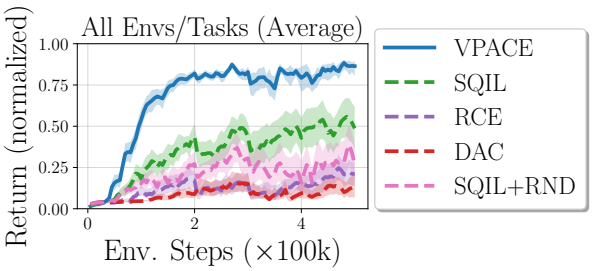


Figure 7.2: Average performance across all environments and tasks studied in this work. Results are shown as an interquartile mean with five seeds per algorithm and task, and shaded regions show 95% stratified bootstrap confidence intervals (Agarwal et al., 2021).

uled auxiliary control (SAC-X) (Riedmiller et al., 2018) to improve exploration in EBC. The SAC-X framework enables practitioners to introduce a set of simple and often reusable auxiliary tasks, in addition to the main task, that help the agent to explore the environment. In this work, the full set of auxiliary tasks that we use across all main tasks are reach, grasp, lift, and release. Instead of defining main and auxiliary tasks with sparse rewards (Riedmiller et al., 2018) or full expert trajectories (Ablett et al., 2023), we define tasks with examples. For each auxiliary task there is a corresponding auxiliary policy that learns to match the set of examples. A scheduler periodically chooses and executes the different auxiliary policies, in addition to the main policy, generating a more diverse set of data to learn from.

We find that the naïve application of SAC-X to EBC can result in overestimated values, leading to sample inefficiency and poor performance. To remedy this, and show how SAC-X can be effectively applied to EBC, this work makes the following contributions:

1. We demonstrate that the introduction of the SAC-X framework significantly improves exploration and learning efficiency in EBC.

2. We remedy this overestimation problem by introducing a value-penalization method that is based on the expected value of the examples.

3. We conduct experiments across four environments with 19 simulated and two real tasks to show the improved sample efficiency and final performance of VPACE over EBC, inverse reinforcement learning, and an exploration bonus.

4. We compare to the use of full trajectories and true sparse rewards, and observe that VPACE has higher sample efficiency than both.

7.2 Related Work

Sparse rewards are a desirable form of feedback for learning unbiased, optimal policies in reinforcement learning (RL), but they are not always obtainable, and present an immense exploration challenge on long-horizon tasks (Gupta et al., 2022). Reward shaping (Ng et al., 1999) and dense rewards can help alleviate the exploration problem in robotics (Popov et al., 2017; Yu et al., 2019), but designing dense

rewards is difficult for practitioners (Andrychowicz et al., 2017). An alternative to manually-defined rewards is to perform inverse RL (IRL), in which a reward function is recovered from demonstrations, and a policy is learned either subsequently (Ng and Russell, 2000) or simultaneously via adversarial imitation learning (AIL) (Kostrikov et al., 2019; Reddy et al., 2020; Ho and Ermon, 2016; Fu et al., 2018a).

Like dense rewards, full trajectory demonstrations can be hard to acquire, suboptimal, or biased. Unlike IRL/AIL, in *example-based control* (EBC), a learning agent is only provided distributions of single *successful example states*. Previous EBC approaches include using generative AIL (GAIL, (Ho and Ermon, 2016)) directly (VICE, (Fu et al., 2018b)), soft actor critic (SAC, (Haarnoja et al., 2018)) with an additional mechanism for generating extra success examples (VICE-RAQ, (Singh et al., 2019)), performing offline RL with conservative Q-learning (CQL, (Kumar et al., 2020)) and a learned reward function (Hatch et al., 2023), and using SAC with a classifier-based reward (RCE, (Eysenbach et al., 2021)).

All EBC methods can naturally suffer from poor exploration, given that success examples are akin to sparse rewards. Hierarchical reinforcement learning (HRL) aims to leverage multiple levels of abstraction in long-horizon tasks (Sutton et al., 1999), improving exploration in RL (Robert et al., 2023; Nachum et al., 2019; Nair et al., 2018). Scheduled auxiliary control (SAC-X, (Riedmiller et al., 2018)) combines a scheduler with semantically meaningful and simple auxiliary sparse rewards or auxiliary full expert trajectories (LfGP, (Ablett et al., 2023, 2021a; Xiang et al., 2024)). Like us, (Wu et al., 2021) combined EBC with hierarchical learning, but their approach required a symbolic planner at test time, and generated very slow policies: our policies are fast and reactive, and don't require high-level planning at test time.

Learning value functions in the off-policy setting can be challenging due to the deadly triad (Sutton and Barto, 2018). Regularization and clipping techniques have been applied to address various problems such as stabilizing the bootstrapping target (Andrychowicz et al., 2017; Adamczyk et al., 2024) and preventing overfitting and out-of-distribution samples (Kumar et al., 2020; James et al., 2022). Our proposed value-penalization technique specifically targets overestimation in EBC.

7.3 Example-Based Control with Value-Penalization and Auxiliary Tasks

Our goal is to generate an agent that can complete a task given, as opposed to rewards or demonstrations, only *final-state examples* of a successfully completed task, with as few environment interactions as possible. We also assume access to final state examples of a small set of reusable auxiliary tasks. We begin by formally describing the problem setting for example-based control in Section 7.3.1. In Section 7.3.2, we describe how scheduled auxiliary tasks can be applied to example-based control. Finally, motivated by the increased exploration diversity of the multitask framework, we propose a new Q-estimation objective in Section 7.3.3 that leverages value penalization for improved learning stability.

7.3.1 Problem Setting

For a background on Markov decision processes (MDPs) and the notation used in this section, see Section 2.2. For any variables x_t, x_{t+1}, we may drop the subscripts and use x, x' instead when the context is clear.

In this work, we focus on *example-based control* (EBC), a more difficult form of imitation learning where we have no access to rewards or demonstrations, but instead are given a set of example states of a completed task: $s^* \in \mathcal{B}^*$, where $\mathcal{B}^* \subseteq \mathcal{S}$ and $|\mathcal{B}^*| < \infty$. The goal is to (i) leverage $\mathcal{B}^*$ and $\mathcal{B}$ to learn or define a state-conditional reward function $\hat{R} : \mathcal{S} \to \mathbb{R}$ that satisfies $\hat{R}(s^*) \geq \hat{R}(s)$ for all $(s^*, s) \in \mathcal{B}^* \times \mathcal{B}$, and (ii) learn a policy $\hat{\pi}$ that maximizes the expected return $\hat{\pi} = \arg\max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \hat{R}(s_t)\right]$.

For any policy π, we can define the value function and Q-function respectively to be $V^\pi(s) = \mathbb{E}_\pi \left[Q^\pi(s, a)\right]$ and $Q^\pi(s, a) = \hat{R}(s) + \gamma \mathbb{E}_{\mathcal{P}} \left[V^\pi(s')\right]$, corresponding to the return-to-go from state s (and action a). Temporal difference (TD) algorithms aim to estimate V^π or Q^π to evaluate a policy (Sutton and Barto, 2018). Given a reward model $\hat{R}(\cdot)$, we can say $\hat{R}(s^*), s^* \in \mathcal{B}^*$, indicates reward for successful states and $\hat{R}(s), s \in \mathcal{B}$, for all other states. Assuming that s^* transitions to itself, then for policy evaluation with mean-squared error (MSE), we can write the TD targets for non-successful and successful states, $y : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, of Q-updates as

$$y(s, s') = \hat{R}(s) + \gamma \mathbb{E}_\pi \left[Q(s', a')\right], \tag{7.1}$$

$$y(s^*, s^*) = \hat{R}(s^*) + \gamma \mathbb{E}_\pi \left[Q(s^*, a')\right], \tag{7.2}$$

where $(s, \cdot, s') \sim \mathcal{B}$ and $s^* \sim \mathcal{B}^*$. The reward function $\hat{R}$ can be based on a learned discriminator that differentiates between $s \sim \mathcal{B}$ and $s^* \sim \mathcal{B}^*$, akin to a state-only version of adversarial imitation learning (Kostrikov et al., 2019; Ho and Ermon, 2016; Fu et al., 2018a), or defined directly, such as $\hat{R}(s^*) = 1$ and $\hat{R}(s) = 0$ (Eysenbach et al., 2021; Reddy et al., 2020).

7.3.2 Learning a Multitask Agent from Examples

We alleviate the challenging exploration problem of EBC by introducing auxiliary control from examples (ACE), an application of the scheduled auxiliary control framework (Riedmiller et al., 2018; Ablett et al., 2023). This hierarchical approach aims to improve exploration by learning from examples of simpler auxiliary tasks, in addition to a main task, to impose a curriculum where the completion of easier tasks provides guidance on how to complete harder ones (Bengio et al., 2009). Auxiliary tasks can be selected to have semantic meaning: in this work, the auxiliary tasks used are reach, grasp, lift, and release. The task definitions, and sometimes the examples themselves, can also be reusable between main tasks. Each task has a corresponding *intention* that learns from the given task-specific examples. ACE leverages a scheduler to sequentially choose and execute individual intention policies allowing for more diverse state coverage to facilitate faster learning of the main task. Crucially, because the method is off-policy, each intention can learn from data generated by any other intention, and all policy interaction data, regardless of which intention generated it, is stored in $\mathcal{B}$.

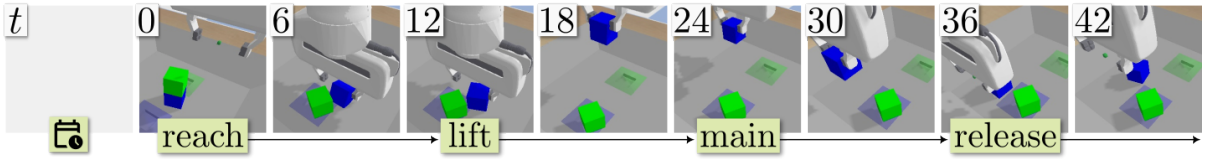


Figure 7.3: An example of fixed-period scheduler choices throughout an `Unstack-Stack` exploratory episode.

Formally, given an MDP $\mathcal{M}$, a task $\mathcal{T}$ is defined by a task-specific example buffer $\mathcal{B}^*_\mathcal{T}$. EBC methods such as RCE (Eysenbach et al., 2021) aim to exclusively complete the main task $\mathcal{T}_{\text{main}}$, while ACE adds auxiliary tasks $\mathcal{T}_{\text{aux}} = \{\mathcal{T}_1, \ldots, \mathcal{T}_K\}$ during learning. We refer the set of all tasks as $\mathcal{T}_{\text{all}} = \mathcal{T}_{\text{aux}} \cup \{\mathcal{T}_{\text{main}}\}$. ACE agents are composed of two types of policies—intentions for each task and a scheduler.

Intentions

For each task $\mathcal{T} \in \mathcal{T}_{\text{all}}$, the corresponding intention consists of a task-specific policy $\pi_\mathcal{T}$, Q-function $Q_\mathcal{T}$, and state-conditioned reward $\hat{R}_\mathcal{T}$. ACE optimizes the task-specific policies by maximizing the policy optimization objective

$$\mathcal{L}(\pi; \mathcal{T}) = \mathbb{E}_{\mathcal{B}, \pi_\mathcal{T}} \left[Q_\mathcal{T}(s, a) \right]. \tag{7.3}$$

The task-specific Q-functions are optimized via minimization of the Bellman residual

$$\mathcal{L}(Q; \mathcal{T}) = \mathbb{E}_{\mathcal{B}, \pi_\mathcal{T}} \left[\left(Q_\mathcal{T}(s, a) - y_\mathcal{T}(s, s') \right)^2 \right] + \mathbb{E}_{\mathcal{B}^*_\mathcal{T}, \pi_\mathcal{T}} \left[\left(Q_\mathcal{T}(s^*, a) - y_\mathcal{T}(s^*, s^*) \right)^2 \right], \tag{7.4}$$

where $y_\mathcal{T}$ are TD targets defined based on Eqs. (7.1) and (7.2) with task-specific reward $\hat{R}_\mathcal{T}$. Intuitively, each $\pi_\mathcal{T}$ aims to maximize the estimated task-specific value $Q_\mathcal{T}$.

Scheduler

Given the $K + 1$ intentions, a scheduler periodically selects a policy $\pi_\mathcal{T}$ to execute within an episode. Since each $\pi_\mathcal{T}$ aims to solve its own task, one can expect the transitions gathered composing by different $\pi_\mathcal{T}$ within the same episode to be diverse. In practice, we implement the scheduler to have a fixed period, guaranteeing a specific number of policy switches within each episode, depending on the episode horizon. We use a weighted random scheduler (WRS) with hyperparameter $p_{\mathcal{T}_{\text{main}}}$ where the probability of choosing the main task or an auxiliary task is $p_{\mathcal{T}_{\text{main}}}$ and $p_{\mathcal{T}_k} = (1 - p_{\mathcal{T}_{\text{main}}})/K$, respectively. We combine a WRS with a small set of simple handcrafted high-level trajectories (e.g., *reach* then *grasp* then *lift*). The handcrafted trajectory definitions are reusable between main tasks, and are not required to use our framework. Ablett et al. (2023) demonstrated that this approach performed better than a more complex learned scheduler. At test time, the scheduler is unused, and only π_{main} is evaluated.

7.3.3 Value Penalization in Example-Based Control

A scheduled multitask agent exhibits far more diverse behaviour than a single-task agent (Riedmiller et al., 2018; Ablett et al., 2023). We show in Figs. 7.8 and 7.9 that the buffer generated by this behavior, consisting of transitions resulting from multiple policies, can result in highly overestimated Q-values in EBC. This overestimation leads the policy to maximize an incorrect objective. In this section, we propose a novel penalty for TD algorithms that encourages Q-estimates to stay within the range of valid returns with respect to the reward model. This penalty applies to both the single-task and multitask regime. For simplicity, we describe value penalization for the former.

Notice that regressing to TD targets Eqs. (7.1) and (7.2) will eventually satisfy the Bellman equation, but in the short term the TD targets do not satisfy $y(s, s') \leq y(s^*, s^*)$. This is because the TD targets are computed by bootstrapping from a Q-estimate that may not satisfy the Bellman equation and can exceed the bounds of valid Q-values, implying that approximation error of Q updated via MSE can be uncontrollable. We resolve this issue with a penalty for our TD updates for $s \in \mathcal{B}$. We add a minimum and a maximum $Q^\pi(s, a)$ as $Q^\pi_{\text{min}} = \hat{R}_{\text{min}}/(1-\gamma)$ and $Q^\pi_{\text{max}} = \mathbb{E}_{\mathcal{B}^*}\left[V^\pi(s^*)\right]$, where $\hat{R}_{\text{min}} \leq \hat{R}(s)$ for all $s \in \mathcal{B}$. Then, the value penalty is defined to be

$$\mathcal{L}^\pi_{\text{pen}}(Q) = \lambda \mathbb{E}_{\mathcal{B}}[(\max(Q(s,a) - Q^\pi_{\text{max}}, 0))^2 + (\max(Q^\pi_{\text{min}} - Q(s,a), 0))^2], \tag{7.5}$$

where $\lambda \geq 0$ is a hyperparameter. When $\lambda \to \infty$, Eq. (7.5) becomes a hard constraint. It immediately follows that $y(s, s') \leq y(s^*, s^*)$ holds with TD updates Eqs. (7.1) and (7.2). We add value penalization $\mathcal{L}^\pi_{\text{pen}}(Q)$ to the MSE loss as a regularization term for learning the Q-function.

7.4 Experiments

We aim to answer the following questions through our experiments:

1. **(RQ1)** How does the sample efficiency of VPACE compare to EBC, inverse RL, and exploration-bonus baselines?
2. **(RQ2)** How important is it to include value penalization (VP) with ACE?
3. **(RQ3)** How does our value penalty compare to existing Q regularizers?
4. **(RQ4)** How does VPACE compare to algorithms that use full trajectories and true sparse rewards?

7.4.1 Experimental Setup

Environments

We conduct experiments in a large variety of tasks and environments, including those originally used in LfGP (Ablett et al., 2023) and RCE (Eysenbach et al., 2021). Specifically, the tasks in (Ablett et al., 2023) involve a simulated Franka Emika Panda arm, a blue and green block, a fixed "bring" area for

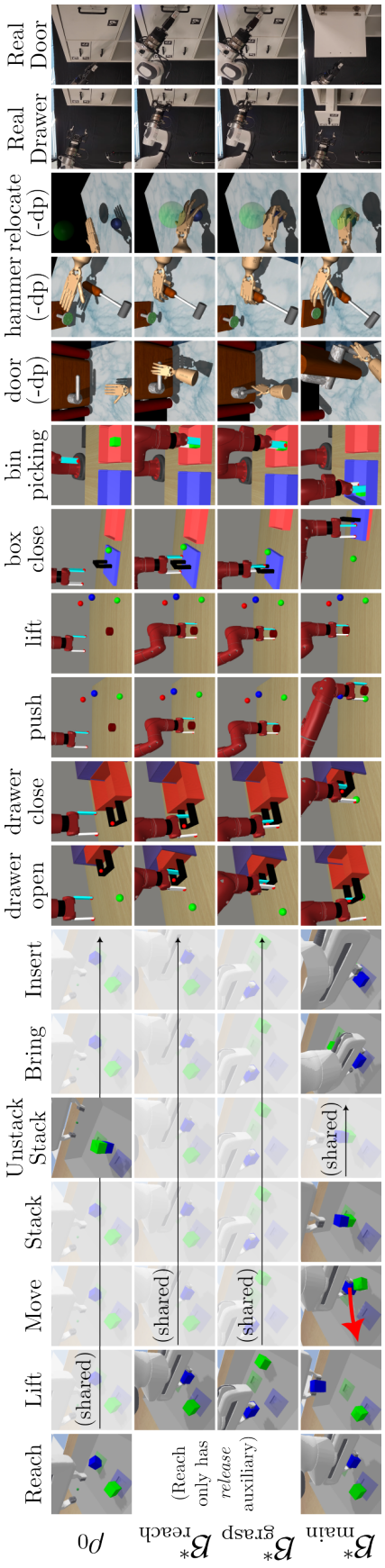


Figure 7.4: Samples from initial-state distribution ρ_0, auxiliary task examples $\mathcal{B}^*_{\text{aux}}$, and main task examples $\mathcal{B}^*_{\text{main}}$ for all tasks. The simulated Panda tasks additionally share $\mathcal{B}^*_{\text{release}}$ and $\mathcal{B}^*_{\text{lift}}$, while the Adroit original and dp environments share data since $\mathcal{S}$ does not change.

Table 7.1: Environment details.

	Obs. Space	Act Space	Aux. Tasks
Sim. Panda (Ablett et al., 2023, 2021a)	pos, vel, grip pos, prev. grip pos, obj. pos, obj. vel	XYZ delta-pos, binary grip	reach, grasp, lift, release
Sawyer (Eysenbach et al., 2021; Yu et al., 2019)	pos, obj. pos, grip pos, 3 frame stack	XYZ delta-pos, binary grip	reach, grasp
Adroit (Eysenbach et al., 2021; Rajeswaran et al., 2018)	pos, vel, obj. pos, finger pos	(up to) 6-DOF abs. / delta-pos, finger pos	reach, grasp
Real Panda	pos, obj. pos (ArUco (Garrido-Jurado et al., 2014)), grip pos, 2 frame stack	X / XY delta-pos, binary grip	reach, grasp

each block, and a small slot with $<$1 mm tolerance for inserting each block. This environment provides various manipulation tasks that share the same state-action space. The tasks in (Eysenbach et al., 2021) are a modified subset of those from (Yu et al., 2019), involving a simulated Sawyer arm, and three of the Adroit hand tasks originally presented in (Rajeswaran et al., 2018). We also generate three modified delta-position (dp) Adroit hand environments, because we found that policies learned in the original absolute-position environments lack finesse and exploit simulator bugs. Finally, we study drawer and door opening tasks with a real Franka Emika Panda.

Baselines

We consider many baselines, including an approach based on recursively classifying examples (**RCE**, (Eysenbach et al., 2021)), a learned reward model for off-policy RL (**DAC**, (Kostrikov et al., 2019)), a defined reward model for off-policy RL (**SQIL**, (Reddy et al., 2020)), and a combination of the best performing baseline (SQIL) with a method that provides an exploration bonus to unseen data (**SQIL+RND**, (Burda et al., 2019; Balloch et al., 2024)). Finding that SQIL significantly outperformed the baselines, we use it as the default reward model for our main experiments, although VP and ACE can be added to any of the approaches described above. Specifically, in our results, **VPACE** refers to SQIL with both VP and ACE applied, while **ACE** refers to SQIL with ACE and *without* VP applied.

Implementation

Observation space, action space, and auxiliary task details are shown in Table 7.1. All simulated tasks use 200 examples per task, while the real world tasks use 50 examples per task. All implementations are built on LfGP (Ablett et al., 2023; Chan, 2020) and SAC (Haarnoja et al., 2018). For more environment and implementation details, see our open-source code.

7.4.2 Performance Results

Main Task VP and ACE Benefits

To answer RQ1, we compare VPACE with existing EBC, inverse RL, and exploration-bonus approaches on all tasks and evaluate their success rates. Since the Sawyer and Adroit tasks do not evaluate success, we only report returns. Policies are evaluated at 25k (environment step) intervals for 50 episodes for the simulated Panda tasks, 10k intervals for 30 episodes for the Sawyer and Adroit tasks, and 5k intervals

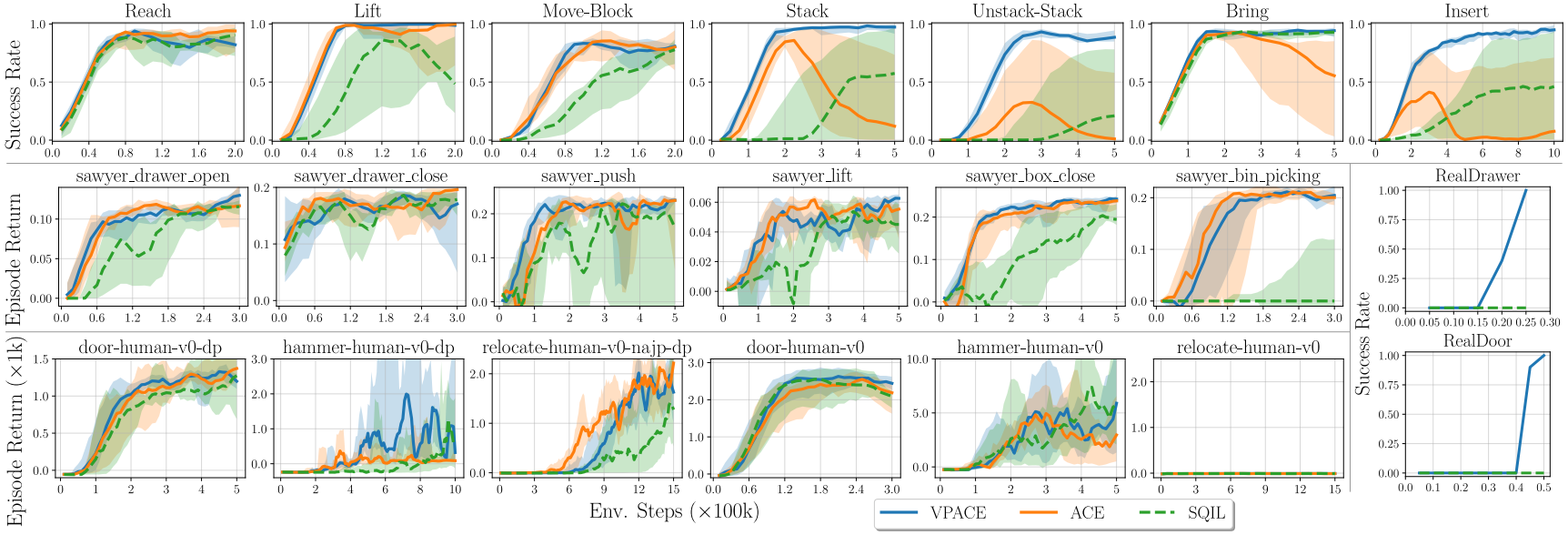


Figure 7.5: Sample efficiency performance plots for main task only for all main tasks. Performance is an interquartile mean (IQM) across 5 timesteps and 5 seeds with shaded regions showing 95% stratified bootstrap confidence intervals (Agarwal et al., 2021). Other baselines (RCE, DAC, SQIL+RND) are shown only in Fig. 7.6 for clarity. The use of ACE significantly improves upon SQIL, and the addition of VP (VPACE) resolves instability issues that occur in `Stack`, `Unstack-Stack`, `Bring`, `Insert`, and `hammer-human-v0-dp`.

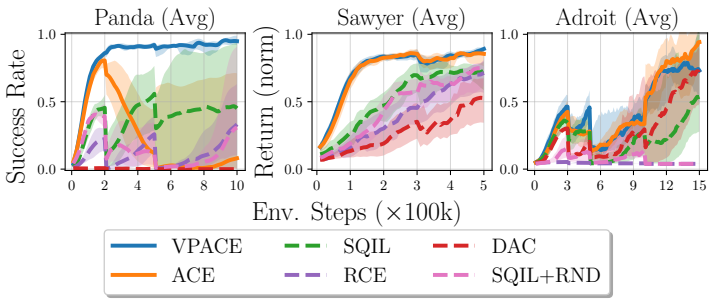


Figure 7.6: Average performance across all simulated main tasks, separated by environment, including baselines not shown in Fig. 7.5.

for 10 episodes for the real Panda tasks. Fig. 7.5 shows that VPACE has significant improvement over other approaches in both sample efficiency and final performance, particularly for the most difficult tasks, such as `Unstack-Stack` and `Insert`. We can observe that VPACE can consistently solve all tasks, while other EBC methods have significantly wider confidence intervals, especially in the Panda environment. Another notable observation is that SQIL+RND is unable to improve upon SQIL. We speculate that the exploration bonus term actually diverts the policy from solving the main task in order to maximize the intrinsic return.

ACE without VP

We also include ACE in Fig. 7.5 to address RQ2. In the simulated Panda environment, ACE performance significantly degrades as training continues. As elaborated on in Section 7.4.3, we hypothesize that this is correlated to value overestimation being exacerbated by executing multiple policies. For Sawyer and Adroit environments, VP does not provide benefit (i.e., VPACE and ACE have similar performance), which might be due to the tasks being simpler and having smaller observation spaces than the Panda tasks. Nonetheless, VP does not harm performance and including auxiliary tasks provides a clear benefit.

Real Robot Performance

The sample-efficiency improvement gained via VPACE allowed us to test it on real-life tasks (two right columns of Fig. 7.4), where the bottom right plots in Fig. 7.5 demonstrate that VPACE can solve real-world tasks in only a few hours of real execution time, whereas SQIL does not solve either task. Our real robot tasks execute at 5 Hz, and, with a small amount of time to allow for environment resetting, 1000 environment steps corresponds to roughly 4 real minutes, meaning `RealDrawer` is learned by VPACE in about 100 minutes, while `RealDoor` is learned in about 200 minutes. The real world examples of success, for both main tasks and auxiliary tasks, are collected in less than a minute. Our open-source code contains further implementation details for our real world setup.

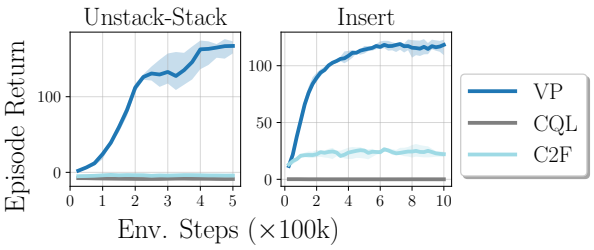


Figure 7.7: Results for different example-based approaches to regularizing Q estimates. We measure performance using return instead of success rate as the alternatives achieved no success at all.

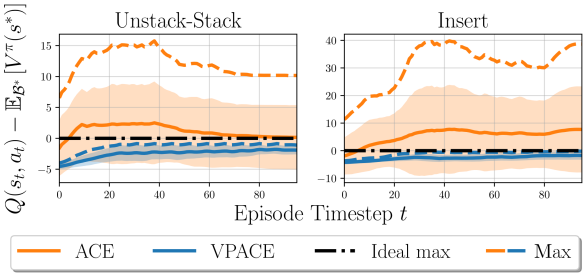


Figure 7.8: Difference between Q-values and expected value for example states for partially trained agents (snapshot from 300k environment steps) for a single episode rollout. The dashed lines show the maximum output across the seeds, and the shaded regions show the standard deviation between seeds. VP ensures that the maximum stays below 0, verifying that $y(s, s') \leq y(s^*, s^*)$.

7.4.3 Analysis and Additional Results

Q-Value Overestimation and Value Penalization

To verify that VP enforces $y(s, s') \leq y(s^*, s^*)$, we took snapshots of each learned `Unstack-Stack` and `Insert` agent, for both VPACE and ACE, at 300k steps and ran each policy for a single episode, recording per-timestep Q-values. Instead of showing Q-values directly, in Figs. 7.8 and 7.9, we show $Q(s_t, a_t) - \mathbb{E}_{s^* \sim \mathcal{B}^*}[V(s^*)]$, which should be at most 0 for $y(s, s') \leq y(s^*, s^*)$ to hold. ACE clearly violates $y(s, s') \leq y(s^*, s^*)$, while VPACE does not. Furthermore, Fig. 7.9 shows examples of the consequences of overestimated Q-values: for (s_t, a_t) pairs that appear to be out-of-distribution (OOD), $Q(s_t, a_t) - \mathbb{E}_{s^* \sim \mathcal{B}^*}[V(s^*)] > 0$, and the resulting policy reaches these states instead of the true goal.

To understand the importance of the $y(s, s') \leq y(s^*, s^*)$ constraint, we investigate other choices of regularization techniques (RQ3), in particular a L2 regularizer on Q-values (**C2F**, (James et al., 2022)) and a regularizer that penalizes the Q-values of out-of-distribution actions (**CQL**, (Kumar et al., 2020)). Fig. 7.7 indicates that using other existing regularization techniques does not enable ACE to perform well. We suspect that the L2 regularizer may be too harsh of a penalty on all learning, while (Kumar et al., 2020) does not have any penalization on the magnitudes of Q-values in general, meaning that they can potentially still have uncontrollable bootstrapping error.

Comparison to Full Demonstrations and True Rewards

A practitioner may ask how our method compares to using traditional forms of feedback, such as using full-expert trajectories and inverse reinforcement learning (IRL), or using RL with true sparse rewards

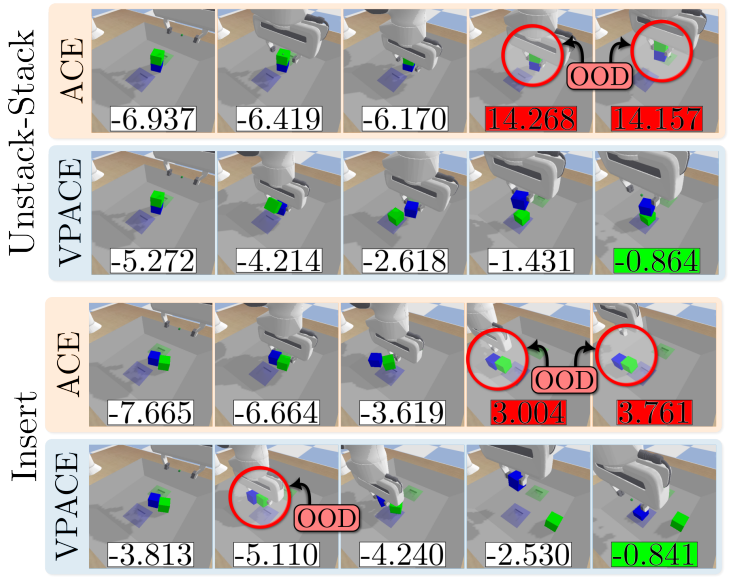


Figure 7.9: $Q(s_t, a_t) - \mathbb{E}_{\mathcal{B}^*}\left[V^\pi(s^*)\right]$ for parts of episodes used to generate the results from Fig. 7.8. It appears that the use of ACE without VP results in overestimated Q-values for out-of-distribution (OOD) (s_t, a_t) pairs (e.g., grasping both blocks simultaneously, or nearly inserting the incorrect block), resulting in highly suboptimal policies. VPACE, on the other hand, correctly learns to label a potentially OOD state with low value.

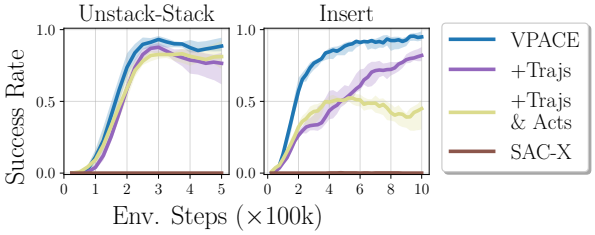


Figure 7.10: Results for changes in the form of feedback. SAC-X indicates the use of true environment sparse rewards only.

(RQ4). To test the former, we include two variants: a buffer containing both 200 examples and 200 (s, a) pairs from expert trajectories of states only (**+Trajs**), and the same for both states and actions (**+Trajs & Acts**), resulting in two variants of LfGP (Ablett et al., 2023) with VP. For sparse rewards, we use the provided sparse-reward function from the task (**SAC-X**), which is scheduled auxiliary control (Riedmiller et al., 2018) with VP, although VP is based on the maximum theoretical return in these experiments, rather than $y(s, s') \leq y(s^*, s^*)$ since s^* are not available to SAC-X.

Fig. 7.10 shows that the peak performance is *reduced* when learning with expert trajectories. We hypothesize that the divergence minimization objective leads to an effect, commonly seen with dense reward functions, known as reward hacking (Skalse et al., 2022). This effect was also previously shown to occur in inverse RL (Ablett et al., 2023). In short, the agent receives the same positive reward for reaching intermediate states in the demonstration as it does for reaching final states, despite the fact that intermediate states may not lead to the optimal final states. This result suggests that inverse RL/adversarial IL can be significantly improved by switching to example-based control. Learning with a sparse-reward function only does not accomplish the main task at all in either `Unstack-Stack` nor `Insert`, exhibiting substantially poorer performance than all other baselines. This result matches

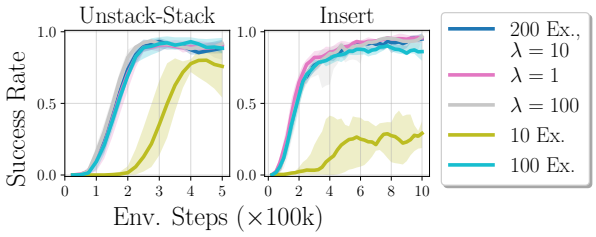


Figure 7.11: Variations of success example quantity and λ value.

previous research in which it has been shown that a demonstration buffer can significantly improve sample efficiency in RL (Nair et al., 2018; Kalashnikov et al., 2018). Our results show that an example buffer may provide a similar effect, and may even be a more optimal form of guidance than full demonstrations.

Expert Data Quantity and λ Value

Finally, to examine the robustness of our approach, we also compare various values for the quantity of examples of success (200 / task for main experiments) as well as the value of λ, which controls the strength of value penalization (10.0 for main experiments). Fig. 7.11 shows that changing the value of λ to 1 and 100 has no effect on performance. Dropping from 200 to 100 examples has only a minor negative impact on performance in `Insert`, while the drop to 10 examples has a significant negative effect. However, even with only 10 examples, performance on `Unstack-Stack` still far exceeds all other baselines.

7.5 Limitations

VPACE suffers from several limitations, though many of them are inherited from the use of reinforcement learning and learning from guided play. For an expansion of this section, including these inherited limitations, see Section B.5. In this work, we exclusively learn from numerical state data, rather than raw images. Numerical state data is not always available, since it requires an external system for state estimation, and it may allow for poorer transferability between tasks where the components of the state change. Finally, raw images may be required for tasks involving objects that are not rigid. As well, we claim that example distributions are easier to generate than full expert trajectories, but for certain tasks, generating these example distributions may also be challenging. Finally, tasks we investigate in this work have roughly unimodal example success state distributions, and our method may not gracefully handle multimodality.

7.6 Summary

In this chapter, we presented VPACE—value-penalized auxiliary control from examples, where we coupled scheduled auxiliary control with value penalization in the example-based setting to significantly improve learning efficiency. We showed that VPACE resolves Q overestimation, greatly improves the

sample efficiency of example-based control against a wide set of baselines, and shows improved performance compared with with other forms of feedback, including true sparse rewards and full trajectories. In fact, VPACE had improved performance over its counterpart that uses full trajectories of expert data, LfGP, from Chapter 6.

Unlike the methods presented in Chapters 3 to 6, VPACE does not directly solve the distribution shift problem presented in Section 2.4.2, because the example-only expert data provided to VPACE cannot be used to train a policy via supervised learning. VPACE, and other approaches to example-based control (EBC), can be interpreted as a way of abstracting away the distribution shift problem altogether; the policy finds its own way, possibly more optimally than a human would, to manipulate the environment and ultimately generate a distribution that matches the expert *example* distribution. Analyzing the theoretical efficiency and worst-case total cost of such an approach would therefore require different tools than those presented in Section 2.4.3, and is a possible direction for future work. Other opportunities for future work include the further investigation of learned approaches to scheduling, as well as the autonomous generation of auxiliary task definitions.

Chapter 8

Conclusion

This dissertation showed how we can modify imitation learning (IL) algorithms to make them both less susceptible to distribution shift and more sample efficient. Chapters 3 and 4 were applied to offline, supervised learning, Chapter 5 was applied to a hybrid between online and offline learning, and Chapters 6 and 7 were applied to online, inverse reinforcement learning (IRL). In Chapters 3 and 4, we showed that leveraging prior knowledge about a potential distribution shift can significantly improve policy performance. In Chapter 5, we showed that an interactive approach to supervised IL, which inherently resolves distribution shift, can be made less costly to an expert with a learned approach to predicting failures. In Chapter 6, we identified a deceptive reward problem in IRL, and resolved the problem with the addition of demonstrations of auxiliary tasks, in addition to the main task. Finally, in Chapter 7, we replaced full expert demonstrations in IRL with examples of success, fully resolving distribution shift but introducing a challenging exploration problem, which we resolve through the application of auxiliary examples and value-penalization.

The methods presented in Chapters 3 to 7 can be thought of as being on a continuum from the highest expert burden to the lowest. To apply a method similar to Chapter 3, the expert must have knowledge of the expected test distribution in order to deliberately widen the training distribution to match. In Chapter 4, the number of demonstrations required to be collected is significantly reduced (on the order of hundreds in Chapter 3 to the order of tens), because we are able to directly modify the demonstrations to improve their quality.

Since Chapter 3 and Chapter 4 both deal exclusively with behavioural cloning, they are still susceptible to inevitable distribution shift. In real applications, a practitioner applying Chapters 3 and 4 could alternately collect more demonstrations, evaluate the policy, and then collect more demonstrations to attempt to improve performance. As stated in Section 2.4.3, there is no theoretical guarantee in BC for the number of expert demonstrations that will be required to achieve expert-like performance, and much work has shown that simply scaling the number of demonstrations provides exponentially smaller gains in performance (Chapter 3, (Mandlekar et al., 2023; Ankile et al., 2024)). Naturally, DAgger (Ross et al., 2011) (see Sections 2.4.2 and 2.4.3) is the ideal choice for resolving this inevitable distribution shift, resulting in a worst-case error reduction from $\mathcal{O}(T^2\epsilon)$ to $\mathcal{O}(T\epsilon)$, but DAgger is not directly applicable to scenarios where the expert is a human being. In FIRE, presented in Chapter 5, we

create a single algorithm for unifying data collection and evaluation. This is combined with a scheme for providing corrective data, while simultaneously reducing the burden to the expert by predicting failures. The scheme is meant to be an alternative to DAgger for *human* experts, providing a similar benefit by ultimately resolving distribution shift.

Unfortunately, FIRE still provides no guarantees on how long the algorithm must run to achieve expert-like performance. While DAgger has a potential upper bound of $\mathcal{O}(T\log(T))$ iterations to achieve expert-like performance, FIRE introduces many suboptimalities that make it challenging to provide a theoretical upper bound on the number of iterations needed. Inverse reinforcement learning (IRL) provides a framework that enables a policy to autonomously explore and match the original expert distribution, theoretically resolving distribution shift with fully online learning. The expert only needs to provide a small number of demonstrations at the start of training. Our results and analysis in Chapter 6 show that IRL can be susceptible to finding a local optimum, where the policy distribution only partially matches expert distribution. LfGP, also presented in Chapter 6, resolves this susceptibility to local optimums by introducing exploration of semantically meaningful auxiliary tasks. Results in Chapter 6 even show that with *decreasing* amounts of expert data, we can still achieve expert-like performance, clearly demonstrating the reduction of expert burden compared with Chapters 3 to 5.

In Chapters 3 to 6, expert demonstrations are all assumed to be full trajectories, which introduces a variety of potential issues: full-trajectory demonstrations (i) require a means of teleoperation or kinesthetic teaching, which is not always possible, (ii) can be suboptimal, and (iii) can potentially be time-consuming to collect. VPACE, presented in Chapter 7, shows that the LfGP framework can be applied to the setting where we only have examples of success, which are far easier to collect and *guaranteed* to be optimal. The application of LfGP to the example-based setting does not work out of the box. VPACE requires a value penalty to resolve overestimated value functions, along with many algorithmic improvements, detailed in both Chapter 7 and Chapter B. Altogether, VPACE is the culmination of our efforts to produce an imitation learning method that achieves our two overarching goals: (i) it resolves distribution shift through autonomous distribution matching via fully online learning, and (ii) it puts minimal burden on the expert, requiring examples of success only and no further interaction by the expert.[1]

Our hope is that our approaches will help reduce the distribution shift and sample inefficiency that limit the applicability of IL and IRL to real world robotics. The remainder of this section includes a summary of the novel contributions associated with each chapter, a note on experimental task selection in this thesis, a short description of potential directions for future work, and a few general thoughts regarding the value of imitation learning in real world robotics.

8.1 Summary of Contributions

This dissertation's main novel contributions, summarized by chapter, are described in this section.

[1] For example, part of this dissertation was written while the policies for our real robot `RealDoor` and `RealDrawer` tasks were obtained through completely autonomous exploration, with no interaction from the author. The expert data for each of these tasks were obtained in less than a minute.

8.1.1 Multiview Manipulation from Demonstrations

We proposed an approach to modifying the initial state distribution of a mobile manipulator for manipulation tasks to increase robustness. Our contributions were:

1. a demonstration of how supervised imitation learning fails in a series of challenging contact-rich tasks in the multiview domain;
2. experimental validation that our approach allows a multiview policy to learn a far more robust policy with an equivalent quantity of data, on both simulated and real mobile manipulators, and is not penalized in the fixed-view versions of tasks;
3. experiments showing that multiview policies can succeed on out-of-distribution data;
4. an analysis of the features learned by a multiview policy, showing increased spatial correlation between views compared with fixed-view policies; and
5. open source data and code to reproduce our results.

8.1.2 Force-Matched Demonstrations

We presented approaches for generating force-matched demonstration data, in addition to a learned approach to multimodal visuotactile sensor mode switching. Our contributions were:

1. a method for tactile force matching: using the readings from a visuotactile sensor, we modify recorded kinesthetic teaching poses so that a new, replayed trajectory recovers the recorded forces *and* poses to generate a force-matched replay;
2. a novel method for switching between *visual* and *tactile* modes with a multimodal visuotactile sensor;
3. an extensive experimental evaluation of the benefits of including STS visual and tactile data as inputs to a multimodal control policy on a real robotic manipulator, especially when compared to a more standard eye-in-hand camera; and
4. open source code to encourage reproduction of our results.

8.1.3 Failure Identification for Interventions

We introduced **f**ailure **i**dentification to **r**educe **e**xpert burden (FIRE), a method for identifying failures in interactive IL. Our contributions were:

1. a novel method for predicting failures in interactive learning based on the output from a discriminator and a threshold that is updated automatically based on human preferences;
2. a statistically grounded improvement to our failure prediction approach based on Gaussian discriminant analysis (GDA);

3. experimental validation of the improvement of our approach over standard behavioural cloning in a variety of simulated robotic domains, including multiview ones introduced in Chapter 3; and

4. a comparison of our methods for failure prediction with existing approaches.

8.1.4 Learning from Guided Play

We proposed **l**earning **f**rom **g**uided **p**lay (LfGP), an application of the scheduled auxiliary control (Riedmiller et al., 2018) framework to adversarial imitation learning (AIL), a popular form of inverse reinforcement learning. Our contributions were:

1. a novel application of the scheduled auxiliary control framework to AIL that learns a reward and policy for a challenging main task by simultaneously learning rewards and policies for auxiliary tasks;

2. manipulation experiments in which we demonstrate that AIL fails, while LfGP significantly outperforms both AIL and behavioural cloning;

3. an extension of our method to transfer learning;

4. empirical analysis, including a simplified representative example and visualization of the learned models of LfGP and AIL, to better understand why AIL fails and how LfGP improves upon it; and

5. open source code, models, and data to encourage reproduction of our results.

8.1.5 Auxiliary Control from Examples

We introduced **v**alue-**p**enalized **a**uxiliary **c**ontrol from **e**xamples (VPACE), an extension of LfGP to example-based control (EBC). Our contributions were:

1. a demonstration showing that the introduction of the scheduled auxiliary control framework significantly improves exploration and learning efficiency in EBC;

2. a novel value-penalization method based on the expected value of the provided examples to resolve an overestimation problem that occurs as a result of naïvely applying the scheduled auxiliary control framework to EBC;

3. experimental results across four environments with 19 simulated and two real robot tasks showing improved sample efficiency and final performance of VPACE over EBC, inverse reinforcement learning, and an exploration bonus;

4. a comparison of VPACE to the use of full trajectories and true sparse rewards as feedback, observing that VPACE has higher sample efficiency than both; and

5. open source code and data to encourage reproduction of our results.

8.2 On Task Selection, Data-Driven Robotics, and Inductive Biases

Unfortunately, most of the tasks that are of practical interest (e.g., loading a dishwasher) are both challenging to model *and* challenging to collect sufficient data to learn, making them bad candidates for an academic research setting. Instead, we are forced to choose easier tasks that clearly have inductive biases that would make them easier to complete. If we were trying to, for example, generate policies with optimal performance in door opening (which is used as an example task in Chapters 3, 4 and 7), we should add further inductive biases such as knowledge about hinges or the difference between a knob and a handle. Generally, the more inductive biases one adds, the better performance one is likely to get with less data, but also the less likely that those assumptions will generalize to multiple tasks or domains. We avoid using inductive biases as much as possible, with the hope that our methods will generalize to more complicated tasks (where inductive biases may not be as obvious) given larger resources and the ability to collect more data.

8.3 Future Research Directions

The approaches to mitigating distribution shift in Chapters 3 and 4 are based on leveraging prior knowledge about how a testing distribution will differ from a training distribution. While the approach in Chapter 5 resolves the same problem without requiring prior knowledge, it can be extremely costly in practice, and it cannot easily be safely applied to many systems (e.g., with fast-moving robots). Instead, the ideal system might be to use a test regime that detects OOD data, recovers from it autonomously (but likely suboptimally), and then provides some kind of later feedback to the operator on what new data they should add to a system to improve the agent.

The approaches described in this dissertation are fairly clearly divided between behavioural cloning (BC) in Chapters 3 to 5 and inverse reinforcement learning (IRL) in Chapters 6 and 7. There have been attempts to combine these two ideas together (Peters and Kober, 2009; Nair et al., 2021; Lu et al., 2021), but it is far from a solved problem. BC has the large advantage of requiring no environmental exploration to learn a policy, while IRL is capable of potentially resolving distribution shift. The obvious solution would be to train a policy with BC and then gradually improve the policy with (I)RL (Rajeswaran et al., 2018), but in the off-policy setting, this continues to be an unsolved problem.

The problem becomes even more challenging if you would like to learn from an offline dataset of *nonoptimal* data (i.e., not necessarily from an expert, or not from an expert completing a single specific task). Learning policies from large batches of offline data is a very active research area (Fujimoto et al., 2019; Lynch and Sermanet, 2021; Levine et al., 2020), and its solution is arguably is the true promise of approaches to learning agents directly from data. Approaches explored in Chapters 6 and 7, in which multiple policies are learned to maximize similarity to different expert datasets, may have value in the fully offline regime.

8.4 On the Value of Imitation Learning to Robotics

Apart from a few sentences in the introduction and individual chapter motivation sections, this dissertation is written under the assumption that imitation learning (IL) provides a viable path for future work in robotics, and spends little time addressing questions of whether IL should be used at all. Practitioners are often focused on whether to use more traditional forms of perception and control or learning-based approaches, but in reality, all learning-based approaches typically use some degree of traditional perception or control. An often overlooked, but extremely crucial, part of IL (and RL) is the design of the observation and action space. This dissertation spent little time describing how and why observation and action spaces were chosen for each robotic platform, but a very large amount of experimental trial-and-error went into these choices. With the goal of creating a self-driving car or autonomous mobile manipulation robot, excluding the use of well-developed navigation tools for determining one's place in the world, for example, would be nonsensical. One could expect similarly poor results from excluding well-developed controls tools and modelling for converting motion commands to motor commands.

The true value of IL, then, is likely not in its ability to (partially) circumvent the need for highly accurate modelling, but rather in its ability to learn things that we *do not currently understand*. Consider a robot designed exclusively to open doors: it should be able to open any type of door, with the infinite possible configurations of door sizes, weights, appearances, handle types, hinge types, preferred door opening speeds, and other considerations. A single hand-crafted control policy with, correspondingly, infinite conditional statements, seems destined to fail. Is there a way to model our understanding of door opening in a way that covers every single one of these cases? Of course, there very well may be, but we certainly do not have that understanding now. In the meantime, we can collect as much data as possible of doors being opened, and train policies leverage this data, with the hope that the learned policies may generalize to new door poses and configurations. Any one of the techniques investigated in this dissertation could be directly applied in the case of a significantly increased dataset size, covering a much wider range of possible initial configurations. Many groups are, in fact, dramatically scaling up dataset sizes in robotics (not exclusively for door-opening, obviously), and the level of success of doing so remains to be seen. In any case, the principles of IL clearly have a place in the future of our autonomous systems, but its combination with our best work in traditional approaches to perception, navigation, planning, and control are likely where the ripest fruits lie.

Appendices

Appendix A

Learning from Guided Play – Additional Details

A.1 Simulated Panda Play Environment Details

A screenshot of our environment, simulated in PyBullet (Coumans and Bai, 2019), is shown in Fig. A.1. We chose this environment because we desired tasks that a) have a large distribution of possible initial states, representative of manipulation in the real world, b) have a shared observation/action space with several other tasks, allowing the use of auxiliary tasks and transfer learning, and c) require a reasonably long horizon and significant use of contact to solve. The environment contains a tray with sloped edges (to keep the blocks within the reachable workspace of the end-effector), as well as a green and a blue block, each of which is 4 cm $\times$ 4 cm $\times$ 4 cm and has a mass of 100 g. The dimensions of the lower part of the tray, before reaching the sloped edges, are 30 cm $\times$ 30 cm. The dimensions of the 'bring' boundaries (shaded blue and green regions) are 8 cm $\times$ 8 cm, while the dimensions of the insertion slots, which are directly in the center of each shaded region, are 4.1 cm $\times$ 4.1 cm $\times$ 1 cm. The boundaries for end-effector movement, relative to the tool center point that is directly between the gripper fingers, are a 30 cm $\times$ 30 cm $\times$ 14.5 cm box, where the bottom boundary is low enough to allow the gripper to interact with objects, but not to collide with the bottom of the tray.

See Table A.1 for a summary of our environment observations. In this work, we use privileged state information (e.g., block poses), but adapting our method to exclusively use image-based data is straightforward since we do not use hand-crafted reward functions as in (Riedmiller et al., 2018).

The environment movement actions are 3-DOF translational position changes, where the position change is relative to the current end-effector position. We leverage PyBullet's built-in position-based inverse kinematics function to generate joint commands. Our actions also contain a fourth dimension that corresponds to actuating the gripper. To allow for the use of policy models with exclusively continuous outputs, this dimension accepts any real number, with any value greater than 0 commanding the gripper to open, and any number less than 0 commanding it to close. Actions are supplied at a rate of 20 Hz, and each training episode is limited to 18 seconds, corresponding to 360 time steps per

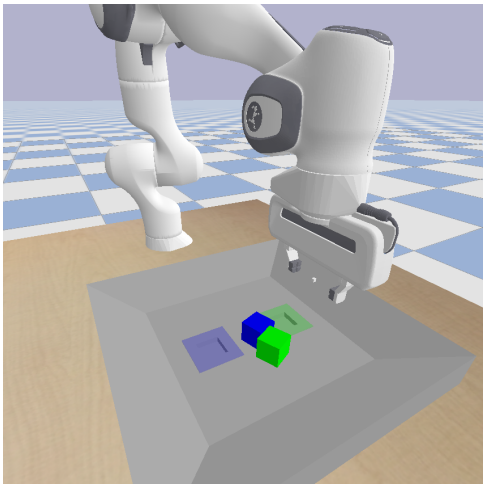


Figure A.1: An image of our multitask environment immediately after a reset has been carried out.

Table A.1: The components used in our environment observations, common to all tasks. Grip finger position is a continuous value from 0 (closed) to 1 (open).

Component	Dim	Unit	Privileged?	Extra info
EE pos.	3	m	No	rel. to base
EE velocity	3	m/s	No	rel. to base
Grip finger pos.	6	[0, 1]	No	current, last 2
Block pos.	6	m	Yes	both blocks
Block rot.	8	quat	Yes	both blocks
Block trans vel.	6	m/s	Yes	rel. to base
Block rot vel.	6	rad/s	Yes	rel. to base
Block rel to EE	6	m	Yes	both blocks
Block rel to block	3	m	Yes	in base frame
Block rel to slot	6	m	Yes	both blocks
Force-torque	6	N,Nm	No	at wrist
Total	**59**			

episode. For play-based expert data collection, we also reset the environment manually every 360 time steps. Between episodes, block positions are randomized to any pose within the tray, and the end-effector is randomized to any position between 5 and 14.5 cm above the tray, within the earlier stated end-effector bounds, with the gripper fully opened. The only exception to these initial conditions is during expert data collection and agent training of the Unstack-Stack task: in this case, the green block is manually set to be on top of the blue block at the start of the episode.

A.2 Reinforcement Learning Implementation Details

Our implementation builds on RL Sandbox (Chan, 2020), an open-source PyTorch (Paszke et al., 2019) framework for RL algorithms. For learning the discriminators, we follow DAC and apply a gradient penalty for regularization (Gulrajani et al., 2017; Kostrikov et al., 2019). We optimize the intentions via the reparameterization trick (Kingma and Welling, 2013), which allows for taking gradients through

the policy distributions. As is commonly done in deep RL, we use the clipped double Q-learning trick (Fujimoto et al., 2018) to mitigate overestimation bias (van Hasselt et al., 2016) and use a target network to mitigate learning instability (Mnih et al., 2015) when training the policies and Q-functions. We also learn the temperature parameter $\alpha_{\mathcal{T}}$ separately for each task $\mathcal{T}$ (see Section 5 of (Haarnoja et al., 2019) for more details on learning α). For Generative Adversarial Imitation Learning (GAIL), we use a common open-source PyTorch implementation (Kostrikov, 2018).

A.3 Procedure for Obtaining Experts

As stated, we used SAC-X (Riedmiller et al., 2018) to train models that we used for generating expert data. We used the same hyperparameters that we used for LfGP (see Table A.2), apart from the discriminator, which, of course, does not exist in SAC-X. See Section A.4 for details on the hand-crafted rewards that we used for training these models. For an example of gathering play-based expert data, please see our attached video.

We made two modifications to regular SAC-X to speed up learning. First, we pre-trained a Move-Object model before transferring this model to each of our main tasks, as we did in Section 5.3 of our main paper, since we found that SAC-X would plateau when we tried to learn the more challenging tasks from scratch. The need for this modification demonstrates another noteworthy benefit of LfGP—when training LfGP, main tasks could be learned from scratch, and generally in fewer time steps, than it took to train our experts. Second, during transfer to the main tasks, we used what we called a conditional weighted scheduler instead of a Q-Table: we defined weights for every combination of tasks, so that the scheduler would pick each task with probability $P(\mathcal{T}^{(h)}|\mathcal{T}^{(h-1)})$, ensuring that $\forall \mathcal{T}' \in \mathcal{T}_{\text{all}}, \sum_{\mathcal{T} \in \mathcal{T}_{\text{all}}} P(\mathcal{T}|\mathcal{T}') = 1$. The weights that we used were fairly consistent between main tasks, and can be found in our packaged code. The conditional weighted scheduler ensured that every task was still explored throughout the learning process, so that we would have high-quality experts for every auxiliary task in addition to the main task. This scheduler can be considered as a more complex alternative to the weighted random scheduler or the addition with handcrafted trajectories from our main paper, and again shows the flexibility of using a semantically-meaningful multitask policy with a common observation and action space.

A.4 Evaluation

As stated in our paper, we evaluated all algorithms by testing the mean output of the main task policy head in our environment and determining a success rate based on 50 randomly selected resets. These evaluation episodes were run for 360 time steps to match our training process, and if a condition for success was met within that time, they were recorded as a success. The rest of this section describes in detail how we evaluated 'success' for each of our main and auxiliary tasks.

As previously stated, we trained experts using a modified SAC-X (Riedmiller et al., 2018) that required us to define a set of reward functions for each task, which we include in this section. The

authors of (Riedmiller et al., 2018) focused on sparse rewards but also showed a few experiments in which dense rewards reduced the time to learn adequate policies, so we chose to use dense rewards. We note that many of these reward functions are particularly complex and required significant manual shaping effort, further motivating the use of an imitation learning scheme like the one presented in our paper. It is possible that we could have made do with sparse rewards, such as those used in (Riedmiller et al., 2018), but our compute resources made this impractical—for example, in (Riedmiller et al., 2018), their agent took 5000 episodes $\times$ 36 actors $\times$ 360 time steps $=$ 64.8 M time steps to learn their stacking task, which would have taken over a month of wall clock time on our fastest machine. To see the specific values used for the rewards and success conditions described in these sections, please review our code.

Unless otherwise stated, each of the success conditions in this section had to be held for 10 time steps, or 0.5 seconds, before being registered as a success. This choice was made to prevent registering a success when, for example, the blue block slipped off the green block during the Stack task.

A.4.1 Common

For each of these functions, we use the following common labels:

- p_b: blue block position,
- v_b: blue block velocity,
- a_b: blue block acceleration,
- p_g: green block position,
- p_e: end-effector tool center point position (TCP),
- p_s: center of a block pushed into one of the slots,
- g_1: (scalar) gripper finger 1 position,
- g_2: (scalar) gripper finger 2 position, and
- a_g: (scalar) gripper open/close action.

A block is flat on the tray when $p_{b,z} = 0$ or $p_{g,z} = 0$. To further reduce training time for SAC-X experts, all rewards were set to 0 if $\|p_b - p_e\| > 0.1$ and $\|p_g - p_e\| > 0.1$ (i.e., the TCP must be within 10 cm of either block). During training while using the Unstack-Stack variation of our environment, a penalty of -0.1 was added to each reward if $\|p_{g,z}\| > 0.001$ (i.e., there was a penalty to all rewards if the green block was not flat on the tray).

A.4.2 Stack and Unstack-Stack

The evaluation conditions for Stack and Unstack-Stack are identical, but in our Unstack-Stack experiments, the environment is manually set to have the green block start on top of the blue block.

Success

Using internal PyBullet commands, we check to see whether the blue block is in contact with the green block and is *not* in contact with either the tray or the gripper.

Reward

We include a term for checking the distance between the blue block and the spot above the the green block, a term for rewarding increasing distance between the block and the TCP once the block is stacked, a term for shaping lifting behaviour, a term to reward closing the gripper when the block is within a tight reaching tolerance, and a term for rewarding the opening the gripper once the block is stacked.

A.4.3 Bring/Insert

We use the same success and reward calculations for Bring and Insert, but for Bring the threshold for success is 3 cm, and for insert, it is 2.5 mm.

Success

We check that the distance between p_b and p_s is less than the defined threshold, that the blue block is touching the tray, and that the end-effector is *not* touching the block. For Insert, the block can only be within 2.5 mm of the insertion target if it is correctly inserted.

Reward

We include a term for checking the distance between the p_b and p_s and a term for rewarding increasing distance between p_b and p_e once the blue block is brought/inserted.

A.4.4 Open-Gripper/Close-Gripper

We use the same success and reward calculations for Open-Gripper and Close-Gripper, apart from inverting the condition.

Success

For Open-Gripper and Close-Gripper, we check to see if $a_g < 0$ or $a_g > 0$ respectively.

Reward

We include a term for checking the action, as we do in the success condition, and also include a shaping term that discourages high magnitudes of the movement action.

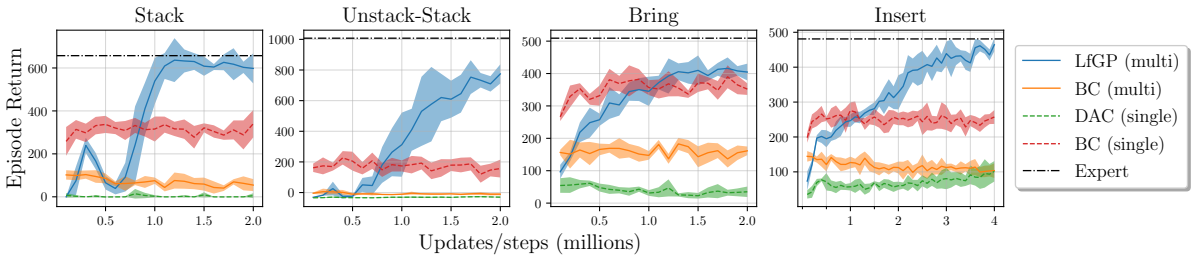


Figure A.2: Episode return for LfGP compared with all baselines. Shaded area corresponds to standard deviation.

A.4.5 Lift

Success

We check to see if $p_{b,z} > 0.06$.

Reward

We add a dense reward for checking the height of the block, but specifically also check that the gripper positions correspond to being closed around the block, so that the block does not simply get pushed up the edges of the tray. We also include a shaping term for encouraging the gripper to close when the block is reached.

A.4.6 Reach

Success

We check to see if $\|p_e - p_b\| < 0.015$.

Reward

We have a single dense term to check the distance between p_e and p_b.

A.4.7 Move-Object

For Move-Object, we changed the required holding time for success to 1 second, or 20 time steps.

Success

We check to see if the $v_b > 0.05$ and $a_b < 5$. The acceleration condition ensures that the arm has learned to move the block by following a smooth trajectory, rather than vigorously shaking it or continuously picking up and dropping it.

Reward

We include a velocity term and an acceleration penalty, as in the success condition, but also include a dense bonus for lifting the block.

A.5 Return Plots

As previously stated, we generated hand-crafted reward functions for each of our tasks for the purpose of training our SAC-X experts. Given that we have these rewards, we can also generate return plots corresponding to our results to add extra insight (see Fig. A.2 and Fig. A.3). The patterns displayed in these plots are, for the most part, quite similar to the success rate plots. One notable exception is that there is an eventual increase in performance when training DAC on Insert, indicating that, perhaps for certain tasks, DAC alone can eventually make progress. Nevertheless, it is clear that LfGP improves learning efficiency, and it is unclear whether DAC would plateau even if it was trained for a longer period.

A.6 Model Architectures and Hyperparameters

All the single-task models share the same network architectures and all the multitask models share the same network architectures. All layers are initialized using the PyTorch default methods (Paszke et al., 2019).

For the single-task variant, the policy is a fully-connected network with two hidden layers followed by ReLU activation. Each hidden layer consists of 256 hidden units. The output of the policy for LfGP and DAC is split into two vectors, mean $\hat{\mu}$ and variance $\hat{\sigma}^2$. For both variants of BC, only the mean $\hat{\mu}$ output is used. The vectors define a Gaussian distribution (i.e. $N(\hat{\mu}, \hat{\sigma}^2\mathbf{I})$, where $\mathbf{I}$ is the identity matrix). When computing actions, we squash the samples using the tanh function and bound the actions to be in range $[-1, 1]$, as done in SAC (Haarnoja et al., 2019). The variance $\hat{\sigma}^2$ is computed by applying a softplus function followed by a sum with an epsilon $\epsilon = 10^{-7}$ to prevent underflow: $\hat{\sigma}_i = \text{softplus}(\hat{x}_i) + \epsilon$. The Q-functions are fully-connected networks with two hidden layers followed by ReLU activations. Each hidden layer consists of 256 units. The output of the Q-function is a scalar corresponding to the value estimate given the current state-action pair. Finally, the discriminator is a fully-connected network with two hidden layers followed by tanh activations. Each hidden layer consists of 256 units. The output of the discriminator is a scalar logit to be used as an input to the sigmoid function. The sigmoid function output can be viewed as the probability of the current state-action pair coming from the expert distribution.

For the multitask variant, the policies and the Q-functions share their initial layers. There are two shared, fully-connected layers followed by ReLU activations. Each layer consists of 256 units. The output of the last shared layer is then fed into the policies and Q-functions. Each policy head and Q-function head corresponds to one task and has the same architecture: a two-layered fully-connected network followed by ReLU activations. The output of the policy head corresponds to the

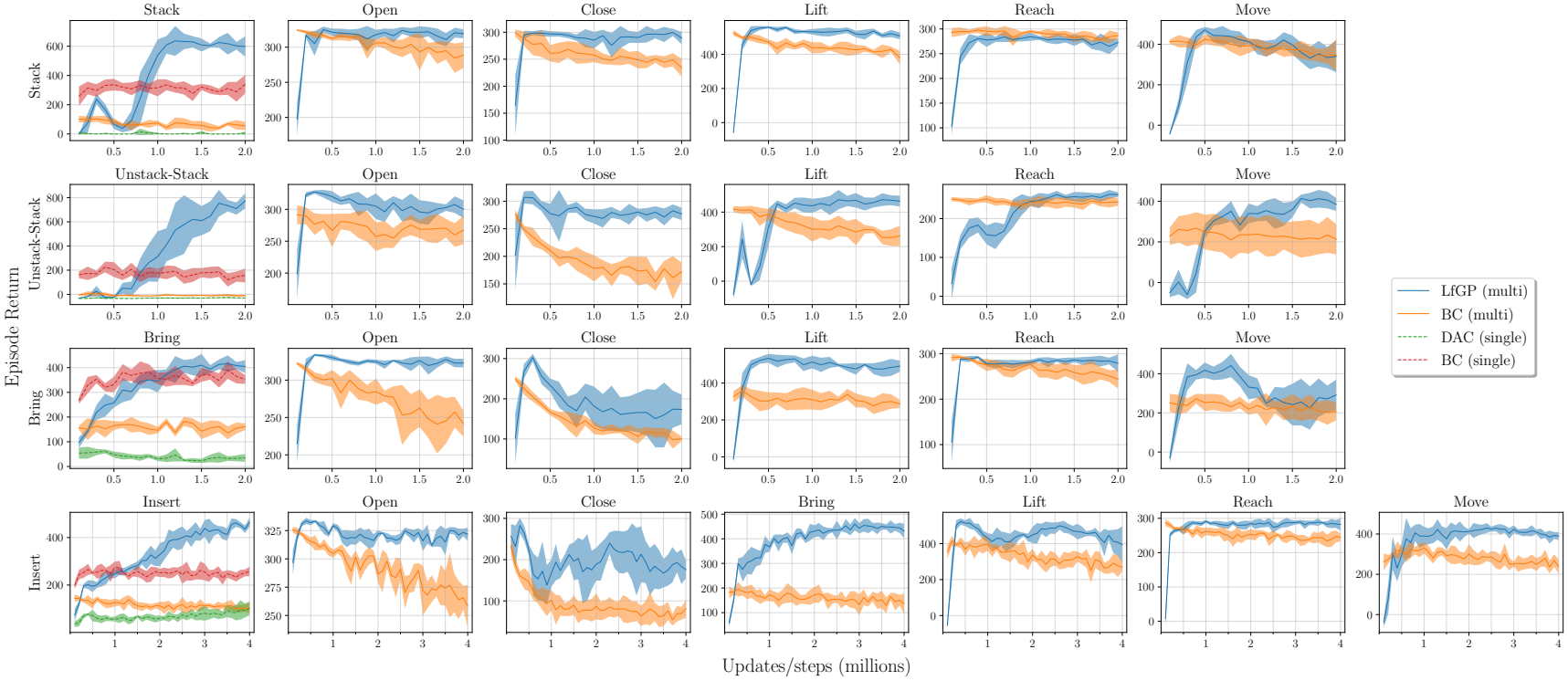


Figure A.3: Episode return for LfGP compared with multitask baselines on all tasks. Shaded area corresponds to standard deviation.

parameters of a Gaussian distribution, as described previously. Similarly, the output of the Q-function head corresponds to the value estimate. Finally, the discriminator is a fully-connected network with two hidden layers followed by tanh activations. Each hidden layer consists of 256 units. The output of the discriminator is a vector, where the i^{th} entry corresponds to the logit input to the sigmoid function for task $\mathcal{T}_i$. The i^{th} sigmoid function output corresponds to the probability of the current state-action pair coming from the expert distribution in task $\mathcal{T}_i$.

The hyperparameters for our experiments are listed in Table A.2 and Table A.4. In the early-stopping variant of BC, *overfit tolerance* refers to the number of full dataset training epochs without an improvement in validation error before we stop training. All models are optimized using Adam Optimizer (Kingma and Ba, 2015) with PyTorch default values, unless specified otherwise.

A.7 Open-Action and Close-Action Distribution Matching

There was one exception to the method we used for collecting our expert data. Specifically, our Open-Gripper and Close-Gripper tasks required additional considerations. It is worth reminding the reader that our Open-Gripper and Close-Gripper tasks were meant to simply open or close the gripper, respectively, while remaining reasonably close to either block. If we were to use the approach described above verbatim, the Open-Gripper and Close-Gripper data would contain no (s, a) pairs where the gripper actually released or grasped the block, instead immediately opening or closing the gripper while simply hovering near the blocks. Perhaps unsurprisingly, this was detrimental to our algorithm's performance: as one example, an agent attempting to learn Stack would, if Open-Gripper was selected while the blue block was held above the green block, move the grasped blue block *away* from the green block before dropping it on the tray. This behaviour, of course, is not what we would want, but it better matches an expert distribution when the environment is reset in between each task execution.

To mitigate this, our Open-Gripper data actually contain a mix of each of the other sub-tasks called for the first 45 time steps, followed by a switch to Open-Gripper, ensuring that the expert dataset contains some degree of block-releasing, with the trade-off being that 50% of the Open-Gripper expert data is specific to whatever the main task happens to be. We left this additional detail out of our main paper for clarity, since it corresponds to only a small portion of the expert data (every other auxiliary task was fully reused). Similarly, the Close-Gripper data calls Lift for 15 time steps before switching to Close-Gripper, ensuring that the Close-gripper dataset will contain a large proportion of data where the block is actually grasped. For the Closer-gripper data, however, this modification did still allow data to be reused between main tasks.

A.8 Attempted and Failed Experiments

In this section, we provide a list of experiments and modifications that did not improve performance, in addition to the alternatives that did.

1. **Pretraining with BC:** We attempted to pretrain LfGP using multitask BC, and then to transition

Table A.2: Hyperparameters for AIL algorithms across all tasks. Parameters that do not appear in the original version of DAC are shown in blue.

Algorithm	LfGP & DAC
Total Interactions	2M (4M for Insert)
Buffer Size	2M (4M for Insert)
Buffer Warmup	25k
Initial Exploration	50k
Evaluations per task	50
Evaluation frequency	100k interactions
Intention	
γ	0.99
Batch Size	256
Q Update Freq.	1
Target Q Update Freq.	1
π Update Freq.	1
Polyak Averaging	1e-4
Q Learning Rate	3e-4
π Learning Rate	1e-5
α Learning Rate	3e-4
Initial α	1e-2
Target Entropy	$-\dim(a) = -4$
Max. Gradient Norm	10
π Weight Decay	1e-2
Q Weight Decay	1e-2
$\mathcal{B}^E$ sampling proportion	0.1
$\mathcal{B}^E$ sampling decay	0.99999
Discriminator	
Learning Rate	3e-4
Batch Size	256
Gradient Penalty λ	10
Weight Decay	1e-2
$(s_T, \mathbf{0})$ sampling bias	0.95

to online learning with LfGP, but we found that this tended to produce significantly poorer final performance. Some existing work (Rajeswaran et al., 2018; Wu et al., 2020) has investigated transitioning from BC to online RL, but achieving this consistently, especially with off-policy RL, remains an open research problem.

2. **Handcrafted Open-Gripper/Close-Gripper policies:** Given the simplicity of designing a reward function in these two cases, a natural question is whether Open-Gripper and Close-Gripper could use hand-crafted reward functions, or even hand-crafted policies, instead of these specialized datasets. In our experiments, both of these alternatives proved to be quite detrimental to our algorithm.

3. **Penalizing Q values:** In our early experiments, we found that LfGP training progress was harmed by exploding Q values. This problem was particularly exacerbated when we added $\mathcal{B}^E$

Table A.3: Hyperparameters for LfGP schedulers.

Scheduler	Learned	WRS	WRS + HC
ξ	45	N/A	N/A
ϕ	0.6	N/A	N/A
Initial Temp.	360	N/A	N/A
Temp. Decay	0.9995	N/A	N/A
Min. Temp.	0.1	N/A	N/A
Main Task Rate	N/A	0.5	0.5
Handcraft Rate	N/A	N/A	0.5

Table A.4: Hyperparameters for BC algorithms (both single-task and multitask) across all tasks.

Version	Main Results	Early Stopping
Batch Size	256	
Learning Rate	1e-5	
Weight Decay	1e-2	
Total Updates	2M (4M for Insert)	N/A
Overfit Tolerance	N/A	100

sampling to our Q and π updates. It appears that this occurs because, at the beginning of training, the differences between discriminator outputs for expert data and non-expert data are so large that the bootstrap Q updates quickly jump to unrealistic values. We attempted to use various forms of Q penalties to resolve this, akin to Conservative Q Learning (CQL) (Kumar et al., 2020), but found that all of our modifications ultimately harmed final performance. Some of the things we tried, in addition to the CQL loss, were reducing γ (.95, .9), clipping Q losses to -5, +5, smooth L1 loss, huber loss, increased gradient penalty λ for D (50, 100), decreased reward scaling (.1), more discriminator updates per π/Q update (10), and weight decay in D only (as is done in (Orsini et al., 2021)). We ultimately resolved exploding Q values by (i) decreasing polyak averaging to a significantly lower value than is used in much other work (1e-4 as opposed to the SAC default of 5e-3), and (ii) adding in weight decay (with a significantly higher value used than is used in other work) to π, Q, and D training (which was required to not overfit with the reduced polyak averaging value). Without the added weight decay, performance started to plateau and eventually to decrease.

4. **Higher Update-to-Data (UTD) Ratio:** Recent work in RL has started increasing the UTD ratio (i.e., increasing the number of policy/Q updates per environment interaction), with the goal of improving environment sample efficiency (Chen et al., 2021). We were actually able to increase this from 1 to 2 and achieve a marginal improvement in environment sample efficiency, but this also nearly doubled the running time of our experiments, so we opted not to include this modification in our final results. Higher values of the UTD ratio also caused our Q values to explode.

Appendix B

Auxiliary Control from Examples – Additional Details

B.1 Reward Model Formulations

In this section, we investigate approaches to off-policy reinforcement learning (RL) studied in this work, modified to accommodate an unknown R and the existence of an example state buffer $\mathcal{B}^*$.

B.1.1 Learning a Reward Function

The most popular approach to reward modelling, known as inverse RL, tackles an unknown R by explicitly learning a reward model. Modern approaches under the class of adversarial imitation learning (AIL) algorithm aim to learn both the reward function and the policy simultaneously. In AIL, the learned reward function, also known as the discriminator, aims to differentiate the occupancy measure between the state-action distributions induced by expert and the learner. The learned reward function $\hat{R}$ is derived from the minimax objective (Kostrikov et al., 2019; Ho and Ermon, 2016; Fu et al., 2018a):

$$\mathcal{L}(D) = \mathbb{E}_{\mathcal{B}} \left[\log \left(1 - D(s) \right) \right] + \mathbb{E}_{\mathcal{B}^*} \left[\log \left(D(s^*) \right) \right], \tag{B.1}$$

where D attempts to differentiate the occupancy measure between the state distributions induced by $\mathcal{B}^*$ and $\mathcal{B}$. The output of $D(s)$ is used to define $\hat{R}(s)$, which is then used for updating the Q-function.

In example-based control, the discriminator D provides a smoothed label of success for states, thus its corresponding reward function can provide more density than a typical sparse reward function, making this approach an appealing choice. Unfortunately, a learned discriminator can suffer from the deceptive reward problem, as previously identified in (Ablett et al., 2023), and this problem is exacerbated in the example-based setting. In the following sections, we describe options to remove the reliance on separately learned discriminators.

B.1.2 Discriminator-Free Reward Labels with Mean Squared Error TD Updates

A simple alternative to using a discriminator as a reward model was initially introduced as soft-Q imitation learning (SQIL) in (Reddy et al., 2020). In standard AIL algorithms, D is trained separately from π and Q^π, where D is trained using data from both $\mathcal{B}$ and $\mathcal{B}^*$, whereas π and Q^π are trained using data exclusively from $\mathcal{B}$. However, most off-policy algorithms do not *require* this choice, and approaches such as (Ablett et al., 2023) and (Vecerik et al., 2018) train Q, and possibly π, using data from both $\mathcal{B}$ and $\mathcal{B}^*$. It is unclear why this choice is often avoided in AIL, but it might be because it can introduce instability due to large discrepancy in magnitudes for Q targets given data from $\mathcal{B}$ and $\mathcal{B}^*$. Sampling from both buffers, we can define $\hat{R}(s_t, a_t)$ to be labels corresponding to whether the data is sampled from $\mathcal{B}$ or $\mathcal{B}^*$—in SQIL, the labels are respectively 0 and 1. The full training objective resembles the minimax objective in Eq. (B.1) where we set $D(s^*) = 1$ and $D(s) = 0$.

The resulting reward function is the expected label $\hat{R}(s, a) = \mathbb{E}_{\hat{\mathcal{B}} \sim \text{Categorical}_s(\{\mathcal{B}, \mathcal{B}^*\})} \left[\mathbf{1}(\hat{\mathcal{B}} = \mathcal{B}^*) \right]$, where Categorical_s is a categorical distribution corresponding to the probability that s belongs to buffers $\mathcal{B}, \mathcal{B}^*$. Consequently, only successful states s^* yields positive reward and the corresponding optimal policy will aim to reach a successful state as soon as possible. If we further assume that s^* transitions to itself, then for policy evaluation with mean-squared error (MSE), we can write the temporal difference (TD) target, y, of Q-updates with:

$$y(s, s') = \gamma \mathbb{E}_{a'} \left[Q(s', a') \right], \tag{B.2}$$

$$y(s^*, s^*) = 1 + \gamma \mathbb{E}_{a'} \left[Q(s^*, a') \right], \tag{B.3}$$

where $(s, \cdot, s') \sim \mathcal{B}$, $a' \sim \pi(a'|s')$, and $s^* \sim \mathcal{B}^*$. This approach reduces complexity as we no longer explicitly train a reward model, and also guarantees discrimination between data from $\mathcal{B}$ and $\mathcal{B}^*$.

B.1.3 Discriminator-Free Reward Labels with Binary Cross Entropy TD Updates

Eysenbach et al. (2021) introduced recursive classification of examples (RCE), a method for learning from examples of success. RCE mostly follows the approach outlined above in Section B.1.2 but uses a weighted binary cross-entropy (BCE) loss with weights $1 + \gamma w$ for data from $\mathcal{B}$ and $1 - \gamma$ for data from $\mathcal{B}^*$. The TD targets are also changed from Eq. (B.2) and Eq. (B.3) to

$$y(s, s') = \gamma w(s') / (1 + \gamma w(s')), \tag{B.4}$$

$$y(s^*, s^*) = 1, \tag{B.5}$$

where $w(s') = V^\pi(s') / (1 - V^\pi(s'))$.

This approach can also be made closer to SQIL by removing the change in weights and by leaving Eq. (B.4) as Eq. (B.2). This makes it equivalent to SQIL, apart from removing bootstrapping from Eq. (B.3), and using a BCE instead of MSE for TD updates.

A major benefit of this approach, compared with the MSE TD updates in Section B.1.2, is that

$y(s, s') \leq y(s^*, s^*)$ is always enforced at every update, meaning that our approach to value penalization would provide no extra benefit. Nonetheless, our results from Section B.4 show that SQIL, even *without* value-penalization, almost always outperforms both RCE and SQIL with BCE loss (referred to SQIL-BCE here).

B.2 Additional Environment, Algorithm, and Implementation Details

The following sections contain further details of the environments, tasks, auxiliary tasks, algorithms, and implementations used in our experiments.

B.2.1 Additional Environment Details

Compared with the original Panda tasks from LfGP (Ablett et al., 2023), we switch from 20Hz to 5Hz control (finding major improvements in performance for doing so), improve handling of rotation symmetries in the observations, and remove the force-torque sensor since it turned out to have errors at low magnitudes. Crucially, these modifications did not require training new expert policies, since the same final observation states from the full trajectory expert data from (Ablett et al., 2023) remained valid. Compared with the original LfGP tasks, we also remove `Move-block` as an auxiliary task from `Stack`, `Unstack-Stack`, `Bring` and `Insert`, since we found a slight performance improvement for doing so, and add `Reach`, `Lift`, and `Move-block` as main tasks. The environment was otherwise identical to how it was implemented in LfGP, including allowing randomization of the block and end-effector positions anywhere above the tray, using delta-position actions, and using end-effector pose, end-effector velocity, object pose, object velocity, and relative positions in the observations. For even further details of the original environment, see (Ablett et al., 2023).

Since the Sawyer tasks from (Eysenbach et al., 2021; Yu et al., 2019) only contain end-effector position and object position by default, they do not follow the Markov property. To mitigate this, we train all algorithms in the Sawyer tasks with frame-stacking of 3 and add in gripper position to the observations, since we found that this, at best, improved performance for all algorithms, and at worst, kept performance the same. We validate that this is true by also performing experiments using the original code and environments from (Eysenbach et al., 2021), unmodified, where the results of RCE without these modifications are presented in Fig. B.7, with results comparable to or poorer than our own RCE results.

B.2.2 Delta-Position Adroit Hand Environments

The Adroit hand tasks from (Rajeswaran et al., 2018) use absolute positions for actions. This choice allows even very coarse policies, with actions that would be unlikely to be successful in the real world, to learn to complete `door-human-v0` and `hammer-human-v0`, and also makes the intricate exploration required to solve `relocate-human-v0` very difficult. Specifically, VPACE-SQIL and several other baselines achieve high return in `door-human-v0` and `hammer-human-v0`, but the learned policies use

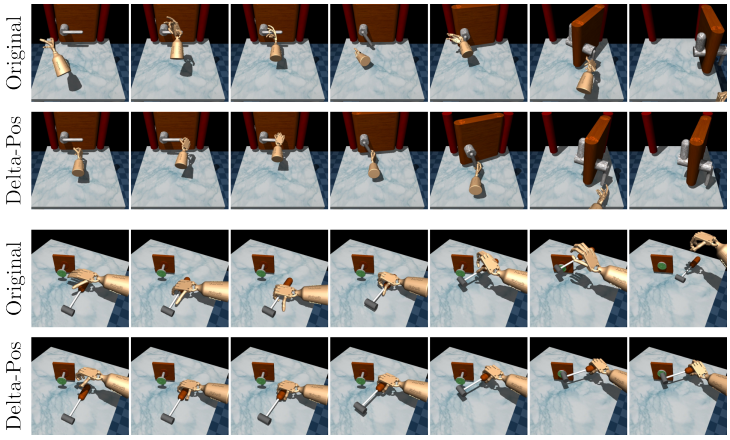


Figure B.1: Learned VPACE-SQIL policies at the final training step for, from top to bottom, `door-human-v0`, `door-human-v0-dp`, `hammer-human-v0`, and `hammer-human-v0-dp`. Although the original versions of the environments are solved, the absolute position action space allows policies to execute very coarse actions that exploit the simulator (above, hitting the handle without grasping it and throwing the hammer, respectively), and would almost certainly cause damage to a real environment.

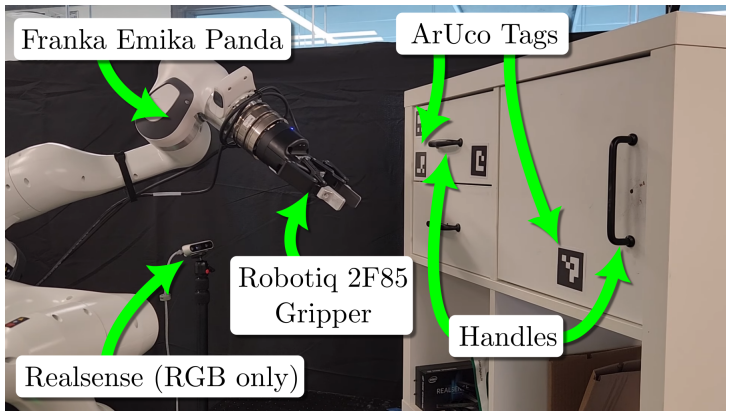


Figure B.2: Experimental setup for our real environment and our `RealDrawer` and `RealDoor` tasks. The robot has a force-torque sensor attached, but it is not used for our experiments.

unrealistic actions that exploit simulator bugs. As well, no methods are able to achieve any return in `relocate-human-v0` in 1.5M environment steps.

In the interest of solving `relocate-human-v0` and learning more skillful policies, we generated modified versions of these environments with delta-position action spaces. Furthermore, in `relocate-human-v0-najp-dp`, the action space rotation frame was changed to be in the palm, rather than at the elbow, and, since relative positions between the palm, ball, and relocate goal are included as part of the state, we removed the joint positions from the state. In our experiments, these modified environments were called `door-human-v0-dp`, `hammer-human-v0-dp`, and `relocate-human-v0-najp-dp`. See Fig. B.1 for a comparison of learned policies in each version of these environments.

B.2.3 Real World Environment Details

Fig. B.2 shows our experimental platform and setup for our two real world tasks. In both `RealDrawer` and `RealDoor`, the observation space contains the end-effector position, an ArUco tag (Garrido-Jurado et al., 2014) to provide drawer or door position (in the frame of the RGB camera; we do not perform extrinsic calibration between the robot and the camera), and the gripper finger position. Observations also include two stacked frames, in lieu of including velocity, to better follow the Markov property. The action space in both contains delta-positions and a binary gripper command for opening and closing. The action space for `RealDrawer` is one-dimensional (allowing motion in a line), while the action space for `RealDoor` is two-dimensional (allowing motion in a plane). The initial state distribution for `RealDrawer` allows for initializing the end-effector anywhere within a 10 cm line approximately 25 cm away from the drawer handle when closed. For `RealDoor`, the initial state distribution is a 20 cm $\times$ 20 cm square, approximately 20cm away from the door handle when closed. Actions are supplied at 5 Hz.

For both environments, for evaluation only, success is determined by whether the drawer or door is fully opened, as detected by the absolute position of the ArUco tag in the frame of the RGB camera. Our robot environment code is built on Polymetis (Lin et al., 2021), and uses the default hybrid impedance controller that comes with the library. To reduce environmental damage from excessive forces and torques, we reduced Cartesian translational stiffness in all dimensions from 750 N/m to 250 N/m, and the force and torque limits in all dimensions from 40 N and 40 Nm to 20 N and 20 Nm.

B.2.4 Additional Task Details

Success examples for the Panda environments were gathered by taking s_T from the existing datasets provided by (Ablett et al., 2023). Success examples for main tasks from the Sawyer environments were generated using the same code from (Eysenbach et al., 2021), in which the success examples were generated manually given knowledge of the task. Auxiliary task data was generated with a similar approach. Success examples for the Adroit hand environments were generated from the original human datasets provided by (Rajeswaran et al., 2018). Success examples for our real world tasks were generated by manually moving the robot to a small set of successful positions for each auxiliary task and main task. All Panda main tasks use the the auxiliary tasks *release*, *reach*, *grasp*, and *lift*.

There are two specific nuances that were left out of the main text for clarity and brevity: (i) the `Reach` main task only uses *release* as an auxiliary task (since it also acts as a "coarse" reach), and (ii) half of the *release* dataset for each task is specific to that task (e.g., containing insert or stack data), as was the case in the original datasets from (Ablett et al., 2023). For the Sawyer, Hand, and real Panda environments, because the observation spaces are not shared, each task has its own separate *reach* and *grasp* data.

B.2.5 Additional Algorithm Details

Algorithm 4 Value-Penalized Auxiliary Control from Examples (VPACE)

Input: Example state buffers $\mathcal{B}^*_{\text{main}}, \mathcal{B}^*_1, \ldots, \mathcal{B}^*_K$, scheduler period ξ, sample batch size N and N^*, and discount factor γ

Parameters: Intentions $\pi_{\mathcal{T}}$ with corresponding Q-functions $Q_{\mathcal{T}}$ (and optionally discriminators $D_{\mathcal{T}}$), and scheduler π_S (e.g. with Q-table Q_S)

Initialize replay buffer $\mathcal{B}$
for $t = 1, \ldots,$ **do**
  # Interact with environment
  For every ξ steps, select intention $\pi_{\mathcal{T}}$ using π_S
  Select action a_t using $\pi_{\mathcal{T}}$
  Execute action a_t and observe next state s'_t
  Store transition $\langle s_t, a_t, s'_t \rangle$ in $\mathcal{B}$

  # Optionally update discriminator $D_{\mathcal{T}'}$ for each task $\mathcal{T}'$
  Sample $\{s_i\}_{i=1}^{N} \sim \mathcal{B}$
  for each task $\mathcal{T}'$ **do**
    Sample $\{s^*_i\}_{i=1}^{N^*} \sim \mathcal{B}^*_k$
    Update $D_{\mathcal{T}'}$ following Eq. (B.1) using GAN + Gradient Penalty
  end for

  # Update intentions $\pi_{\mathcal{T}'}$ and Q-functions $Q_{\mathcal{T}'}$ for each task $\mathcal{T}'$
  Sample $\{(s_i, a_i)\}_{i=1}^{N} \sim \mathcal{B}$
  for each task $\mathcal{T}'$ **do**
    Sample $\{s^*_i\}_{i=1}^{N^*} \sim \mathcal{B}^*_{\mathcal{T}'}$
    Sample $a^*_i \sim \pi_{\mathcal{T}'}(s^*_i)$ for $i = 1, \ldots, N^*$
    Compute rewards $\hat{R}_{\mathcal{T}'}(s_i)$ and $\hat{R}_{\mathcal{T}'}(s^*_j)$ for $i = 1, \ldots, N$ and $j = 1, \ldots, N^*$
    # Compute value penalization terms, see Section B.2.6
    Compute $Q^{\pi_{\mathcal{T}'}}_{\max} = \frac{1}{N^*} \sum_{j=1}^{N^*} Q(s^*_j, a^*_j)$
    Compute $Q^{\pi_{\mathcal{T}'}}_{\min} = \min(\wedge_{i=1}^{N} \hat{R}_{\mathcal{T}'}(s_i), \wedge_{j=1}^{N^*} \hat{R}_{\mathcal{T}'}(s^*_j))/(1-\gamma)$
  end for
  Update π following Eq. (B.6)
  Update Q following Eq. (B.7) with value penalization Eq. (B.8)

  # *Optional* Update learned scheduler π_S
  if at the end of effective horizon **then**
    Compute main task return $G_{\mathcal{T}_{\text{main}}}$ using reward estimate from D_{main}
    Update π_S (e.g. update Q-table Q_S using EMA and recompute Boltzmann distribution)
  end if
end for

Algorithm 4 shows a summary of VPACE, built on LfGP (Ablett et al., 2023) and SAC-X (Riedmiller et al., 2018). As a reminder, our multi-policy objective function is

$$\mathcal{L}(\pi; \mathcal{T}) = \mathbb{E}_{\mathcal{B}, \pi_{\mathcal{T}}} \left[Q_{\mathcal{T}}(s, a) \right], \tag{B.6}$$

Algorithm	Value Penalization	Sched. Aux. Tasks	Reward Model	TD Error Loss	Source
VPACE	✓	✓	SQIL	MSE	Ours
ACE	✗	✓	SQIL	MSE	Ours
VPACE-DAC	✓	✓	DAC	MSE	Ours
ACE-RCE	✗	✓	RCE	BCE	Ours
VP-SQIL	✓	✗	SQIL	MSE	Ours
DAC	✗	✗	DAC	MSE	Ours
SQIL	✗	✗	SQIL	MSE	Ours
RCE	✗	✗	RCE	BCE	Ours
SQIL+RND	✗	✗	SQIL	MSE	Ours
RCE (theirs)	✗	✗	RCE	BCE	(Eysenbach et al., 2021)
SQIL-BCE	✗	✗	SQIL	BCE	(Eysenbach et al., 2021)

Table B.1: Major differences between algorithms studied in this work. MSE refers to mean squared error, while BCE refers to binary cross entropy.

our multi-Q objective function is

$$\mathcal{L}(Q;\mathcal{T}) = \mathbb{E}_{\mathcal{B},\pi_\mathcal{T}}\left[(Q_\mathcal{T}(s,a) - y_\mathcal{T}(s,s'))^2\right] + \mathbb{E}_{\mathcal{B}^*_\mathcal{T},\pi_\mathcal{T}}\left[(Q_\mathcal{T}(s^*,a) - y_\mathcal{T}(s^*,s^*))^2\right], \tag{B.7}$$

and our value-penalized update is

$$\mathcal{L}^\pi_{\text{pen}}(Q) = \lambda\mathbb{E}_\mathcal{B}[(\max(Q(s,a) - Q^\pi_{\text{max}}, 0))^2 + (\max(Q^\pi_{\text{min}} - Q(s,a), 0))^2], \tag{B.8}$$

where

$$Q^\pi_{\text{min}} = \hat{R}_{\text{min}}/(1-\gamma), \tag{B.9}$$

and

$$Q^\pi_{\text{max}} = \mathbb{E}_{\mathcal{B}^*}\left[V^\pi(s^*)\right]. \tag{B.10}$$

Table B.1 shows a breakdown of some of the major differences between all of the algorithms studied in this work.

B.2.6 Additional Implementation Details

In this section, we list some specific implementation details of our algorithms. We only list parameters or choices that may be considered unique to this work, but a full list of all parameter choices can be found in our code. We also provide the VPACE pseudocode in Algorithm 4, with blue text only applying to learned discriminator-based reward functions (see Section B.1 for various reward models).

Whenever possible, all algorithms and baselines use the same modifications. Table B.2 also shows our choices for common off-policy RL hyperparameters as well as choices for those introduced by this work.

DAC reward function: for VPACE-DAC and VP-DAC, although there are many options for re-

Table B.2: Hyperparameters shared between all algorithms, unless otherwise noted.

General	
Total Interactions	Task-specific (see Section B.3)
Buffer Size	Same as total interactions
Buffer Warmup	5k
Initial Exploration	10k
Evaluations per task	50 (Panda), 30 (Sawyer/Adroit)
Evaluation frequency	25k (Panda), 10k (Sawyer/Adroit)
Learning	
γ	0.99
$\mathcal{B}$ Batch Size	128
$\mathcal{B}^*$ Batch Size	128
Q Update Freq.	1 (sim), 4 (real)
Target Q Update Freq.	1 (sim), 4 (real)
π Update Freq.	1 (sim), 4 (real)
Polyak Averaging	1e-3
Q Learning Rate	3e-4
π Learning Rate	3e-4
D Learning Rate	3e-4
α Learning Rate	3e-4
Initial α	1e-2
Target Entropy	$-\dim(a)$
Max. Gradient Norm	10
Weight Decay (π, Q, D)	1e-2
D Gradient Penalty	10
Reward Scaling	0.1
SQIL Labels (Section B.2.6	$(-1, 1)$
Expert Aug. Factor (Section B.2.6)	0.1
Value Penalization (VP)	
λ	10
Q^π_{max}, Q^π_{min} num. filter points (Section B.2.6)	50
Auxiliary Control (ACE) Scheduler (Section B.2.6)	
Num. Periods	8 (Panda), 5 (Sawyer/Adroit)
Main Task Rate	0.5 (Panda), 0.0 (Sawyer/Adroit)
Handcraft Rate	0.5 (Panda), 1.0 (Sawyer/Adroit)

ward functions that map D to $\hat{R}$ (Ghasemipour et al., 2019), following (Ablett et al., 2023; Kostrikov et al., 2019; Fu et al., 2018a), we set the reward to $\hat{R}_{\mathcal{T}}(s) = \log\left(D_{\mathcal{T}}(s)\right) - \log\left(1 - D_{\mathcal{T}}(s)\right)$.

n-step returns and entropy in TD error: following results from (Eysenbach et al., 2021), we also add n-step returns and remove the entropy bonus in the calculation of the TD error for all algorithms in all Sawyer and Adroit environments, finding a significant performance gain for doing so.

Absorbing states and terminal states: for all algorithms, we do not include absorbing states (introduced in (Kostrikov et al., 2019)) or terminal markers (sometimes referred to as "done"), since we found that both of these additions cause major bootstrapping problems when environments only terminate on timeouts, and timeouts do not necessarily indicate failure. Previous work supports bootstrapping on terminal states when they are caused by non-failure timeouts (Pardo et al., 2018).

SQIL labels for policy data: The original implementation of SQIL uses labels of 0 and 1 TD updates in Eq. (B.2) and Eq. (B.3), respectively. We found that changing the label for Eq. (B.2) from 0 to -1 improved performance.

Reward Scaling of .1: we use a reward scaling parameter of .1 for all implementations. Coupled with a discount rate $\gamma = 0.99$ (common for much work in RL), this sets the expected minimum and maximum Q values for SQIL to $\frac{-.1}{1-\gamma} = -10$ and $\frac{.1}{1-\gamma} = 10$.

No multitask weight sharing: intuitively, one may expect weight sharing to be helpful for multitask implementations. We found that it substantially hurt performance, so all of our multitask methods do not share weights between tasks or between actor and critic. However, the multitask discriminator in VPACE-DAC *does* have an initial set of shared weights due to its significantly poorer performance without this choice.

$\mathcal{B}^*$ sampling for Q: in SQIL, DAC and RCE, we sample from both $\mathcal{B}$ and $\mathcal{B}^*$ for Q updates, but not for π updates (which only samples from $\mathcal{B}$). The original DAC implementation in (Kostrikov et al., 2019) only samples $\mathcal{B}^*$ for updating D, sampling only from $\mathcal{B}$ for updating Q.

All other architecture details, including neural network parameters, are the same as (Ablett et al., 2023), which our own implementations are built on top of. Our code is built on top of the code from (Ablett et al., 2023), which was originally built using (Chan, 2020).

Maintaining Q^{π}_{max}, Q^{π}_{min} Estimates for Value Penalization

Our approach to value penalization requires maintaining estimates for or choosing Q^{π}_{max} and Q^{π}_{min}. In both DAC and SQIL, the estimate of Q^{π}_{max} comes from taking the mini-batch of data from $\mathcal{B}^*$, passing it through the Q function, taking the mean, and then using a median moving average filter to maintain an estimate. The "Q^{π}_{max}, Q^{π}_{min} num. filter points" value from Table B.2 refers to the size of this filter. We chose 50 and used it for all of our experiments. We set Q^{π}_{min} to

$$Q^{\pi}_{\text{min}} = \frac{\text{rew. scale} \times \min(\hat{R}(s))}{1 - \gamma}, \tag{B.11}$$

where in SQIL, $\min(\hat{R}(s)) = \hat{R}(s)$ is set to 0 or -1, and in DAC, we maintain an estimate of the minimum learned reward $\min(\hat{R})$ using a median moving average filter with the same length as the

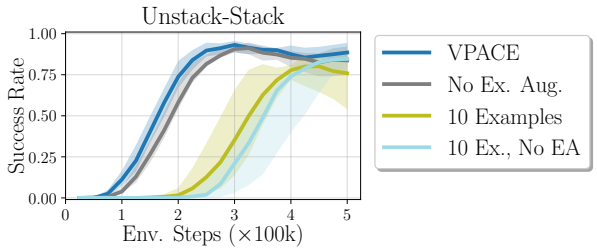


Figure B.3: Variations of example augmentation. Our example augmentation scheme provides a small but noticeable bump in performance, which is magnified with when fewer expert examples are used.

one used for Q^{π}_{max} .

Scheduler Choices

Our *ACE* algorithms use the same approach to scheduling from (Ablett et al., 2023). Specifically, we use a weighted random scheduler (WRS) combined a small set of handcrafted high-level trajectories. The WRS forms a prior categorical distribution over the set of tasks, with a higher probability mass $p_{\mathcal{T}_{\text{main}}}$ (Main Task Rate in Table B.2) for the main task and $\frac{p_{\mathcal{T}_{\text{main}}}}{K}$ for all other tasks. Additionally, we choose whether to uniformly sample from a small set of handcrafted high-level trajectories, instead of from the WRS, at the *Handcraft Rate* from Table B.2.

Our selections for handcrafted trajectories are quite simple, and reusable between main tasks within each environment. In the Panda tasks, there are eight scheduler periods per episode and four auxiliary tasks (*reach*, *grasp*, *lift*, *release*), and the handcrafted trajectory options are:

1. *reach*, *lift*, *main*, *release*, *reach*, *lift*, *main*, *release*
2. *lift*, *main*, *release*, *lift*, *main*, *release*, *lift*, *main*
3. *main*, *release*, *main*, *release*, *main*, *release*, *main*, *release*

In the Sawyer and Adroit environments, we actually found that the WRS was unnecessary to efficiently learn the main task, and simply used two handcrafted high-level trajectories. In these environments, there are five scheduler periods per episode and two auxiliary tasks (*reach*, *grasp*), and the handcrafted trajectory options are:

1. *reach*, *grasp*, *main*, *main*, *main*
2. *main*, *main*, *main*, *main*, *main*

Expert Data Augmentation

We added a method for augmenting our expert data to artificially increase dataset size. The approach is similar to other approaches that simply add Gaussian or uniform noise to data in the buffer (Yarats et al., 2022; Sinha et al., 2021). In our case, we go one step further than the approach from (Sinha et al.,

2021), and first calculate the per-dimension standard deviation of each observation in $\mathcal{B}^*$, scaling the Gaussian noise added to each dimension of each example based on the dimension's standard deviation. For example, if a dimension in $\mathcal{B}^*$ has zero standard deviation (e.g., in `Insert`, the pose of the blue block is always the same), it will have no noise added by our augmentation approach. The parameter "Expert Aug. Factor" from Table B.2 controls the magnitude of this noise, after our per-dimension normalization scheme.

In Fig. B.3, we show the results of excluding expert augmentation, where there is a clear, if slight, performance decrease when it is excluded, which is even more pronounced with a smaller $\mathcal{B}^*_\mathcal{T}$ size. All methods and baselines from our own implementation use expert data augmentation.

Other Ablation Details

In our ablation experiments from, we included three baselines with full trajectory data, in addition to success examples. We added 200 (s, a) pairs from full expert trajectories to make datasets comparable to the datasets from (Ablett et al., 2023), where they used 800 expert (s, a) pairs, but their environment was run at 20Hz instead of 5Hz, meaning they needed four times more data to have roughly the same total number of expert trajectories. We generated these trajectories using high-performing policies from our main experiments, since the raw trajectory data from (Ablett et al., 2023) would not apply given that we changed the control rate from 20Hz to 5Hz.

Real Panda Implementation Details

While most of the design choices in Section B.2.6 apply to all environments tested, our real Panda environment had some small specific differences, mostly due to the complications of running reinforcement learning in the real world. We list the differences here, but for an exhaustive list, our open source code contains further details.

Maximum episode length: The maximum episode length for both `RealDrawer` and `RealDoor` is 1000 steps, or 200 seconds in real time. This was selected to reduce how often the environment had to be reset, which is time consuming. Running episodes for this long, and executing actions at 5 Hz, our environments complete 5000 environment steps in roughly 20 minutes. The extra time is due to the time to reset the environment after 1000 steps or after a collision. VPACE took approximately 100 minutes to learn to complete `RealDrawer` consistently, and about 200 minutes to learn to complete the more difficult `RealDoor`.

Shorter initial exploration: To attempt to learn the tasks with fewer environment samples, we reduce buffer warmup to 500 steps, and initial random exploration to 1000 steps.

Frame stack: For training, we stacked two regular observations to avoid state aliasing.

Ending on success: We ended episodes early if they were determined to be successful at the main task only. Although this is not necessary for tasks to be learned (and this information was *not* provided to the learning algorithm), it gave us a way to evaluate training progress.

Extra gradient steps: To add efficiency during execution and training, we completed training steps during the gap in time between an action being executed and an observation being gathered. In-

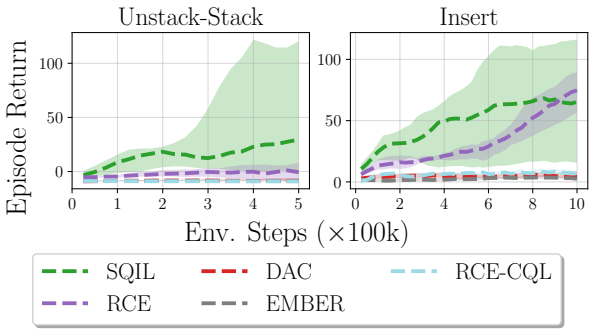


Figure B.4: Results for different reward models for EBC, excluding the use of both value penalization (VP) and auxiliary control from examples (ACE).

stead of completing a single gradient step at this time, as is the case for standard SAC (and VPACE), we completed four gradient steps, finding in simulated tasks that this gave a benefit to learning efficiency without harming performance. Previous work (Ablett et al., 2023), (Chen et al., 2021) has found that increasing this update rate can reduce performance, but we hypothesize that our value penalization scheme helps mitigate this issue.

Collisions: If the robot is detected to have exceeded force or torque limits (20 N and 20 Nm in our case, respectively), the final observation prior to collision is recorded, and the environment is immediately reset. There are likely more efficient ways to handle such behaviour, but we did not investigate any in this work.

B.3 Additional Performance Results

In this section, we expand upon our performance results and our Q-value overestimation analysis.

B.3.1 Expanded Main Task Performance Results

Fig. B.5 shows expanded results for our simulated environments, with each baseline shown for each individual environment.

B.3.2 ACE Reward Model Comparison

The SAC-X framework, which we describe as ACE when used with example states only, is agnostic to the choice of reward model (see Section B.1). Therefore, we also completed experiments in each of our simulated environments with DAC and with RCE as the base reward model, instead of SQIL, which was used for all of our main VPACE results. The results are shown in Fig. B.6, where it is clear that VPACE with SQIL learns more efficiently than VPACE with DAC and with higher final performance than ACE with RCE. We do not use value penalization (VP) for RCE, since RCE uses a classification loss for training Q that would not benefit from the use of value penalization.

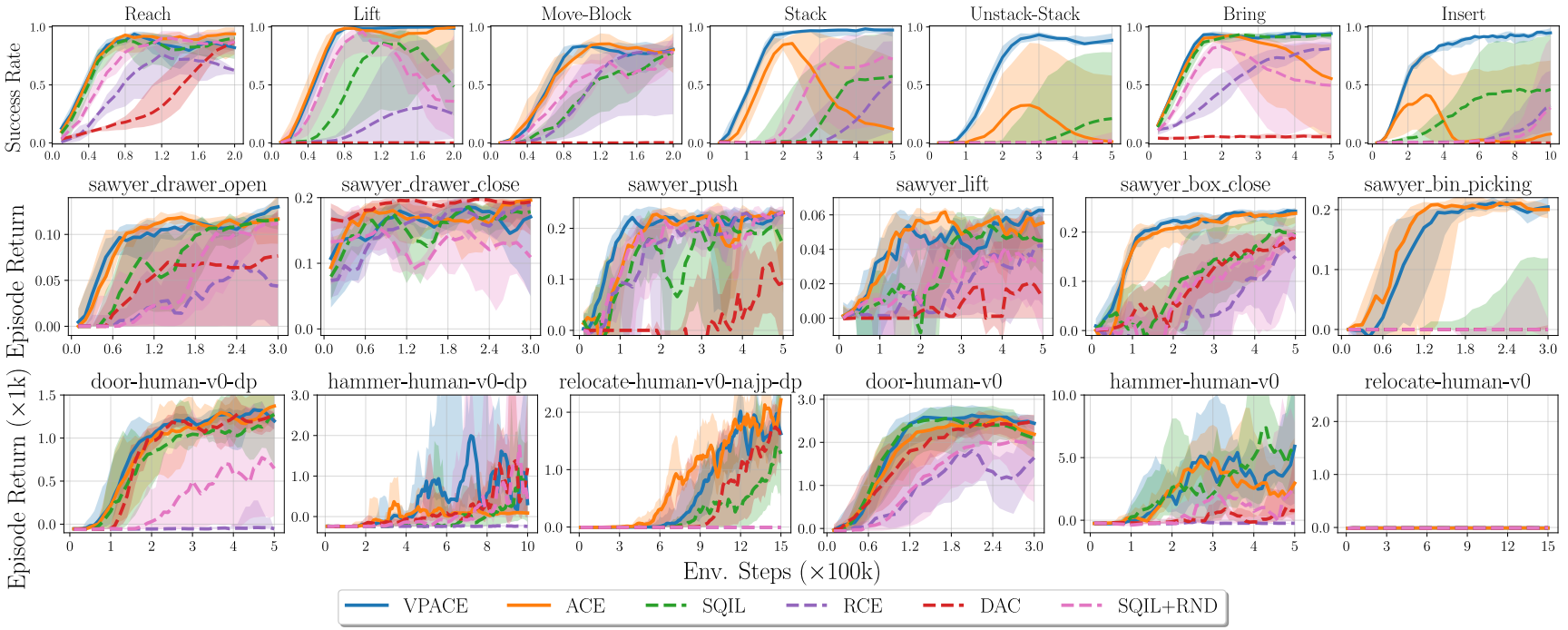


Figure B.5: Sample efficiency performance plots for main task only for all simulated main tasks, including baselines not shown in the main text. Methods introduced in this work have solid lines, while baselines are shown with dashed lines. Performance is an interquartile mean (IQM) across 5 timesteps and 5 seeds with shaded regions showing 95% stratified bootstrap confidence intervals (Agarwal et al., 2021).

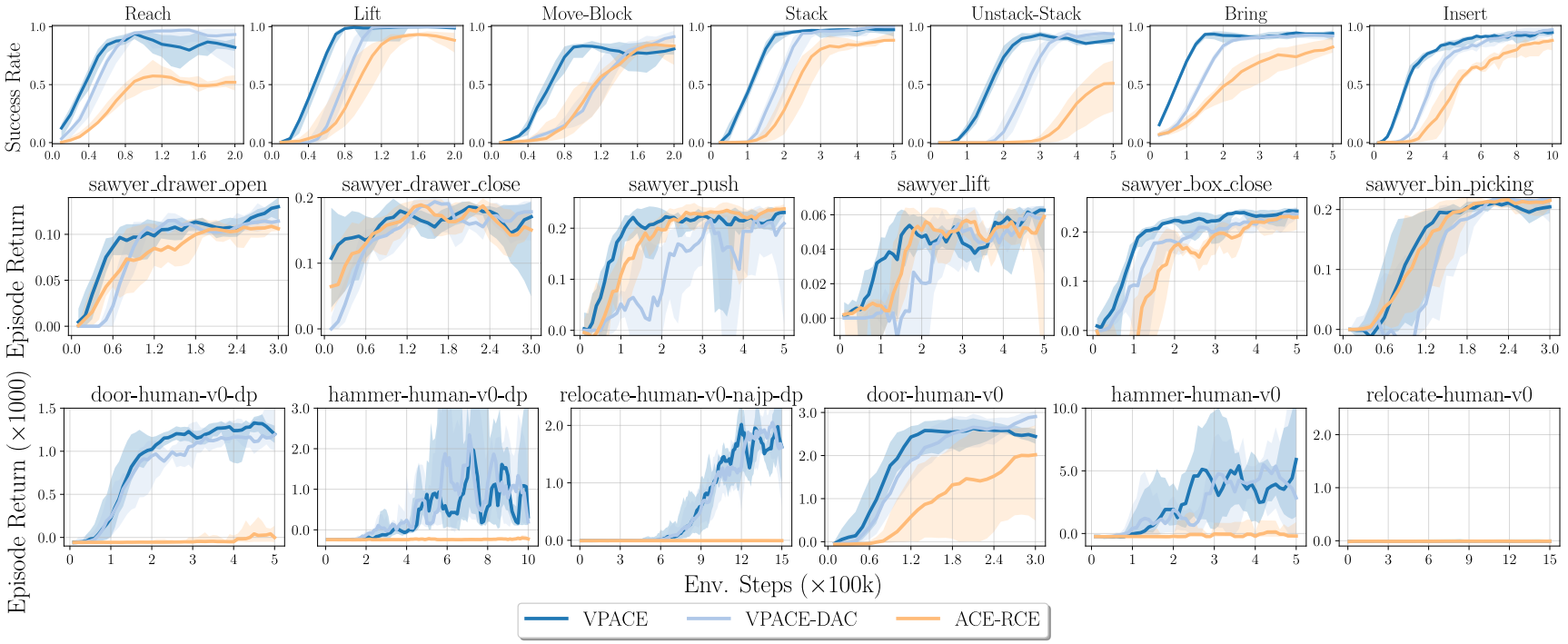


Figure B.6: Comparison of underlying reward models when used with ACE. Our main results (VPACE) used SQIL as the reward model. VPACE-DAC eventually reaches similar performance to VPACE-SQIL, but tends to take longer, and ACE-RCE often has far poorer performance than both.

B.3.3 Single-task Reward Model Comparison

We compare SQIL, DAC, and RCE , without adding value penalization (VP) or auxiliary control from examples (ACE) in Fig. B.4, with two additional, more recent EBC baselines: **EMBER** (Wu et al., 2021) and RCE combined with conservative Q-learning (**RCE-CQL**)(Kumar et al., 2020), (Hatch et al., 2021)). We find that both EMBER and RCE-CQL perform quite poorly, and much more poorly than SQIL, DAC, and RCE, further justifying their use as our primary reward models in our main experiments.

B.3.4 Single-task Value Penalization

Our scheme for value penalization, while initially motivated by the use of ACE, can be used in the single task regime. In Fig. B.8, we compare the performance of VPACE, SQIL, and SQIL with value penalization, but *without* auxiliary tasks (VP-SQIL). VP-SQIL either has no effect on performance or results in an improvement over SQIL, as expected, but is still strongly outperformed by VPACE on the more complicated tasks, such as `Stack`, `Unstack-Stack`, `Insert`, `sawyer_box_close`, `sawyer_bin_picking`, and the `dp` variant of `relocate-human-v0`.

B.3.5 Q-Value Overestimation and Penalization – All Environments

Fig. B.9 shows the same value overestimation analysis plots shown in the original paper for all tasks in. The results for other difficult tasks also show clear violations of $y(s, a) \leq y(s^*, a^*)$, and tasks in which this rule is violated also often have poorer performance. Intriguingly, although the rule is severely violated for many Adroit hand tasks, ACE without VP still has reasonable performance in some cases. This shows that highly uncalibrated Q estimates can still, sometimes, lead to adequate performance. We hypothesize that this occurs because these tasks do not necessarily need $s_T \sim \mathcal{B}$ to match $s^* \sim \mathcal{B}^*_{\text{main}}$ to achieve high return, but we leave investigating this point to future work.

B.4 Why Does SQIL Outperform RCE?

Our results show that VPACE-SQIL outperforms ACE-RCE, and that VP-SQIL and SQIL outperform RCE in almost all cases. This result is in conflict with results from (Eysenbach et al., 2021), which showed SQIL strongly outperformed by RCE. In this section, we show results that help explain our findings.

(Eysenbach et al., 2021) claimed that the highest performing baseline against RCE was SQIL, but it is worth noting that their implementation[1] is a departure from the original SQIL implementation, which uses MSE for TD updates, described in Section B.1.2. Furthermore, (Eysenbach et al., 2021) also noted that it was necessary to add n-step returns to off-policy learning to get reasonable performance. In their experiments, however, this adjustment was not included for their version of SQIL with BCE loss. We complete the experiments using their implementation and show the average results across all

[1] Available at https://github.com/google-research/google-research/tree/master/rce at time of writing.

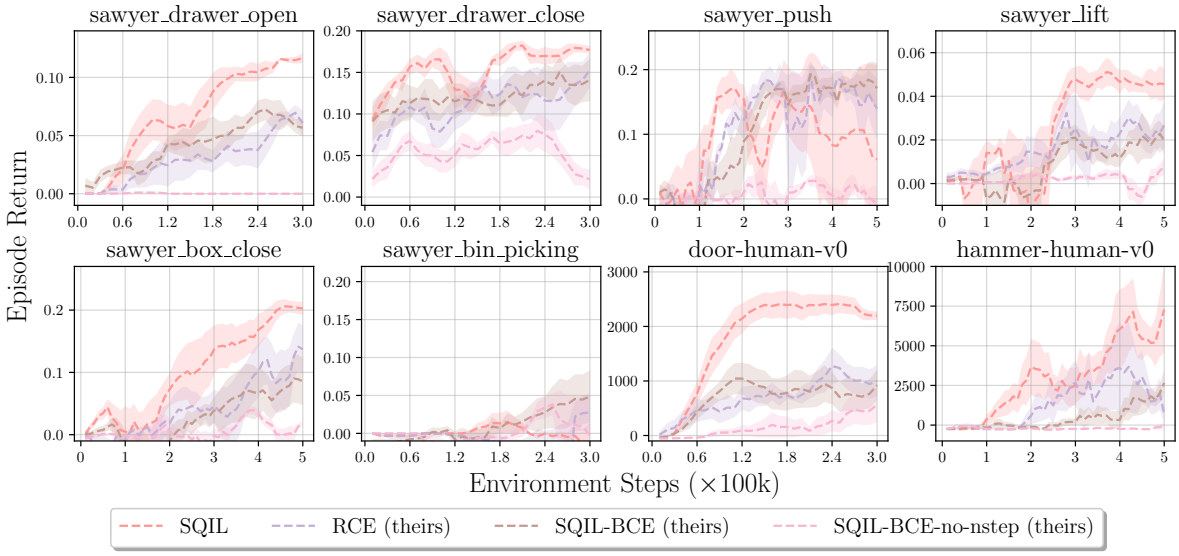


Figure B.7: Performance results on the Sawyer and Adroit tasks considered in (Eysenbach et al., 2021), using their own implementations based on binary cross entropy (BCE) loss, but with an additional SQIL-BCE (including n-step) baseline. We also show our implementation of SQIL (without value penalization or auxiliary tasks), with Mean Squared Error TD updates, which outperforms all of the BCE-based methods on average.

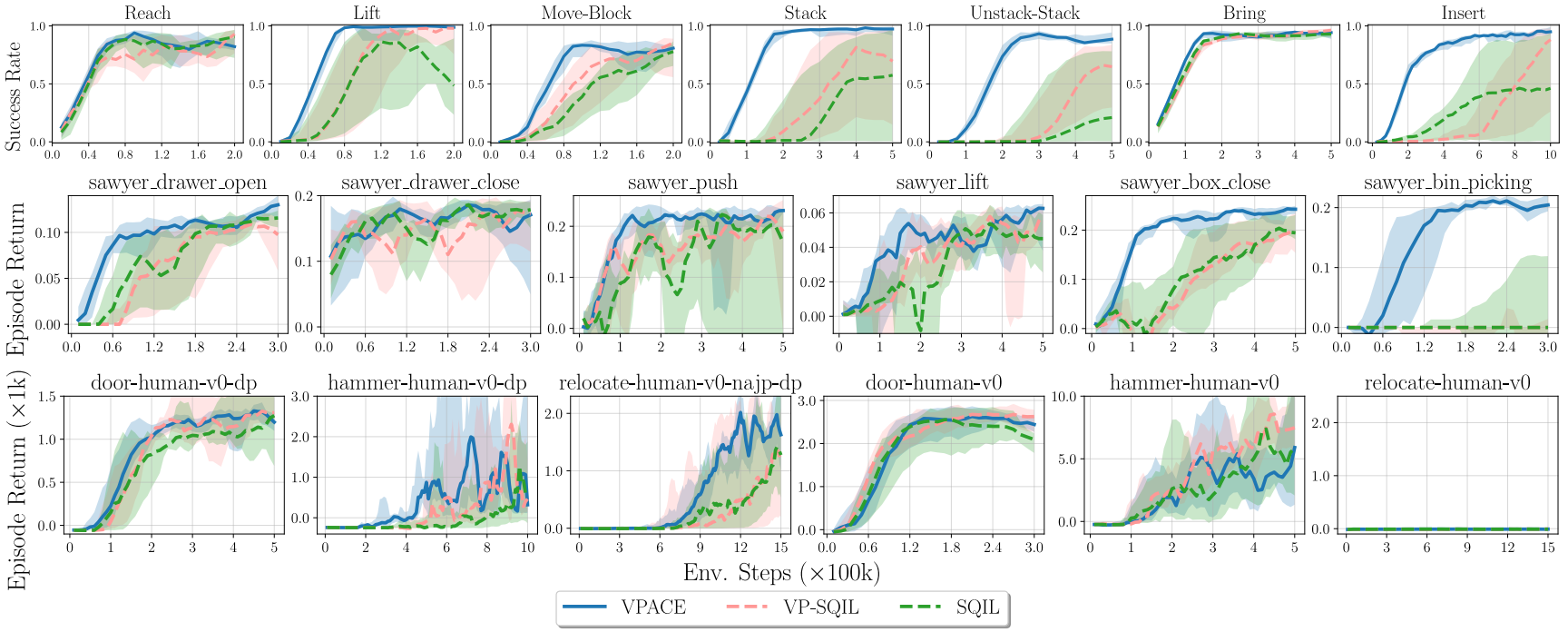


Figure B.8: Comparison of different variants of SQIL: both VP and ACE (VPACE), VP-only (VP-SQIL), or SQIL alone. VP-SQIL generally either results in an improvement or has no effect on performance, compared with SQIL, but the exclusion of ACE results in far poorer performance than VPACE, especially for the most complex tasks.

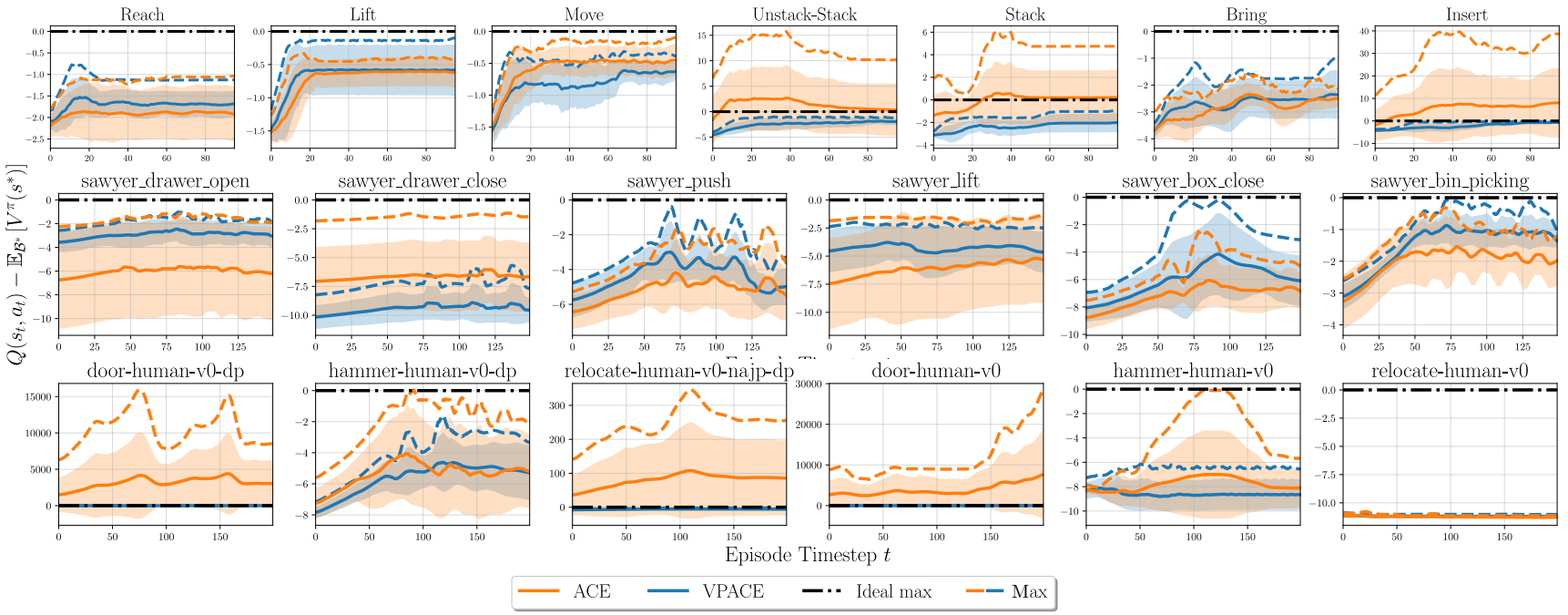


Figure B.9: Comparison of different variants of SQIL: both VP and ACE (VPACE), VP-only (VP-SQIL), or SQIL alone. VP-SQIL generally either results in an improvement or has no effect on performance, compared with SQIL, but the exclusion of ACE results in far poorer performance than VPACE, especially for the most complex tasks.

RCE environments in Fig. B.7, and also compare to using SQIL with MSE (without value penalization or auxiliary tasks) for TD updates.

These results empirically show that RCE and SQIL with BCE loss perform nearly identically, indicating that benefits of the changed TD targets and weights described in Section B.1.3 may not be as clear as previously described. SQIL with MSE clearly performs better on average, although it still performs worse than VPACE (see Section B.3).

B.5 Expanded Limitations

In this section, we expand on some of the limitations originally discussed in Section 7.5, where we previously skipped limitations that are inherent to reinforcement learning and to learning from guided play (LfGP, (Ablett et al., 2023)), both of which are core parts of our approach.

Experimental limitation—Numerical state data only All of our tests are done with numerical data, instead of image-based data. Other work (Riedmiller et al., 2018), (Yarats et al., 2022) has shown that for some environments, image-based learning just results in slowing learning compared with numerical state data, and we assume that the same would be true for our method as well.

Assumption—Generating example success states is easier We claim that success example distributions are easier to generate than full trajectory expert data, and while we expect this to be true in almost all cases, there may still be tasks or environments where accomplishing this is not trivial. As well, similar to other imitation learning methods, knowing how much data is required to generate an effective policy is unknown, but adding a way to append to the existing success state distribution (e.g., (Singh et al., 2019)) would presumably help mitigate this.

Assumption/failure mode—Unimodal example distributions Although we do not explicitly claim that unimodal example state distributions are required for VPACE to work, all of our tested tasks have roughly unimodal example state distributions. It is not clear whether our method would gracefully extend to the multimodal case, and investigating this is an interesting direction for future work.

Experimental limitation—Some environment-specific hyperparameters While the vast majority of hyperparameters were transferable between all environments and algorithms, the scheduler period, the inclusion of n-step targets, and the use of entropy in TD updates, were different between environments to maximize performance. Scheduler periods will be different for all environments, but future work should further investigate why n-step targets and the inclusion of entropy in TD updates makes environment-specific differences.

B.5.1 VPACE and LfGP Limitations

VPACE shares the following limitations with LfGP (Ablett et al., 2023).

Assumption—Existence of auxiliary task datasets VPACE and LfGP require the existence of auxiliary task example datasets $\mathcal{B}^*_{\text{aux}}$, in addition to a main task dataset $\mathcal{B}^*_{\text{main}}$. This places higher initial

burden on the practitioner. In future, choosing environments where this data can be reused as much as possible will reduce this burden.

Assumption—Clear auxiliary task definitions VPACE and LfGP require a practitioner to manually define auxiliary tasks. We expect this to be comparatively easier than generating a similar dense reward function, since it does not require evaluating the relative contribution of individual auxiliary tasks. As well, all tasks studied in this work share task definitions, and the Panda environment even shares task data itself, leading us to assume that these task definitions will extend to other manipulation tasks as well.

Assumption—Clear choices for handcrafted scheduler trajectories VPACE and LfGP use a combination of a weighted random scheduler with a handcrafted scheduler, randomly sampling from predefined trajectories of high level tasks. (Ablett et al., 2023) found that the handcrafted scheduler added little benefit compared with a weighted random scheduler, and further work should investigate this claim, or perhaps attempt to use a learned scheduler, as in (Riedmiller et al., 2018).

B.5.2 Reinforcement Learning Limitations

Experimental limitation—Free environment exploration As is common in reinforcement learning methods, our method requires exploration of environments for a considerable amount of time (on the order of hours), which may be unacceptable for tasks with, e.g., delicate objects.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/.

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML'04)*, Banff, Alberta, Canada. ACM Press.

Ablett, T., Chan, B., and Kelly, J. (2021a). Learning from Guided Play: A Scheduled Hierarchical Approach for Improving Exploration in Adversarial Imitation Learning. In *Proceedings of the Neural Information Processing Systems (NeurIPS'21) Deep Reinforcement Learning Workshop*, Online.

Ablett, T., Chan, B., and Kelly, J. (2023). Learning From Guided Play: Improving Exploration for Adversarial Imitation Learning With Simple Auxiliary Tasks. *IEEE Robotics and Automation Letters*, 8(3):1263–1270.

Ablett, T., Chan, B., Wang, J. H., and Kelly, J. (2024a). Fast Reinforcement Learning without Rewards or Demonstrations via Auxiliary Task Examples. In *CoRL 2024 Workshop on Mastering Robot Manipulation in a World of Abundant Data*, Munich, Germany.

Ablett, T., Chan, B., Wang, J. H., and Kelly, J. (2025). Efficient Imitation Without Demonstrations via Value-Penalized Auxiliary Control from Examples. In *IEEE International Conference on Robotics and Automation (ICRA'25)*, Atlanta, GA, USA.

Ablett, T., Limoyo, O., Sigal, A., Jilani, A., Kelly, J., Siddiqi, K., Hogan, F., and Dudek, G. (2024b). Multimodal and Force-Matched Imitation Learning With a See-Through Visuotactile Sensor. *IEEE Transactions on Robotics*, 41:946–959.

Ablett, T., Marić, F., and Kelly, J. (2020). Fighting Failures with FIRE: Failure Identification to Reduce Expert Burden in Intervention-Based Learning. Technical Report STARS-2020-001, University of Toronto.

Ablett, T., Zhai, Y., and Kelly, J. (2021b). Seeing All the Angles: Learning Multiview Manipulation Policies for Contact-Rich Tasks from Demonstrations. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'21)*, pages 7843–7850, Prague, Czech Republic.

Abu-Dakka, F. J., Rozo, L., and Caldwell, D. G. (2018). Force-based variable impedance learning for robotic manipulation. *Robotics and Autonomous Systems*, 109:156–167.

Achiam, J. (2018). Spinning Up in Deep Reinforcement Learning. https://spinningup.openai.com/en/latest/index.html.

Adamczyk, J., Makarenko, V., Tiomkin, S., and Kulkarni, R. V. (2024). Boosting Soft Q-Learning by Bounding. *Reinforcement Learning Journal*, 1(1).

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. (2021). Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Advances in Neural Information Processing Systems (Neurips'21)*, volume 34.

Akgun, B. and Subramanian, K. (2011). Robot Learning from Demonstration: Kinesthetic Teaching vs. Teleoperation. *Technical Report*.

Amini, A., Gilitschenski, I., Phillips, J., Moseyko, J., Banerjee, R., Karaman, S., and Rus, D. (2020). Learning Robust Control Policies for End-to-End Autonomous Driving From Data-Driven Simulation. *IEEE Robotics and Automation Letters*, 5(2):1143–1150.

Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight Experience Replay. In *Advances in Neural Information Processing Systems (NIPS'17)*, Long Beach, CA, USA.

Ankile, L., Simeonov, A., Shenfeld, I., Torne, M., and Agrawal, P. (2024). From Imitation to Refinement – Residual RL for Precise Assembly.

Bagnell, J. A. (2005). Robust Supervised Learning. In *AAAI Conference on Artificial Intelligence (AAAI'05)*.

Bain, M. and Sammut, C. (1996). A Framework for Behavioural Cloning. In *Machine Intelligence 15*, pages 103–129. Oxford University Press.

Bajracharya, M., Borders, J., Helmick, D., Kollar, T., Laskey, M., Leichty, J., Ma, J., Nagarajan, U., Ochiai, A., Petersen, J., Shankar, K., Stone, K., and Takaoka, Y. (2020). A Mobile Manipulation System for One-Shot Teaching of Complex Tasks in Homes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'20)*, pages 11039–11045.

Balloch, J. C., Bhagat, R., Zollicoffer, G., Jia, R., Kim, J., and Riedl, M. O. (2024). Is Exploration All You Need? Effective Exploration Characteristics for Transfer in Reinforcement Learning. arXiv:2404.02235.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying Count-Based Exploration and Intrinsic Motivation. In *Conference on Neural Information Processing Systems*, volume 29.

Bellman, R. (1957). *Dynamic Programming*. Dover Books on Computer Science Series. Princeton University Press.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8, Montreal, Quebec, Canada. ACM Press.

Billard, A. G., Calinon, S., and Dillmann, R. (2016). Learning from Humans. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 1995–2014. Springer International Publishing, Cham.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. (2016). End to End Learning for Self-Driving Cars. *arXiv:1604.07316 [cs]*.

Bradski, G. (2000). The OpenCV library. *Dr. Dobb's Journal of Software Tools*.

Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24(2):123–140.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2019). Exploration by Random Network Distillation. In *International Conference on Learning Representations (ICLR'19)*, New Orleans, LA, USA. arXiv.

Cabi, S., Gómez, S., Matthew, C., Hoffman, W., Denil, M., Wang, Z., De, N., and Deepmind, F. (2017). The Intentional Unintentional Agent: Learning to Solve Many Continuous Control Tasks Simultaneously. In *Conference on Robot Learning (CoRL'17)*, Mountain View, USA.

Cabi, S., Gómez Colmenarejo, S., Novikov, A., Konyushova, K., Reed, S., Jeong, R., Zolna, K., Aytar, Y., Budden, D., Vecerik, M., Sushkov, O., Barker, D., Scholz, J., Denil, M., de Freitas, N., and Wang, Z. (2020). Scaling Data-Driven Robotics with Reward Sketching and Batch Reinforcement Learning. In *Proceedings of Robotics: Science and Systems (RSS'20)*. Robotics: Science and Systems Foundation.

Celemin, C., Pérez-Dattari, R., Chisari, E., Franzese, G., Rosa, L. d. S., Prakash, R., Ajanović, Z., Ferraz, M., Valada, A., and Kober, J. (2022). Interactive Imitation Learning in Robotics: A Survey. *Foundations and Trends® in Robotics*, 10(1-2):1–197.

Chan, B. (2020). RL sandbox. https://github.com/chanb/rl_sandbox_public.

Chang, W.-D., Fujimoto, S., Meger, D., and Dudek, G. (2024). Imitation learning from observation through optimal transport. *Reinforcement Learning Journal*, 4:1911–1923.

Chebotar, Y., Kroemer, O., and Peters, J. (2014). Learning robot tactile sensing for object manipulation. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3368–3375.

Chen, X., Wang, C., Zhou, Z., and Ross, K. (2021). Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. *arXiv:2101.05982 [cs]*.

Chi, C., Sun, X., Xue, N., Li, T., and Liu, C. (2018). Recent Progress in Technologies for Tactile Sensors. *Sensors*, 18(4):948.

Codevilla, F., Müller, M., López, A., Koltun, V., and Dosovitskiy, A. (2018). End-to-End Driving Via Conditional Imitation Learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'18)*, pages 4693–4700, Brisbane, Australia.

Coumans, E. and Bai, Y. (2016–2019). PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Cui, Y., Isele, D., Niekum, S., and Fujimura, K. (2019). Uncertainty-aware data aggregation for deep imitation learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'19)*, pages 761–767, Montreal, QC, Canada.

Dasari, S., Wang, J., Hong, J., Bahl, S., Lin, Y., Wang, A. S., Thankaraj, A., Chahal, K. S., Calli, B., Gupta, S., Held, D., Pinto, L., Pathak, D., Kumar, V., and Gupta, A. (2021). RB2: Robotic Manipulation Benchmarking with a Twist. In *Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

de Haan, P., Jayaraman, D., and Levine, S. (2019). Causal Confusion in Imitation Learning. In *Advances in Neural Information Processing Systems (Neurips'19)*, pages 11693–11704.

Dwibedi, D., Tompson, J., Lynch, C., and Sermanet, P. (2018). Learning Actionable Representations from Visual Observations. In *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'18)*, pages 1577–1584, Madrid, Spain.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2021). First return, then explore. *Nature*, 590(7847):580–586.

Eslami, S. M. A., Jimenez Rezende, D., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., Reichert, D. P., Buesing, L., Weber, T., Vinyals, O., Rosenbaum, D., Rabinowitz, N., King, H., Hillier, C., Botvinick, M., Wierstra, D., Kavukcuoglu, K., and Hassabis, D. (2018). Neural scene representation and rendering. *Science*, 360(6394):1204–1210.

Eysenbach, B., Levine, S., and Salakhutdinov, R. (2021). Replacing Rewards with Examples: Example-Based Policy Search via Recursive Classification. In *Advances in Neural Information Processing Systems (NeurIPS'21)*, Virtual.

Figueroa, N. (2023). Easy-kinesthetic-recording. https://github.com/nbfigueroa/easy-kinesthetic-recording.

Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *Proceedings of the 33rd International Conference on Machine Learning (ICML'16)*, ICML'16, pages 49–58, New York, NY, USA. JMLR.org.

Fischer, K., Kirstein, F., Jensen, L. C., Krüger, N., Kukliński, K., aus der Wieschen, M. V., and Savarimuthu, T. R. (2016). A comparison of types of robot control for programming by Demonstration. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 213–220.

Florence, P., Manuelli, L., and Tedrake, R. (2020). Self-Supervised Correspondence in Visuomotor Policy Learning. *IEEE Robotics and Automation Letters*, 5(2):492–499.

Florence, P. R., Manuelli, L., and Tedrake, R. (2018). Dense Object Nets: Learning Dense Visual Object Descriptors By and For Robotic Manipulation. In *The 2nd Annual Conference on Robot Learning (CoRL'18)*, pages 373–385, Zurich, Switzerland.

Fu, J., Luo, K., and Levine, S. (2018a). Learning Robust Rewards with Adverserial inverse Reinforcement Learning. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*, Vancouver, BC, Canada.

Fu, J., Singh, A., Ghosh, D., Yang, L., and Levine, S. (2018b). Variational Inverse Control with Events: A General Framework for Data-Driven Reward Definition. In *Advances in Neural Information Processing Systems (NeurIPS'18)*, Montreal, Canada.

Fujimoto, S., Meger, D., and Precup, D. (2019). Off-Policy Deep Reinforcement Learning without Exploration. In *International Conference on Machine Learning (ICML'19)*, Long Beach, CA, USA.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, pages 1582–1591, Stockholm, Sweden.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., and Marín-Jiménez, M. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292.

Ghasemipour, S. K. S., Zemel, R. S., and Gu, S. S. (2019). A Divergence Minimization Perspective on Imitation Learning Methods. In *Conference on Robot Learning (CoRL'19)*.

Giusti, A., Guzzi, J., Ciresan, D. C., He, F.-L., Rodriguez, J. P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Caro, G. D., Scaramuzza, D., and Gambardella, L. M. (2016). A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots. *IEEE Robotics and Automation Letters*, 1(2):661–667.

Goecks, V. G., Gremillion, G. M., Lawhern, V. J., Valasek, J., and Waytowich, N. R. (2019). Efficiently Combining Human Demonstrations and Interventions for Safe Training of Autonomous Systems

in Real-Time. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, pages 2462–2470.

Goecks, V. G., Gremillion, G. M., Lawhern, V. J., Valasek, J., and Waytowich, N. R. (2020). Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMS'20)*, AAMAS '20, pages 465–473, Richland, SC, USA. International Foundation for Autonomous Agents and Multiagent Systems.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT press.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14)*, NIPS'14, pages 2672–2680, Montreal, QC, Canada. MIT Press.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved Training of Wasserstein GANs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Conference on Neural Information Processing Systems*, pages 5767–5777, Long Beach, CA, USA. Curran Associates, Inc.

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. (2019). Relay Policy Learning: Solving Long Horizon Tasks Via Imitation and Reinforcement Learning. In *Conference on Robot Learning (CoRL'19)*.

Gupta, A., Pacchiano, A., Zhai, Y., Kakade, S. M., and Levine, S. (2022). Unpacking Reward Shaping: Understanding the Benefits of Reward Engineering on Sample Complexity. In *Advances in Neural Information Processing Systems (Neurips'22)*, New Orleans, LA, USA.

Gupta, A., Yu, J., Zhao, T. Z., Kumar, V., Rovinsky, A., Xu, K., Devlin, T., and Levine, S. (2021). Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors without Human Intervention. In *Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA'21)*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, pages 1861–1870, Stockholm, Sweden.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2019). Soft Actor-Critic Algorithms and Applications. *arXiv:1812.05905 [cs, stat]*.

Hansen, J., Hogan, F., Rivkin, D., Meger, D., Jenkin, M., and Dudek, G. (2022). Visuotactile-RL: Learning Multimodal Manipulation Policies with Deep Reinforcement Learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8298–8304.

Hardt, M. and Recht, B. (2022). *Patterns, Predictions, and Actions: Foundations of Machine Learning*. Princeton University Press.

Hatch, K., Yu, T., Rafailov, R., and Finn, C. (2021). Example-Based Offline Reinforcement Learning without Rewards. In *Advances in Neural Information Processing Systems (NeurIPS'21) Offline Reinforcement Learning Workshop*.

Hatch, K. B., Eysenbach, B., Rafailov, R., Yu, T., Salakhutdinov, R., Levine, S., and Finn, C. (2023). Contrastive Example-Based Control. In Matni, N., Morari, M., and Pappas, G. J., editors, *Learning for Dynamics and Control (L4DC'23)*, volume 211 of *Proceedings of Machine Learning Research*, pages 155–169, Philadelphia, PA, USA. PMLR.

Hausman, K., Chebotar, Y., Schaal, S., Sukhatme, G., and Lim, J. (2017). Multi-Modal Imitation Learning from Unstructured Demonstrations using Generative Adversarial Nets. In *Conference on Neural Information Processing Systems*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, Las Vegas, NV, USA. IEEE.

Henderson, P., Chang, W.-D., Bacon, P.-L., Meger, D., Pineau, J., and Precup, D. (2018). OptionGAN: Learning Joint Reward-Policy Options Using Generative Adversarial Inverse Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'18)*.

Ho, J. and Ermon, S. (2016). Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems (NIPS'16)*, Barcelona, Spain.

Hogan, F. R., Jenkin, M., Rezaei-Shoshtari, S., Girdhar, Y., Meger, D., and Dudek, G. (2021). Seeing Through your Skin: Recognizing Objects with a Novel Visuotactile Sensor. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1217–1226.

Hogan, F. R., Tremblay, J.-F., Baghi, B. H., Jenkin, M., Siddiqi, K., and Dudek, G. (2022). Finger-STS: Combined Proximity and Tactile Sensing for Robotic Manipulation. *IEEE Robotics and Automation Letters*, 7(4):10865–10872.

Hogan, N. (1984). Impedance Control: An Approach to Manipulation. In *1984 American Control Conference*, pages 304–313.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Huang, I. and Bajcsy, R. (2020). Robot Learning from Demonstration with Tactile Signals for Geometry-Dependent Tasks. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8323–8328, Las Vegas, NV, USA. IEEE.

Hussenot, L., Andrychowicz, M., Vincent, D., Dadashi, R., Raichuk, A., Ramos, S., Momchev, N., Girgin, S., Marinier, R., Stafiniak, L., Orsini, M., Bachem, O., Geist, M., and Pietquin, O. (2021). Hyperparameter Selection for Imitation Learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, pages 4511–4522.

Iriondo, A., Lazkano, E., Susperregi, L., Urain, J., Fernandez, A., and Molina, J. (2019). Pick and Place Operations in Logistics Using a Mobile Manipulator Controlled with Deep Reinforcement Learning. *Applied Sciences*, 9(2):348.

James, S., Wada, K., Laidlow, T., and Davison, A. J. (2022). Coarse-to-Fine Q-Attention: Efficient Learning for Visual Robotic Manipulation Via Discretisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13739–13748.

Jena, R., Liu, C., and Sycara, K. (2020). Augmenting GAIL with BC for sample efficient imitation learning. In *Proceedings of Robotics: Science and Systems (RSS'20) Workshop on Advances & Challenges in Imitation Learning for Robotics*.

Jilani, A. (2024). Tactile recovery of shape from texture deformation. Master's thesis, McGill University.

Jing, M., Huang, W., Sun, F., Ma, X., Kong, T., Gan, C., and Li, L. (2021). Adversarial Option-Aware Hierarchical Imitation Learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, pages 5097–5106.

Johns, E. (2021). Coarse-to-Fine Imitation Learning: Robot Manipulation from a Single Demonstration. *arXiv:2105.06411 [cs]*.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., and Levine, S. (2018). QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. In *Conference on Robot Learning (CoRL'18)*, Zurich, Switzerland.

Kelly, M., Sidrane, C., Driggs-Campbell, K., and Kochenderfer, M. J. (2019). HG-DAgger: Interactive Imitation Learning with Human Experts. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'19)*, pages 8077–8083, Montreal, QC, Canada.

Kim, B. and Pineau, J. (2013). Maximum Mean Discrepancy Imitation Learning. In *Proceedings of Robotics: Science and Systems (RSS'13)*, Berlin, Germany. Robotics: Science and Systems Foundation.

Kim, W., Kim, W. D., Kim, J.-J., Kim, C.-H., and Kim, J. (2022). UVtac: Switchable UV Marker-Based Tactile Sensing Finger for Effective Force Estimation and Object Localization. *IEEE Robotics and Automation Letters*, 7(3):6036–6043.

Kindle, J., Furrer, F., Novkovic, T., Chung, J. J., Siegwart, R., and Nieto, J. (2020). Whole-Body Control of a Mobile Manipulator using End-to-End Reinforcement Learning. *arXiv:2003.02637 [cs]*.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*, San Diego, CA, USA.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational Dropout and the Local Reparameterization Trick. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*.

Kormushev, P., Calinon, S., and Caldwell, D. G. (2011). Imitation Learning of Positional and Force Skills Demonstrated via Kinesthetic Teaching and Haptic Input. *Advanced Robotics*, 25(5):581–603.

Kostrikov, I. (2018). PyTorch Implementations of Reinforcement Learning Algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail.

Kostrikov, I., Agrawal, K. K., Dwibedi, D., Levine, S., and Tompson, J. (2019). Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning. In *Proceedings of the International Conference on Learning Representations (ICLR'19)*, New Orleans, LA, USA.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative Q-Learning for Offline Reinforcement Learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.-F., and Lin, H.-T., editors, *Advances in Neural Information Processing Systems (Neurips'20)*.

Laskey, M., Lee, J., Fox, R., Dragan, A. D., and Goldberg, K. (2017a). DART: Noise Injection for Robust Imitation Learning. In *Proceedings of the 1st Annual Conference on Robot Learning (CoRL'17)*, pages 143–156, Mountain View, CA, USA.

Laskey, M., Powers, C., Joshi, R., Poursohi, A., and Goldberg, K. (2017b). Learning Robust Bed Making using Deep Imitation Learning with DART. *arXiv:1711.02525 [cs]*.

Laskey, M., Staszak, S., Hsieh, W. Y., Mahler, J., Pokorny, F. T., Dragan, A. D., and Goldberg, K. (2016). SHIV: Reducing supervisor burden in DAgger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'16)*, pages 462–469, Stockholm, Sweden.

Lee, A. X., Lu, H., Gupta, A., Levine, S., and Abbeel, P. (2015). Learning force-based manipulation of deformable objects from multiple demonstrations. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 177–184.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv:2005.01643 [cs, stat]*.

Li, H., Zhang, Y., Zhu, J., Wang, S., Lee, M. A., Xu, H., Adelson, E., Fei-Fei, L., Gao, R., and Wu, J. (2023a). See, hear, and feel: Smart sensory fusion for robotic manipulation. In Liu, K., Kulic, D., and

Ichnowski, J., editors, *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1368–1378. PMLR.

Li, K., Chappell, D., and Rojas, N. (2023b). Immersive Demonstrations are the Key to Imitation Learning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5071–5077.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y., editors, *International Conference on Learning Representations (ICLR'16)*, San Juan, Puerto Rico.

Limoyo, O., Ablett, T., and Kelly, J. (2023). Learning Sequential Latent Variable Models from Multimodal Time Series Data. In Petrovic, I., Menegatti, E., and Marković, I., editors, *Intelligent Autonomous Systems 17*, Lecture Notes in Networks and Systems, pages 511–528, Cham. Springer Nature Switzerland.

Limoyo, O., Ablett, T., Marić, F., Volpatti, L., and Kelly, J. (2018). Self-Calibration of Mobile Manipulator Kinematic and Sensor Extrinsic Parameters Through Contact-Based Interaction. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8.

Limoyo, O., Chan, B., Marić, F., Wagstaff, B., Mahmood, A. R., and Kelly, J. (2020). Heteroscedastic Uncertainty for Robust Generative Latent Dynamics. *IEEE Robotics and Automation Letters*, 5(4):6654–6661.

Lin, Y., Wang, A. S., Sutanto, G., Rai, A., and Meier, F. (2021). Polymetis. https://facebookresearch.github.io/fairo/polymetis/.

Lu, Y., Hausman, K., Chebotar, Y., Yan, M., Jang, E., Herzog, A., Xiao, T., Irpan, A., Khansari, M., Kalashnikov, D., and Levine, S. (2021). AW-Opt: Learning Robotic Skills with Imitation and Reinforcement at Scale. *arXiv:2111.05424 [cs]*.

Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. (2019). Learning Latent Plans from Play. In *Conference on Robot Learning (CoRL'19)*.

Lynch, C. and Sermanet, P. (2021). Language Conditioned Imitation Learning over Unstructured Data.

Ma, D., Donlon, E., Dong, S., and Rodriguez, A. (2019). Dense Tactile Force Estimation using GelSlim and inverse FEM. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5418–5424.

Maeda, G., Väätäinen, J., and Yoshida, H. (2020). Visual Task Progress Estimation with Appearance Invariant Embeddings for Robot Control and Planning. In *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'20)*, pages 7941–7948, Las Vegas, NV, USA.

Mandlekar, A., Nasiriany, S., Wen, B., Akinola, I., Narang, Y., Fan, L., Zhu, Y., and Fox, D. (2023). MimicGen: A data generation system for scalable robot learning using human demonstrations. In *7th Annual Conference on Robot Learning*.

Mandlekar, A., Xu, D., Wong, J., Nasiriany, S., Wang, C., Kulkarni, R., Fei-Fei, L., Savarese, S., Zhu, Y., and Martín-Martín, R. (2022). What matters in learning from offline human demonstrations for robot manipulation. In Faust, A., Hsu, D., and Neumann, G., editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1678–1690. PMLR.

Menda, K., Driggs-Campbell, K., and Kochenderfer, M. J. (2019). EnsembleDAgger: A bayesian approach to safe imitation learning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'19)*, pages 5041–5048, Macau, China.

Minsky, M. (1961). Steps toward Artificial Intelligence. *Proceedings of the IRE*, 49(1):8–30.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Murphy, K. P. (2022). *Probabilistic Machine Learning: An Introduction*. MIT Press.

Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., and Levine, S. (2019). Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning? In *Proceedings of the Neural Information Processing Systems (NeurIPS'19) Deep Reinforcement Learning Workshop*.

Nair, A., Gupta, A., Dalal, M., and Levine, S. (2021). AWAC: Accelerating Online Reinforcement Learning with Offline Datasets. *arXiv:2006.09359 [cs, stat]*.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming Exploration in Reinforcement Learning with Demonstrations. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA'18)*, pages 6292–6299, Brisbane, Australia.

Ng, A. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML'00)*, pages 663–670.

Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99)*, ICML '99, pages 278–287, San Francisco, CA, USA.

Ng, A. Y. and Jordan, M. I. (2003). *Shaping and Policy Search in Reinforcement Learning*. PhD thesis, University of California, Berkeley.

Orsini, M., Raichuk, A., Hussenot, L., Vincent, D., Dadashi, R., Girgin, S., Geist, M., Bachem, O., Pietquin, O., and Andrychowicz, M. (2021). What Matters for Adversarial Imitation Learning? In *Conference on Neural Information Processing Systems*.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., and Peters, J. (2018). An Algorithmic Perspective on Imitation Learning. *Foundations and Trends in Robotics*, 7(1-2):1–179.

Padmanabha, A., Ebert, F., Tian, S., Calandra, R., Finn, C., and Levine, S. (2020). OmniTact: A Multi-Directional High-Resolution Touch Sensor. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 618–624.

Pan, Y., Cheng, C.-A., Saigol, K., Lee, K., Yan, X., Theodorou, E. A., and Boots, B. (2018). Agile Autonomous Driving using End-to-End Deep Imitation Learning. In Kress-Gazit, H., Srinivasa, S. S., Howard, T., and Atanasov, N., editors, *Proceedings of Robotics: Science and Systems (RSS'18)*, Pittsburgh, PA, USA.

Pardo, F., Tavakoli, A., Levdik, V., and Kormushev, P. (2018). Time Limits in Reinforcement Learning. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4042–4051. PMLR.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pateria, S., Subagdja, B., Tan, A.-h., and Quek, C. (2021). Hierarchical Reinforcement Learning: A Comprehensive Survey. *ACM Computing Surveys*, 54(5):109:1–109:35.

Pervez, A., Ali, A., Ryu, J.-H., and Lee, D. (2017). Novel learning from demonstration approach for repetitive teleoperation tasks. In *2017 IEEE World Haptics Conference (WHC)*, pages 60–65, Munich, Germany. IEEE.

Peters, J. and Kober, J. (2009). Using reward-weighted imitation for robot Reinforcement Learning. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 226–232, Nashville, TN, USA. IEEE.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. (2018). Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. *arXiv:1802.09464 [cs]*.

Pomerleau, D. A. (1989). ALVINN: An Autonomous Land Vehicle in a Neural Network. In Touretzky, D. S., editor, *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'89)*, pages 305–313, Denver, CO, USA. Morgan-Kaufmann.

Popov, I., Heess, N., Lillicrap, T., Hafner, R., Barth-Maron, G., Vecerik, M., Lampe, T., Tassa, Y., Erez, T., and Riedmiller, M. (2017). Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. arXiv:1704.03073 [cs].

Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2009). *Dataset Shift in Machine Learning*. The MIT Press.

Raibert, M. H. and Craig, J. J. (1981). Hybrid Position/Force Control of Manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 103(2):126–133.

Rajeswaran, A., Kumar, A., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2018). Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS'18)*, Pittsburgh, PA, USA.

Reddy, S., Dragan, A. D., and Levine, S. (2020). SQIL: Imitation Learning Via Reinforcement Learning with Sparse Rewards. In *International Conference on Learning Representations (ICLR'20)*.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by Playing Solving Sparse Reward Tasks from Scratch. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, pages 4344–4353, Stockholm, Sweden.

Robert, A., Pike-Burke, C., and Faisal, A. A. (2023). Sample Complexity of Goal-Conditioned Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS'23)*, New Orleans, LA, USA.

Ross, S. (2013). *Interactive Learning for Sequential Decisions and Predictions*. PhD thesis, Carnegie Mellon University.

Ross, S. and Bagnell, D. (2010). Efficient Reductions for Imitation Learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10)*, pages 661–668. JMLR Workshop and Conference Proceedings.

Ross, S., Gordon, G. J., and Bagnell, D. (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS'11)*, pages 627–635, Fort Lauderdale, FL, USA.

Ross, S., Melik-Barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A., and Hebert, M. (2013). Learning monocular reactive UAV control in cluttered natural environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'13)*, pages 1765–1772, Karlsruhe, Germany.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Rummery, G. and Niranjan, M. (1994). On-Line Q-Learning Using Connectionist Systems. *Technical Report CUED/F-INFENG/TR 166*.

Sadeghi, F., Toshev, A., Jang, E., and Levine, S. (2018). Sim2Real Viewpoint Invariant Visual Servoing by Recurrent Control. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'18)*, pages 4691–4699, Salt Lake City, UT, USA. IEEE.

Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., Levine, S., and Brain, G. (2018). Time-Contrastive Networks: Self-Supervised Learning from Video. In *IEEE International Conference on Robotics and Automation (ICRA'18)*, pages 1134–1141, Brisbane, QLD, Australia.

Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(1):379–423.

Sharma, M., Sharma, A., Rhinehart, N., and Kitani, K. M. (2019). Directed-Info GAIL: Learning Hierarchical Policies from Unsegmented Demonstrations using Directed Information. In *International Conference on Learning Representations (ICLR'19)*.

Siciliano, B. (2009). Force Control. In Sciavicco, L., Villani, L., and Oriolo, G., editors, *Robotics: Modelling, Planning and Control*, Advanced Textbooks in Control and Signal Processing, pages 363–405. Springer, London.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Singh, A., Yang, L., Hartikainen, K., Finn, C., and Levine, S. (2019). End-to-End Robotic Reinforcement Learning without Reward Engineering. In *Robotics: Science and Systems (RSS'19)*, Freiburg im Breisgau, Germany.

Sinha, S., Mandlekar, A., and Garg, A. (2021). S4RL: Surprisingly Simple Self-Supervision for Offline Reinforcement Learning. In *Conference on Robot Learning (CoRL'21)*, London, UK.

Skalse, J., Howe, N., Krasheninnikov, D., and Krueger, D. (2022). Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems (NeurIPS'22)*, New Orleans, LA, USA.

Skinner, B. F. (1953). *Science and Human Behavior*. Science and Human Behavior. Macmillan, Oxford, England.

Sun, W., Venkatraman, A., Gordon, G. J., Boots, B., and Bagnell, J. A. (2017). Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, volume 70 of *Proceedings of Machine Learning Research*, pages 3309–3318, International Convention Centre, Sydney, Australia.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press, 2 edition.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1-2):181–211.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'17)*, pages 23–30, Vancouver, BC, Canada.

Tsai, R. and Lenz, R. (1989). A new technique for fully autonomous and efficient 3D robotics hand/eye calibration. *IEEE Transactions on Robotics and Automation*, 5(3):345–358.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. In *AAAI Conference on Artificial Intelligence*, Pheonix, USA.

Vasan, G., Wang, Y., Shahriar, F., Bergstra, J., Jägersand, M., and Mahmood, A. R. (2024). Revisiting Sparse Rewards for Goal-Reaching Reinforcement Learning. *Reinforcement Learning Journal*, 1(1).

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. (2018). Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards.

Villani, L. and De Schutter, J. (2016). Force Control. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, Springer Handbooks, pages 195–220. Springer International Publishing, Cham.

Wang, C., Zhang, Q., Tian, Q., Li, S., Wang, X., Lane, D., Petillot, Y., and Wang, S. (2020). Learning Mobile Manipulation through Deep Reinforcement Learning. *Sensors (Basel, Switzerland)*, 20(3).

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Welschehold, T., Dornhege, C., and Burgard, W. (2017). Learning Mobile Manipulation Actions from Human Demonstrations. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'17)*, pages 3196–3201, Vancouver, BC, Canada.

Williams, R. J. and Peng, J. (1991). Function Optimization using Connectionist Reinforcement Learning Algorithms. *Connection Science*, 3(3):241–268.

Wu, B., Nair, S., Fei-Fei, L., and Finn, C. (2021). Example-Driven Model-Based Reinforcement Learning for Solving Long-Horizon Visuomotor Tasks. *arXiv:2109.10312 [cs]*.

Wu, Y., Mozifian, M., and Shkurti, F. (2020). Shaping Rewards for Reinforcement Learning with Imperfect Demonstrations using Generative Models. *arXiv:2011.01298 [cs]*.

Xiang, G., Li, S., Shuang, F., Gao, F., and Yuan, X. (2024). SC-AIRL: Share-Critic in Adversarial Inverse Reinforcement Learning for Long-Horizon Task. *IEEE Robotics and Automation Letters*, 9(4):3179–3186.

Yamaguchi, A. and Atkeson, C. G. (2017). Implementing tactile behaviors using FingerVision. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 241–248.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. (2022). Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2019). Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *Conference on Robot Learning (CoRL'19)*, volume 100 of *Proceedings of Machine Learning Research*, pages 1094–1100, Osaka, Japan. PMLR.

Yuan, W. (2014). Tactile Measurement with a GelSight Sensor. Master's thesis, Massachusetts Institute of Technology.

Yuan, W., Dong, S., and Adelson, E. H. (2017). GelSight: High-Resolution Robot Tactile Sensors for Estimating Geometry and Force. *Sensors*, 17(12):2762.

Zeng, A., Song, S., Yu, K.-T., Donlon, E., Hogan, F. R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E., Fazeli, N., Alet, F., Chavan Dafle, N., Holladay, R., Morona, I., Nair, P. Q., Green, D., Taylor, I., Liu, W., Funkhouser, T., and Rodriguez, A. (2022). Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *The International Journal of Robotics Research*, 41(7):690–705.

Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2023). *Dive into Deep Learning*. Cambridge University Press.

Zhang, J. and Cho, K. (2017). Query-Efficient Imitation Learning for End-to-End Simulated Driving. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, pages 2891–2897, New York, NY, USA.

Zhang, T., McCarthy, Z., Jowl, O., Lee, D., Chen, X., Goldberg, K., and Abbeel, P. (2018). Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'18)*, pages 5628–5635, Brisbane, QLD, Australia. IEEE.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum Entropy Inverse Reinforcement Learning. In Fox, D. and Gomes, C. P., editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1433–1438. AAAI Press.

Zolna, K., Reed, S., Novikov, A., Colmenarejo, S. G., Budden, D., Cabi, S., Denil, M., de Freitas, N., and Wang, Z. (2021). Task-Relevant Adversarial Imitation Learning. In *Proceedings of the 2020 Conference on Robot Learning*, pages 247–263.