UNIVERSITY OF CALIFORNIA SAN DIEGO

Learning Generalizable Robot Policies by Understanding Semantics and Logic from Task Demonstrations

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Intelligent Systems, Robotics and Control)

by

Tianyu Wang

2024

University of California San Diego

2024

DEDICATION

I dedicate this thesis to my family: my parents, Yingjun Wang and Mengjuan Li, for encouraging me to pursue my academic dreams and being supportive and understanding even though we are thousands of miles apart. My dear wife, Yanan (Ivy) Zou, for your unwavering support throughout my academic journey. Through all trials and tribulations, you stand by me, trust in me and give me confidence. My son, Gabriel Y. Wang, for bringing so much joy to our family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am especially grateful to my supervisor, Professor Nikolay Atanasov, for your exceptional mentoring and support throughout my PhD career. I was lucky to join your lab and became one of your early students at UCSD. You have always motivated me with insightful advice in our one on one meetings and treated me as an independent researcher. I have learned so much from you, technically and professionally to progress in my future endeavors.

I also appreciate the helpfulness from the committee members: Professor Henrik Christensen, Professor Hao Su, Professor Nuno Vasconcelos, and Professor Michael Yip.

Furthermore, I thank Professor Xiaolong Wang, for his guidance and support. He is a co-author in two of my papers. I would also like express my deepest gratitude to Professor Manfred Morari, Professor Vijay Kumar, and Professor Yaodong Yang for their involvement and assistance in my research. They are each co-authors in one of my papers.

I would like to acknowledge and thank Professor Vikas Dhiman and Steven Chen for spending several late nights before deadlines coding and discussing research topics with me. You inspired me when I was taking the initial steps in my PhD path. I am also thankful to Yuzhe Qin and Binghao Huang, for your time and insightful discussions on robot manipulation.

Lastly, I would like to express my gratitude to all the members of Existential Robotics Lab with whom I had the pleasure to work with, including (alphabetically): Ibrahim Akbar, Abdullah Altawaitan, Arash Asgharivaskasi, Saqib Azim, Dwait Bhatt, Hanwen Cao, Kun Chen, Zhirui Dai, Thai Duong, Qiaojun Feng, Siwei Guo, Nikhil Karnwal, Shumon Koga, Jinzhao Li, Zhichao Li, Kehan Long, Yue Meng, Sumangala Patki, Siddhant Saoji, Mo Shan, Baoqian Wang, Youxing Wang, Yifan Wu, Yinzhuang Yi and Ehsan Zobeidi. It has been an incredible time working with all of you.

Chapter 3, in part, is a reprint of the material as it appears in T. Wang, V. Dhiman and N. Atanasov, “Learning Navigation Cost from Demonstrations in Partially Observable Environments,” IEEE International Conference on Robotics and Automation (ICRA), pp. 4434-4440, 2020. Chapter 3, in part, is a reprint of the material as it appears in T. Wang, V. Dhiman and N. Atanasov, “Learning Navigation Costs from Demonstrations with Semantic Observations,” Learning for Dynamics and Control, pp. 245–255, 2020. Chapter 3, in part, is a reprint of the material as it appears in T. Wang, V. Dhiman and N. Atanasov, “Inverse Reinforcement Learning for Autonomous Navigation via Differentiable Semantic Mapping and Planning,” Autonomous Robots, 47(6), 809-830. The dissertation author is the primary author of these papers. Chapter 3, in part, is a reprint of the material as it appears in T. Wang and N. Atanasov, “Inverse Reinforcement Learning of Autonomous Behaviors Encoded as Weighted Finite Automata”, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 7429-7435, 2022. The dissertation author is the primary author of this paper.

Chapter 4, in part, is currently being prepared for submission for publication of the material as it may appear in T. Wang, N. Karnwal and N. Atanasov, “Latent Policies for Adversarial Imitation Learning.” The dissertation author is the primary author of this paper. Chapter 4, in part, is currently being prepared for submission for publication of the material as it may appear in T. Wang, D. Bhatt, X. Wang and N. Atanasov, “Cross Embodiment Robot Manipulation Skill Transfer from Cycle Consistency.” The dissertation author is the primary author of this paper. Chapter 4, in part, is a reprint of the material as it appears in B. Huang, Y. Chen, T. Wang, Y. Qin, Y. Yang, N. Atanasov, X. Wang, “Dynamic Handover: Throw and Catch with Bimanual Hands, ” Conference on Robot Learning, 2023. The dissertation author is the co-author of this paper.

VITA

2016	Bachelor of Science in Physics, Haverford College
2018	Master of Science in Electrical Engineering (Intelligent Systems, Robotics and Control), University of California San Diego
2024	Doctor of Philosophy in Electrical Engineering (Intelligent Systems, Robotics and Control), University of California San Diego

PUBLICATIONS

Tianyu Wang, Nikhil Karnwal, Nikolay Atanasov, "Latent Policies for Adversarial Imitation Learning", in preparation for submission.

Tianyu Wang, Dwait Bhatt, Xiaolong Wang, Nikolay Atanasov, "Cross Embodiment Robot Manipulation Skill Transfer from Cycle Consistency", in preparation for submission.

Binghao Huang, Yuanpei Chen, **Tianyu Wang**, Yuzhe Qin, Yaodong Yang, Nikolay Atanasov, Xiaolong Wang, "Dynamic Handover: Throw and Catch with Bimanual Hands", *Conference on Robot Learning*, 2023.

Tianyu Wang, Vikas Dhiman, Nikolay Atanasov, "Inverse Reinforcement Learning for Autonomous Navigation via Differentiable Semantic Mapping and Planning", *Autonomous Robots*, 2023.

Tianyu Wang, Nikolay Atanasov, "Inverse Reinforcement Learning of Autonomous Behaviors Encoded as Weighted Finite Automata", *IEEE International Conference on Intelligent Robot and Systems*, 2022.

Steven W. Chen, **Tianyu Wang**, Nikolay Atanasov, Vijay Kumar, Manfred Morari, "Large Scale Model Predictive Control with Neural Networks and Primal Active Sets", *Automatica*, 2022.

Tianyu Wang, Vikas Dhiman, Nikolay Atanasov, "Learning Navigation Costs from Demonstrations with Semantic Observations", *Learning for Dynamics and Control Conference*, 2020.

Tianyu Wang, Vikas Dhiman, Nikolay Atanasov, "Learning Navigation Cost from Demonstrations in Partially Observable Environments", *IEEE International Conference on Robotics and Automation*, 2020.

ABSTRACT OF THE DISSERTATION

Learning Generalizable Robot Policies by Understanding Semantics and Logic from Task Demonstrations

by

Tianyu Wang

Doctor of Philosophy in Electrical Engineering (Intelligent Systems, Robotics and Control)

University of California San Diego, 2024

Professor Nikolay Atanasov, Chair

Autonomous robots have the potential to play a critical role in various aspects of modern life, including search and rescue, autonomous driving, medical surgery, agricultural farms, etc. Reinforcement learning algorithms allow intelligent agents to discover optimal behavior through trial and error from the interactions with the environment and have been successfully applied to playing video games, mastering the game of Go and training large language models. In robotics, this data driven learning approach is also promising for locomotion, manipulation and navigation.

When demonstrations are available, an agent can learn to perform a task by imitating expert behavior. However, the agent has to generalize to novel scenarios that are not seen in training. This thesis introduces two aspects to learn generalizable policies from demonstrations. The first method infers a cost function from semantic and geometric information from observations and can generalize to unseen, dynamic, partially observable simulated environments for autonomous driving scenarios. The second method infers task logic from demonstrations which are in turn used as constraints for motion planning. It exploits the hierarchical logic structure from demonstrated trajectories and can generalize to sequential, compositional planning problems.

Another challenge towards deploying robots in the natural world is the ability to bridge the simulation to reality gap. While simulation provides training data at low cost, the policy should be able to account for mismatches in sensing and actuation when deployed on real robots. To address these challenges, this dissertation introduces a latent space alignment approach where policies trained on a source robot can be adapted to a target robot of different embodiments. Finally, this dissertation also presents a sim-to-real method for throwing and catching objects with bimanual robots, where they need to cooperate precisely to interact with diverse objects at high speed.

Chapter 1

Introduction

Deep reinforcement learning has demonstrated incredible performance on many human-level challenging tasks, from video games like StarCraft and Dota [168, 14] to professional Go games [151, 152]. When modern machine learning techniques learn to excel at specific and well-defined tasks, their real-world applications are limited when generalization gaps exist between training and testing environments [34, 164]. It remains challenging for robots to understand physical entities of the world like humans do and adjust their learned skills to react to environment changes. However, humans have the ability to extrapolate from prior knowledge and experience to generalize across different situations. The ability to distill abstract information enables us to adaptively improve and generalize to unseen scenarios.

Consider teaching a robot to pour a cup of tea by imitating a human expert. The robot needs to understand the semantic and geometric meanings of the physical objects in the scene, e.g., tea kettle, cups, plates and table. From demonstrations, it needs to associate positive rewards for bringing the kettle to a cup and negative rewards for undesired behaviors like tossing plates and knocking over cups. It also needs to layout a task plan and decompose a given high-level objective into subgoals, e.g., fetching the kettle, pouring into the cup, and putting the kettle back on the table. Once the robot understands the concepts for pouring tea, it needs to execute the skills by taking into account the differences between the physical limitations of the human demonstrator and itself. For example, the human hand has five fingers while the robot is equipped

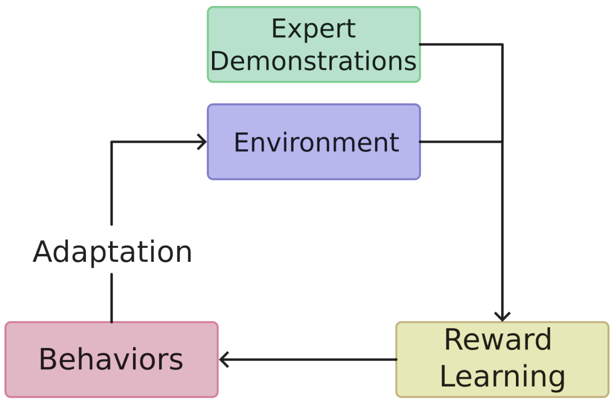


Figure 1.1. A robot development process includes understanding the demonstrated concepts to infer reward functions, developing corresponding skills and behaviors, and adapting to robot and environment changes during deployment.

with a parallel gripper. How should the robot obtain a grasp pose which makes pouring easy when the center of mass changes for the tilted kettle? Inspired by our reasoning process, this dissertation proposes novel deep learning models to infer cost functions from demonstrations which understand the semantics and logic from observations and novel domain adaptation techniques to allow robots to solve real world problems. Fig. 1.1 highlights the core components of this dissertation. The robot first infer the high-level structures and low-level reward functions of the demonstrated task. It then develops skills that are consistent with the learned structure and reward. Finally, the robot accounts for the physical differences in the demonstrator to carry out the skills and imitate the desired behavior. On a high level, the main contributions of this dissertation are as follows:

- First, it introduces novel techniques to learn reward functions from demonstrations, encapsulating semantic relations and logical constraints from observations.
- Second, it introduces novel domain adaptation approaches to address training and deployment mismatch, including robot embodiment difference and sim-to-real gap.

1.1 Learning from Demonstrations

Imitation learning learns a mapping from observations directly to a control policy that matches expert demonstrations. However, it typically suffers from distribution mismatch between training and testing and does not consider long-horizon planning. Inverse reinforcement learning (IRL) aims at recovering a cost function under which the expert demonstrations are optimal. Prior works in IRL include maximum-margin [139] and maximum-entropy [192] formulations. In this dissertation, we investigate how to discover and exploit observation information, such as semantics, geometry and logic, from demonstrated trajectories and design a cost function that implicitly satisfy these constraints. Chapter 3 of this dissertation addresses the problem of learning from demonstrations by understanding semantics and inferring logic. We propose a cost function representation that is learned using semantic features and a differentiable motion planning algorithm that efficiently optimizes the cost function parameters using an objective function that maximizes the likelihood of the expert demonstrated behavior. We also propose to discover hierarchical logic structure from demonstrations and encode as constraints in the differentiable motion planning algorithm. The proposed methods allow us to generalize the control policy to unseen environments and dynamic obstacles by understanding semantic entities and solving long-horizon, sequential and compositional planning problems.

1.2 Generalization from Simulation to Real World

Reinforcement learning algorithms emerge as a promising tool for solving complex decision making and control problems. However, it remains challenging to deploy the trained policy on physically embodied agents. While simulation can provide training data at low cost, it always inherits physical mismatches from reality. Common techniques for bridging this gap include domain randomization [82, 4], where the policy is trained with perturbed simulation settings in order to cover the real world data distribution, and domain adaptation [66, 166, 58], where models learned in the source domain are adapted to a different target domain usually through a unified feature space. Chapter 4 of this dissertation presents several approaches to bridge the gap between training policies in simulation and deploying robots in real world. We first present a framework to learn policy functions in latent action space that captures the task

structure when imitating expert behavior in complex robotic tasks. It allows adversarial imitation learning algorithms to be trained efficiently in the latent structured action space, especially for high-dimensional systems. Next, we present a novel approach which aligns source and target robots with different embodiments in a common latent state and action space which is trained with distribution matching and cycle consistency objectives. Through this latent space, policies trained on the source robot in simulation can be transferred to a real target robot without further finetuning. Finally, we demonstrate a bimanual robot manipulation skill where the policy is trained with reinforcement learning in simulation and sim-to-real transfer is performed on real robots. It precisely throws and catches diverse objects at high speed using two arms with multi-finger hands.

Chapter 2

Background

This dissertation focuses on using inverse reinforcement learning algorithms to infer task logic and reward function from demonstrations and on deploying the algorithms in autonomous robot navigation and robot manipulation applications. This section provides background information on reinforcement learning and inverse reinforcement learning.

2.1 Reinforcement Learning

In reinforcement learning (RL) [158] we consider a Markov decision process which consists of

- a set of states $\mathcal{S}$, where $\mathbf{s} \in \mathcal{S}$ may be either discrete or continuous,
- a set of actions $\mathcal{A}$, where $\mathbf{a} \in \mathcal{A}$ may be either discrete or continuous,
- dynamics function $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ that maps a state-action pair to a distribution of the next state,
- a reward function $r(\mathbf{s}, \mathbf{a})$,
- a scalar discount factor $\gamma \in (0, 1]$ that emphasizes immediate rewards.

In this thesis, we may use the symbol $\mathbf{x} \in \mathcal{X}$ to denote states interchangeably with $\mathbf{s}$, and $\mathbf{u} \in \mathcal{U}$ to denote actions or controls interchangeably with $\mathbf{a}$. Additionally, we define a cost function as

$c(\mathbf{s},\mathbf{a}) = -r(\mathbf{s},\mathbf{a})$. The goal of an RL algorithm is to learn a policy, which defines a distribution over actions conditioned on states, $\pi(\mathbf{a} \mid \mathbf{s})$. A trajectory is a sequence of states and actions of length H, given by $\tau = (\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_H, \mathbf{a}_H)$ where H might be infinite. The trajectory distribution p_π for a given MDP and policy π is given by

$$p_\pi(\tau) = p(\mathbf{s}_0) \prod_{t=0}^{H} \pi(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t). \tag{2.1}$$

The value function of a policy, V^π, can be expressed as an expectation under this trajectory distribution

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\tau \sim p_\pi} \left[\sum_{t=0}^{H} \gamma^t r(\mathbf{s},\mathbf{a}) \mid \mathbf{s}_0 = \mathbf{s} \right], \tag{2.2}$$

and the optimal value function $V^*(\mathbf{s}) = \max_\pi V^\pi(\mathbf{s})$ is the maximal possible long-term return a policy can achieve from a state $\mathbf{s}$. A policy π^* is optimal if $V^{\pi^*}(\mathbf{s}) = V^*(\mathbf{s}) \;\; \forall \mathbf{s}$.

When the state and action spaces are discrete, a popular algorithm for calculating V^* and π^* is value iteration. We initialize $V_0(\mathbf{s})$ randomly and at each iteration k we compute:

$$V_{k+1}(\mathbf{s}) = \max_{\mathbf{a}} Q_k(\mathbf{s},\mathbf{a}) \tag{2.3}$$

$$Q_k(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}'} p(\mathbf{s}' \mid \mathbf{s},\mathbf{a}) V_k(\mathbf{s}'). \tag{2.4}$$

It is well known that the value function V_k converges as $k \to \infty$ [16]. We define a stochastic ε-greedy policy as:

$$\pi_k(\mathbf{s}) = \begin{cases} \text{random } \mathbf{a} \in \mathcal{A} \text{ with probability } \frac{\varepsilon}{\|\mathcal{A}\|} \\ \arg\max_{\mathbf{a}} Q_k(\mathbf{s},\mathbf{a}) \text{ with probability } 1 - \varepsilon \end{cases} \tag{2.5}$$

for small $\varepsilon \geq 0$. The deterministic optimal policy can be obtained as $k \to \infty$ and $\varepsilon \to 0$, i.e.,

$\pi^*(\mathbf{s}) = \arg\max_{\mathbf{a}} Q_\infty(\mathbf{s}, \mathbf{a})$.

When the state and action spaces are continuous, policy gradient methods directly optimize the policy π by computing the value function gradient. In this case, we assume the policy (sometimes referred to as *actor*) is parameterized by a parameter vector θ. Typically, the policy $\pi_\theta(\mathbf{a} \mid \mathbf{s})$ is represented by a deep neural network and θ are the neural network weights. The gradient of the objective with respect to θ can be written as

$$\nabla_\theta V^{\pi_\theta} = \mathbb{E}_{\tau \sim p_{\pi_\theta}} \left[\sum_{t=0}^{H} \gamma^t \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \left(\sum_{t'=t}^{H} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - b(\mathbf{s}_t) \right) \right]. \tag{2.6}$$

The advantage function $A(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{H} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - b(\mathbf{s}_t)$ which estimates the return of the trajectory, can be learned as another neural network (sometimes referred to as *critic*) or it can be simply be estimated with Monte Carlo samples from generated trajectories. The baseline $b(\mathbf{s}_t)$, which depends only on the states and not actions, is used to reduce the variance of the return estimator. Common algorithms for implementing policy gradients include SAC [70], DDPG [110], TD3 [55], etc.

2.2 Inverse Reinforcement Learning

In inverse reinforcement learning (IRL), the reward function of the MDP is not provided. Instead, a set of expert demonstrations $\mathcal{D} = \{\tau_1, \ldots, \tau_N\}$ are supplied. The principal goal of IRL is to infer the underlying reward r of the expert demonstrations, from which we can predict behavior using standard RL algorithms. In maximum entropy inverse reinforcement learning (MaxEnt IRL) [192], the demonstrations are modeled by a Boltzmann distribution, where the energy is given by the reward function r_ϕ:

$$p_\phi(\tau) = \frac{1}{Z_\phi} \exp\left(\sum_{t=0}^{H} r_\phi(\mathbf{s}_t, \mathbf{a}_t)\right). \tag{2.7}$$

Intuitively speaking, the optimal trajectories have the highest likelihood and the expert can generate suboptimal trajectories whose probabilities decrease exponentially as the trajectories become more costly. The partition function Z_ϕ is the integral of $\exp(\sum_{t=0}^{H} r_\phi(\mathbf{s}_t, \mathbf{a}_t))$ over all trajectories that are consistent with the dynamics of the environment. We can interpret the IRL problem as solving the maximum likelihood problem of observing the demonstrated trajectories:

$$\max_\phi \mathbb{E}_{\tau\sim\mathcal{D}}\left[\log p_\phi(\tau)\right] \Leftrightarrow \max_\phi \mathbb{E}_{\tau\sim\mathcal{D}}\left[\sum_{t=0}^{H} r_\phi(\mathbf{s}_t, \mathbf{a}_t) - \log Z_\phi\right]. \tag{2.8}$$

When the state and action spaces are discrete with known dynamics and the reward function is parameterized to be linear in reward features, we can solve the above optimization problem with dynamic programming [192]. When the system is continuous and the reward function is non-linear, importance sampling methods can be used to approximate the partition function Z_ϕ [50].

Chapter 3

Learning Cost Functions from Demonstrations for Robot Navigation

Autonomous systems are expected to achieve reliable performance in increasingly complex environments with increasingly complex objectives. Yet, it is often challenging to design a mathematical formulation that captures all safety and liveness requirements across various operational conditions. Minimizing a misspecified cost function may lead to undesirable performance, regardless of the quality of the optimization algorithm. However, a domain expert is often able to demonstrate desirable or undesirable behavior that implicitly captures the task specifications. In this chapter, we consider inverse reinforcement learning (IRL) problems in which observations containing semantic and logic information about the environment are available.

We consider two motivating scenarios. First, Fig. 3.1 shows an autonomous vehicle navigating in an urban environment. The car is equipped with sensors that can reveal information about the semantic categories of surrounding objects and areas. An expert driver can reason about a course of action based on this contextual information. For example, staying on the road relates to making progress, while hitting the sidewalk or a tree should be avoided. One key challenge in IRL is to infer a cost function when such expert reasoning is not explicit. If reasoning about semantic entities can be learned from the expert demonstrations, the cost model may generalize to new environment configurations. Next, we consider a navigation task in Fig. 3.2, requiring a door to be unlocked before reaching a goal state. Instead of encoding the task requirements as a

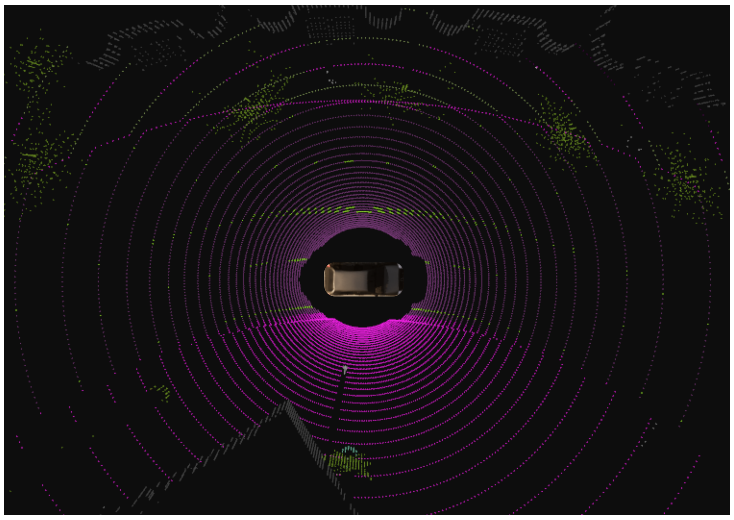


Figure 3.1. Autonomous vehicle in an urban street, simulated via the CARLA simulator [41]. The vehicle is equipped with a LiDAR scanner, four RGB cameras, and a segmentation algorithm, providing a semantically labeled point cloud. An expert driver demonstrates lane keeping (green) and avoidance of sidewalks (pink) and buildings (gray). This chapter considers inferring the expert's cost function and generating behavior that can imitate the expert's response to semantic observations in new operational conditions.

cost function, an expert may provide several demonstrations of navigating to the goal, some of which require picking up the key whenever the door is locked. A reinforcement learning agent should infer the underlying logic sequence of the demonstrated task in order to learn the desired behavior.

Imitation learning (IL) has a long history in reinforcement learning and robotics [142, 7, 5, 128, 190, 137, 125]. The goal is to learn a mapping from observations to a control policy to mimic expert demonstrations. Behavioral cloning [142] is a supervised learning approach that directly maximizes the likelihood of the expert demonstrated behavior. However, it typically suffers from distribution mismatch between training and testing and does not consider long-horizon planning. Another view of IL is through inverse reinforcement learning where the learner recovers a cost function under which the expert is optimal [120, 121, 1]. Recently, [60] and [88] independently developed a unifying probabilistic perspective for common IL algorithms using various f-divergence metrics between the learned and expert policies as minimization objectives. For example, behavioral cloning minimizes the Kullback-Leibler (KL) divergence between the

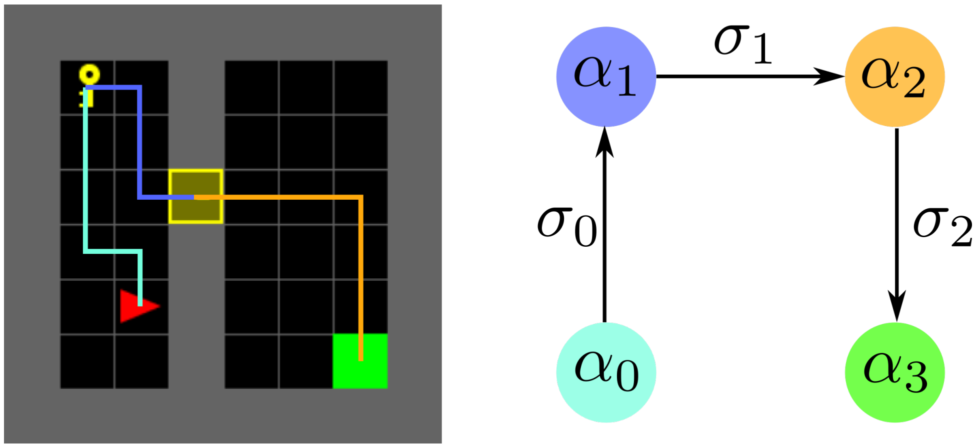


Figure 3.2. (Left) An example trajectory in the MiniGrid environment [28], where an agent has to pick up a key, open the door, and navigate to the goal. (Right) The trajectory can be decomposed into three segments, identified by hidden states $\boldsymbol{\alpha}_t$. Transitions between the high-level states are triggered by events, such as picking up key (σ_0), or opening door (σ_1).

learner and expert policy distribution while adversarial training methods, such as AIRL [53] and GAIL [77] minimize the KL divergence and Jenson Shannon divergence, respectively, between state-control distributions under the learned and expert policies.

Learning a cost function from demonstration requires a control policy that is differentiable with respect to the cost parameters. Computing policy derivatives has been addressed by several successful IRL approaches [120, 139, 192]. Early works assume that the cost is linear in the feature vector and aim at matching the feature expectations of the learned and expert policies. [139] compute subgradients of planning algorithms to guarantee that the expected reward of an expert policy is better than any other policy by a margin. Value iteration networks (VIN) by [161] show that the value iteration algorithm can be approximated by a series of convolution and maxpooling layers, allowing automatic differentiation to learn the cost function end-to-end. [192] develop a dynamic programming algorithm to maximize the likelihood of observed expert data and learn a policy with maximum entropy (MaxEnt). Many works [107, 174, 154] extend MaxEnt to learn a nonlinear cost function using Gaussian Processes or deep neural networks. [50] use a sampling-based approximation of the MaxEnt partition function to learn the cost function under unknown dynamics for high-dimensional continuous systems. However, the cost in most existing work is learned offline using full observation sequences from the expert demonstrations. A major contribution of this chapter is to develop cost representations and planning algorithms that rely only on causal partial observations. In the case where demonstrations are suboptimal with respect to the true cost function, a learned cost function can be recovered with preference-based comparisons [22, 83], self-supervision [24] or human corrections and improvements [12, 81]. In this chapter, we assume that only the demonstrations are provided and we cannot assess the demonstrator's suboptimality with respect to the unknown true cost.

There has been significant progress in semantic segmentation techniques, including deep neural networks for RGB image segmentation [126, 9, 25] and point cloud labeling via spherical depth projection [172, 40, 115, 36]. Maps that store semantic information can be generated from segmented images [146, 111]. [57] and [156] generalize binary occupancy mapping [78] to multi-class semantic mapping in 3D. In this work, we parameterize the navigation cost of an autonomous vehicle as a nonlinear function of such semantic map features to explain expert demonstrations. Achieving safe and robust navigation is directly coupled with the quality of the environment representation and the cost function specifying desirable behaviors. Traditional approaches combine geometric mapping of occupancy probability [78] or distance to the nearest obstacle [122] with hand-specified planning cost functions. Recent advances in deep reinforcement learning demonstrated that control inputs may be predicted directly from sensory observations [105]. However, special model designs [89] that serve as a latent map are needed in navigation tasks where simple reactive policies are not feasible. [68] decompose visual navigation into two separate stages explicitly: mapping the environment from first-person RGB images in local coordinates and planning through the constructed map with VIN [161]. Our model constructs a global map instead and, yet, remains scalable with the size of the environment due to our sparse tensor implementation.

Hierarchical reinforcement learning and options framework [160, 99, 8, 140] are formulations that learn task decomposition and temporal abstraction. Options are high-level macro-actions consisting of primitive actions. [51] introduces a multi-level hierarchical model to discover options from demonstrations where option boundaries are inferred for trajectory segmentation. [92] uses an unsupervised encoder-decoder model to predict subtask segmentation and categorical latent encoding. [176] uses graph recurrent neural networks with relational features between objects for high-level planning and low-level primitive dynamics prediction.

Formal methods have been applied in robotics to prove and guarantee different behavioral properties such as safety and correctness [131, 112, 46, 44]. For example, linear temporal logic (LTL) [11] is used to specify safety and liveness objectives with temporal ordering constraints in control and reinforcement learning problems [97, 98, 44, 45, 17, 52]. Specification mining of LTL formulas can learn finite state automata from execution traces [103, 84]. LTL formulas can also be inferred from Bayesian inference [149] or from graph connectivity of directed acyclic graphs over atomic propositions [31]. We consider weighted finite automata (WFA) in which the transitions carry weights. Whereas classical automata determine whether a word is accepted or rejected, WFA can compute quantitative values as a function of the weighted transitions from the execution of words [43]. WFA offer the expressive power to model quantitative properties, such as resources, time or cost, of the demonstrated behavior. Under certain assumptions of the semiring on which the WFA is defined, it can be shown that WFA is expressively equivalant to weighted monadic second-order (MSO) logic [42, 43].

In this chapter, we first propose an IRL algorithm that learns a cost function from semantic features of the environment. While other works learn a black-box neural network parametrization to map observations directly to costs [174, 154], we take advantage of semantic segmentation and occupancy mapping before inferring the cost function. A metric-semantic map is constructed from causal partial semantic observations of the environment to provide features for cost function learning. Instead of dynamic programming over the entire state space, our formulation allows efficient deterministic search over a subset of promising states. A key advantage of our approach is that this deterministic planning process can be differentiated in closed-form with respect to the parameters of the learnable cost function. Next, we introduce an IRL model that learns to infer high-level task specifications in addition to the low-level control costs to imitate demonstrated behavior. We use a spectral method to learn a WFA which encodes the task logic structure from demonstrations. The agent's interaction with the environment is modeled as a product between the learned WFA and a labeled Markov decision process (L-MDP). We demonstrate that our WFA-IRL method correctly classifies accepting and rejecting sequences and learns a cost function that generalizes the demonstrated behavior to new settings. In summary, the contributions of this chapter are:

- We propose a cost function representation composed of a *map encoder*, capturing semantic class probabilities from online, first-person, distance and semantic observations and a *cost encoder*, defined as a deep neural network over the semantic features.
- We propose to encode the high-level demonstration structure as weighted finite automata, which can be combined with the low-level space for efficient motion planning.
- We propose a new expert model which enables cost parameter optimization with a closed-form subgradient of the cost-to-go, computed only over a subset of promising states.
- We evaluate our models in autonomous navigation experiments in a 2D minigrid environment [28] with multiple semantic categories as well as an autonomous driving task that respects traffic rules in the CARLA simulator [41].

3.1 Differentiable Semantic Mapping and Planning

3.1.1 Problem Formulation

Environment and Agent Models

Consider an agent aiming to reach a goal in an a priori unknown environment with different terrain types. Fig. 3.3 shows a grid-world illustration of this setting. Let $\mathbf{x}_t \in \mathcal{X}$ denote the agent state (e.g., pose, twist, etc.) at discrete time t. In this work, we will consider $\mathbf{x}_t \in SE(2)$ composed of 2D position and orientation. Let $\mathbf{x}_g \in \mathcal{X}$ be the goal state. The agent state evolves according to known deterministic dynamics, $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$, with control input $\mathbf{u}_t \in \mathcal{U}$. The control space $\mathcal{U}$ is assumed finite. Let $\mathcal{K} = \{0, 1, 2, \ldots, K\}$ be a set of class labels, where 0 denotes "free" space and $k \in \mathcal{K} \setminus \{0\}$ denotes a particular semantic class such as road, sidewalk, or car. Let $m^* : \mathcal{X} \to \mathcal{K}$ be a function specifying the *true* semantic occupancy of the environment by labeling states with semantic classes. We implicitly assume that m^* assigns labels to agent positions rather than to other state variables. We do not introduce an output function, mapping an agent state to its position, to simplify the notation. Let $\mathcal{M}$ be the space of possible environment realizations m^*. Let $c^* : \mathcal{X} \times \mathcal{U} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$ be a cost function specifying desirable agent behavior in a given environment, e.g., according to an expert user or an optimal design. We assume that the agent does not have access to either the true semantic map m^* or the true cost function c^*. However, the agent is able to obtain point-cloud observations $\mathbf{P}_t = \{(\mathbf{p}_l, \mathbf{y}_l)\}_l \in \mathcal{P}$ at each step t, where $\mathbf{p}_l$ is the measurement location. In the following sections, we consider $\mathbf{p}_l \in \mathbb{R}^2$ for MiniGrid experiments in Sec. 3.1.3 and $\mathbf{p}_l \in \mathbb{R}^3$ for CARLA experiments in Sec. 3.1.3. The vector of weights $\mathbf{y}_l = \left[y_l^1, \ldots, y_l^K\right]^\top$, where $y_l^k \in \mathbb{R}$, indicates the likelihood that semantic class $k \in \mathcal{K} \setminus \{0\}$ was observed. For example, $\mathbf{y}_l \in \mathbb{R}^K$ can be obtained from the softmax output of a semantic segmentation algorithm [126, 9, 25] that predicts the semantic class of the corresponding measurement location $\mathbf{p}_l$ in an RGBD image. The observed point cloud $\mathbf{P}_t$ depends on the agent state $\mathbf{x}_t$ and the environment realization m^*.

Expert Model

We assume that an expert user or algorithm demonstrates desirable agent behavior in the form of a training set $\mathcal{D} := \left\{(\mathbf{x}_{t,n}, \mathbf{u}^*_{t,n}, \mathbf{P}_{t,n}, \mathbf{x}_{g,n})\right\}_{t=1,n=1}^{T_n,N}$. The training set consists of N demonstrated executions with different lengths T_n for $n \in \{1, \ldots, N\}$. Each demonstration trajectory contains the agent states $\mathbf{x}_{t,n}$, expert controls $\mathbf{u}^*_{t,n}$, and sensor observations $\mathbf{P}_{t,n}$ encountered during navigation to a goal state $\mathbf{x}_{g,n}$.

The design of an IRL algorithm depends on a model of the stochastic control policy $\pi^*(\mathbf{u} \mid \mathbf{x}; c^*, m^*)$ used by the expert to generate the training data $\mathcal{D}$, given the *true* cost c^* and environment m^*. The state of the art relies on the MaxEnt model [192], which assumes that the expert minimizes the weighted sum of the stage cost $c^*(\mathbf{x}, \mathbf{u}; m^*)$ and the negative policy entropy

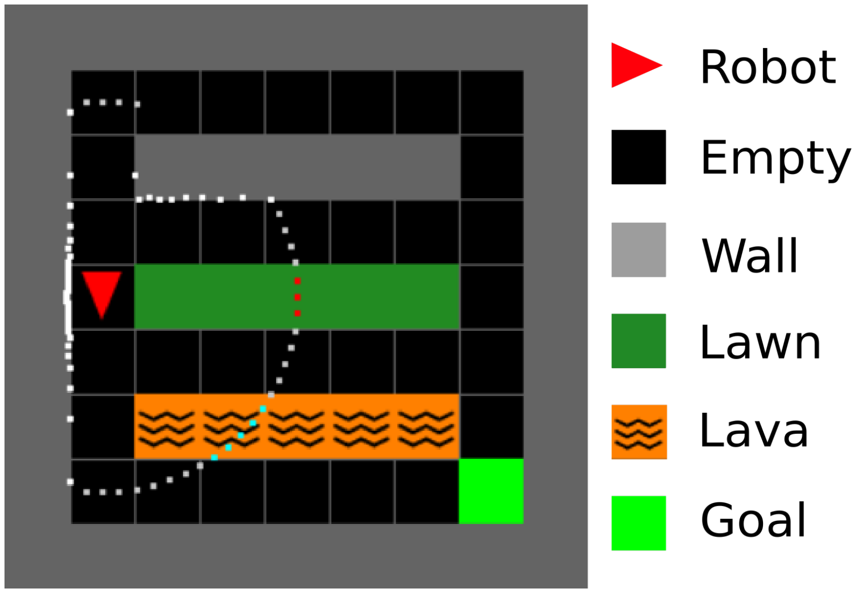


Figure 3.3. A 9×9 grid environment with cells from four semantic classes: empty, wall, lawn, lava. An autonomous agent (red triangle, facing down) starts from the top left corner and is heading towards the goal in the bottom right. The agent prefers traversing the lawn but dislikes lava. LiDAR points detect the semantic labels of the corresponding tiles (gray on empty, white on wall, purple on lawn and cyan on lava).

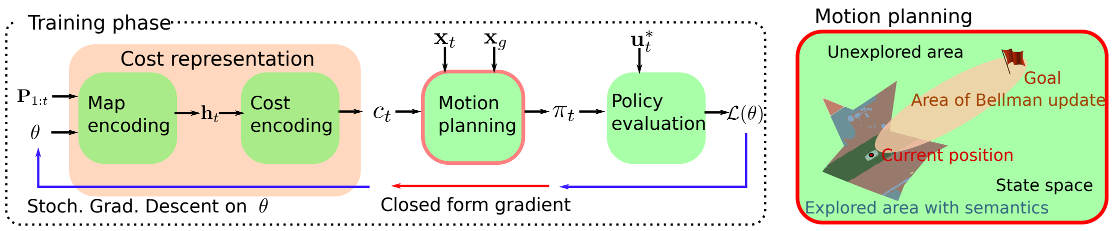


Figure 3.4. Architecture for cost function learning from demonstrations with semantic observations. Our main contribution is a cost representation, combining a probabilistic *semantic map encoder*, with recurrent dependence on semantic observations $\mathbf{P}_{1:t}$, and a *cost encoder*, defined over the semantic features $\mathbf{h}_t$. Efficient forward policy computation and closed-form subgradient backpropagation are used to optimize the cost representation parameters θ in order to explain the expert behavior.

over the agent trajectory.

We propose a new model of expert behavior to explain rational deviation from optimality. We assume that the expert is aware of the optimal value function:

$$Q^*(\mathbf{x}_t, \mathbf{u}_t; c^*, m^*) := \min_{T, \mathbf{u}_{t+1:T-1}} \sum_{k=t}^{T-1} c^*(\mathbf{x}_k, \mathbf{u}_k; m^*) \qquad \text{s.t. } \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k),\ \mathbf{x}_T = \mathbf{x}_g. \tag{3.1}$$

but does not always choose strictly rational actions. Instead, the expert behavior is modeled as a Boltzmann policy over the optimal value function:

$$\pi^*(\mathbf{u}_t \mid \mathbf{x}_t; c^*, m^*) = \frac{\exp(-\frac{1}{\alpha} Q^*(\mathbf{x}_t, \mathbf{u}_t; c^*, m^*))}{\sum_{\mathbf{u} \in \mathcal{U}} \exp(-\frac{1}{\alpha} Q^*(\mathbf{x}_t, \mathbf{u}; c^*, m^*))} \tag{3.2}$$

where α is a temperature parameter. The Boltzmann policy stipulates an exponential preference of controls that incur low long-term costs. We will show in Sec. 3.1.2 that this expert model allows very efficient policy search as well as computation of the policy gradient with respect to the stage cost, which is needed for inverse cost learning. In contrast, the MaxEnt policy requires either value iteration over the full state space [192] or sampling-based estimation of a partition function [50]. Section 3.1.4 provides a comparison between our model and the MaxEnt formulation.

Problem Statement

Given the training set $\mathcal{D}$, our goal is to:

- learn a cost function estimate $c_t : \mathcal{X} \times \mathcal{U} \times \mathcal{P}^{\times} \Theta \to \mathbb{R}_{\geq 0}$ that depends on an observation sequence $\mathbf{P}_{1:t}$ from the true latent environment and is parameterized by $\boldsymbol{\theta} \in \Theta$,
- design a stochastic policy π_t from c_t such that the agent behavior under π_t matches the demonstrations in $\mathcal{D}$.

The optimal value function corresponding to a stage cost estimate c_t is:

$$Q_t(\mathbf{x}_t, \mathbf{u}_t; \mathbf{P}_{1:t}, \boldsymbol{\theta}) := \min_{T, \mathbf{u}_{t+1:T-1}} \sum_{k=t}^{T-1} c_t(\mathbf{x}_k, \mathbf{u}_k; \mathbf{P}_{1:t}, \boldsymbol{\theta}) \text{ s.t. } \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k),\ \mathbf{x}_T = \mathbf{x}_g. \tag{3.3}$$

Following the expert model proposed in Sec. 3.1.1, we define a Boltzmann policy corresponding to Q_t:

$$\pi_t(\mathbf{u}_t \mid \mathbf{x}_t; \mathbf{P}_{1:t}, \boldsymbol{\theta}) \propto \exp(-\frac{1}{\alpha} Q_t(\mathbf{x}_t, \mathbf{u}_t; \mathbf{P}_{1:t}, \boldsymbol{\theta})) \tag{3.4}$$

and aim to optimize the stage cost parameters $\boldsymbol{\theta}$ to match the demonstrations in $\mathcal{D}$.

Problem 1. Given demonstrations $\mathcal{D}$, optimize the cost function parameters $\boldsymbol{\theta}$ so that log-likelihood of the demonstrated controls $\mathbf{u}^*_{t,n}$ is maximized by policy functions $\pi_{t,n}$ obtained according to (3.4):

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) := -\sum_{n=1}^{N} \sum_{t=1}^{T_n} \log \pi_{t,n}(\mathbf{u}^*_{t,n} \mid \mathbf{x}_{t,n}; \mathbf{P}_{1:t,n}, \boldsymbol{\theta}). \tag{3.5}$$

The problem setup is illustrated in Fig. 3.4. An important consequence of our expert model is that the computation of the optimal value function corresponding to a given stage cost estimate is a standard deterministic shortest path (DSP) problem [16]. However, the challenge is to make the value function computation differentiable with respect to the cost parameters $\boldsymbol{\theta}$ in order to propagate the loss in (3.5) back through the DSP problem to update $\boldsymbol{\theta}$. Once the parameters are optimized, the associated agent behavior can be generalized to navigation tasks in new partially observable environments by evaluating the cost c_t based on the observations $\mathbf{P}_{1:t}$ iteratively and re-computing the associated policy π_t.

3.1.2 Cost Function Representation and Learning

We propose a cost function representation with two components: a *semantic occupancy map encoder* with parameters $\boldsymbol{\Psi}$ and a *cost encoder* with parameters $\boldsymbol{\phi}$. The model is differentiable by design, allowing its parameters to be optimized by the subsequent planning algorithm described in Sec. 3.1.2.

Semantic Occupancy Map Encoder

We develop a semantic occupancy map that stores the likelihood of the different semantic categories in $\mathcal{K}$ in different areas of the map. We discretize the state space $\mathcal{X}$ into J cells and let $\mathbf{m} = \left[m^1, \dots, m^J\right]^\top \in \mathcal{K}^J$ be an a priori unknown vector of true semantic labels over the cells. Given the agent states $\mathbf{x}_{1:t}$ and observations $\mathbf{P}_{1:t}$ over time, our model maintains the semantic occupancy posterior $\mathbb{P}(\mathbf{m} = \mathbf{k} \mid \mathbf{x}_{1:t}, \mathbf{P}_{1:t})$, where $\mathbf{k} = \left[k^1, \dots, k^J\right]^\top \in \mathcal{K}^J$. The representation complexity may be reduced significantly if one assumes independence among the map cells m^j: $\mathbb{P}(\mathbf{m} = \mathbf{k} \mid \mathbf{x}_{1:t}, \mathbf{P}_{1:t}) = \prod_{j=1}^{J} \mathbb{P}(m^j = k^j \mid \mathbf{x}_{1:t}, \mathbf{P}_{1:t})$.

We generalize the binary occupancy grid mapping algorithm [163, 78] to obtain incremental Bayesian updates for the mutli-class probability at each cell m^j. In detail, at time $t-1$, we maintain a vector $\mathbf{h}_{t-1,j}$ of class log-odds at each cell and update them given the observation $\mathbf{P}_t$ obtained from state $\mathbf{x}_t$ at time t.

Definition 1. The vector of class log-odds associated with cell m^j at time t is $\mathbf{h}_{t,j} = \left[h^0_{t,j}, \dots, h^K_{t,j}\right]^\top$ with elements:

$$h^k_{t,j} := \log \frac{\mathbb{P}(m^j = k \mid \mathbf{x}_{1:t}, \mathbf{P}_{1:t})}{\mathbb{P}(m^j = 0 \mid \mathbf{x}_{1:t}, \mathbf{P}_{1:t})} \text{ for } k \in \mathcal{K} \,. \tag{3.6}$$

Note that by definition, $h^0_{t,j} = 0$. Applying Bayes rule to (3.6) leads to a recursive Bayesian update for the log-odds vector:

$$\begin{aligned} h^k_{t,j} &= h^k_{t-1,j} + \log \frac{p(\mathbf{P}_t \mid m^j = k, \mathbf{x}_t)}{p(\mathbf{P}_t \mid m^j = 0, \mathbf{x}_t)} \\ &= h^k_{t-1,j} + \sum_{(\mathbf{p}_l, \mathbf{y}_l) \in \mathbf{P}_t} \left(\log \frac{\mathbb{P}(m^j = k \mid \mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l))}{\mathbb{P}(m^j = 0 \mid \mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l))} - h^k_{0,j} \right), \end{aligned} \tag{3.7}$$

where $p(\mathbf{P}_t \mid m^j = k, \mathbf{x}_t)$ is the likelihood of observing $\mathbf{P}_t$ from agent state $\mathbf{x}_t$ when cell m^j has semantic label k. Here, we assume that the observations $(\mathbf{p}_l, \mathbf{y}_l) \in \mathbf{P}_t$ at time t, given the cell m^j

and state $\mathbf{x}_t$, are independent among each other and of the previous observations $\mathbf{P}_{1:t-1}$. The semantic class posterior can be recovered from the log-odds vector $\mathbf{h}_{t,j}$ via a softmax function $\mathbb{P}(m^j = k \mid \mathbf{x}_{1:t}, \mathbf{P}_{1:t}) = \sigma^k(\mathbf{h}_{t,j})$, where $\sigma : \mathbb{R}^{K+1} \to \mathbb{R}^{K+1}$ satisfies:

$$\begin{aligned}
&\sigma(\mathbf{z}) = \left[\sigma^0(\mathbf{z}), \ldots, \sigma^K(\mathbf{z})\right]^\top, \\
&\sigma^k(\mathbf{z}) = \frac{\exp(z^k)}{\sum_{k' \in \mathcal{K}} \exp(z^{k'})}, \\
&\log \frac{\sigma^k(\mathbf{z})}{\sigma^{k'}(\mathbf{z})} = z^k - z^{k'} .
\end{aligned} \tag{3.8}$$

To complete the Bayesian update in (3.7), we propose a parametric inverse observation model, $\mathbb{P}(m^j = k \mid \mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l))$, relating the class likelihood of map cell m^j to a labeled point $(\mathbf{p}_l, \mathbf{y}_l)$ obtained from state $\mathbf{x}_t$.

Definition 2. Consider a labeled point $(\mathbf{p}_l, \mathbf{y}_l)$ observed from state $\mathbf{x}_t$. Let $\mathcal{J}_{t,l} \subset \{1, \ldots, J\}$ be the set of map cells intersected by the sensor ray from $\mathbf{x}_t$ toward $\mathbf{p}_l$. Let m^j be an arbitrary map cell and $d(\mathbf{x}, m^j)$ be the distance between $\mathbf{x}$ and the center of mass of m^j. Define the inverse observation model of the class label of cell m^j as:

$$\mathbb{P}(m^j = k \mid \mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l)) = \begin{cases} \sigma^k(\mathbf{\Psi}_l \bar{\mathbf{y}}_l \delta p_{t,l,j}), & \delta p_{t,l,j} \le \varepsilon, j \in \mathcal{J}_{t,l} \\ \sigma^k(\mathbf{h}_{0,j}), & \text{otherwise,} \end{cases} \tag{3.9}$$

where $\mathbf{\Psi}_l \in \mathbb{R}^{(K+1)\times(K+1)}$ is a learnable parameter matrix, $\delta p_{t,l,j} := d(\mathbf{x}_t, m^j) - \|\mathbf{p}_l - \mathbf{x}_t\|_2$, $\varepsilon > 0$ is a hyperparameter (e.g., set to half the size of a cell), and $\bar{\mathbf{y}}_l := \left[0, \mathbf{y}_l^\top\right]^\top$ is augmented with a trivial observation for the "free" class.

Intuitively, the inverse observation model specifies that cells intersected by the sensor ray are updated according to their distance to the ray endpoint and the detected semantic class probability, while the class likelihoods of other cells remain unchanged and equal to the prior. For example, if m^j is intersected, the likelihood of the class label is determined by a softmax squashing of a linear transformation of the measurement vector $\mathbf{y}_l$ with parameters $\boldsymbol{\Psi}_l$, scaled by the distance $\delta p_{t,l,j}$. Otherwise, Def. 2 specifies an uninformative class likelihood in terms of the prior log-odds vector $\mathbf{h}_{0,j}$ of cell m^j (e.g., $\mathbf{h}_{0,j} = \mathbf{0}$ specifies a uniform prior over the semantic classes).

Definition 3. The log-odds vector of the inverse observation model associated with cell m^j and point observation $(\mathbf{p}_l, \mathbf{y}_l)$ from state $\mathbf{x}_t$ is $\mathbf{g}_j(\mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l))$ with elements:

$$g_j^k(\mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l)) = \log \frac{\mathbb{P}(m^j = k \mid \mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l))}{\mathbb{P}(m^j = 0 \mid \mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l))}\,. \tag{3.10}$$

The log-odds vector of the inverse observation model, $\mathbf{g}_j$, specifies the increment for the Bayesian update of the cell log-odds $\mathbf{h}_{t,j}$ in (3.7). Using the softmax properties in (3.8) and Def. 2, we can express $\mathbf{g}_j$ as:

$$\mathbf{g}_j(\mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l)) = \begin{cases} \boldsymbol{\Psi}_l \bar{\mathbf{y}}_l \delta p_{t,l,j}, & \delta p_{t,l,j} \le \varepsilon, j \in \mathcal{J}_{t,l} \\ \mathbf{h}_{0,j}, & \text{otherwise.} \end{cases} \tag{3.11}$$

Note that the inverse observation model definition in (3.9) resembles a single neural network layer. One can also specify a more expressive multi-layer neural network that maps the observation $\mathbf{y}_l$ and the distance differential $\delta p_{t,l,j}$ along the l-th ray to the log-odds vector:

$$\mathbf{g}_j(\mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l); \boldsymbol{\Psi}_l) = \begin{cases} \mathbf{NN}(\bar{\mathbf{y}}_l, \delta p_{t,l,j}; \boldsymbol{\Psi}_l) & \delta p_{t,l,j} \le \varepsilon, j \in \mathcal{J}_{t,l} \\ \mathbf{h}_{0,j} & \text{otherwise.} \end{cases} \tag{3.12}$$

Proposition 1. *Given a labeled point cloud $\mathbf{P}_t = \{(\mathbf{p}_l, \mathbf{y}_l)\}_l$ obtained from state $\mathbf{x}_t$ at time t, the Bayesian update of the log-odds vector of any map cell m^j is:*

$$\mathbf{h}_{t,j} = \mathbf{h}_{t-1,j} + \sum_{(\mathbf{p}_l, \mathbf{y}_l) \in \mathbf{P}_t} \left[\mathbf{g}_j(\mathbf{x}_t, (\mathbf{p}_l, \mathbf{y}_l)) - \mathbf{h}_{0,j} \right]. \tag{3.13}$$

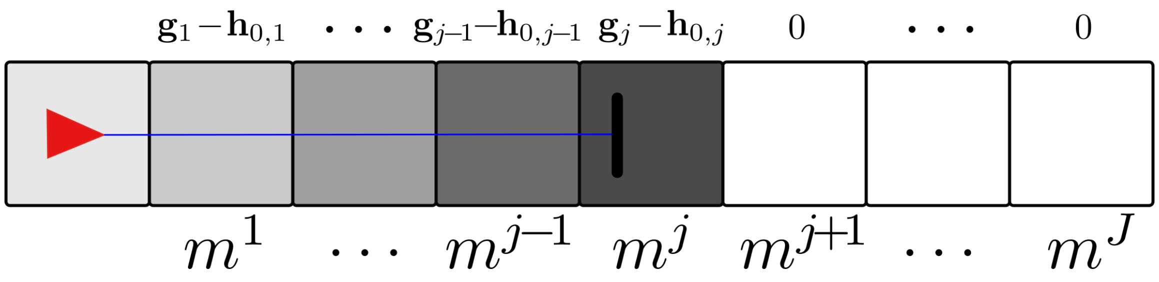

$\mathbf{g}_1 - \mathbf{h}_{0,1}$
$\cdots$
$\mathbf{g}_{j-1} - \mathbf{h}_{0,j-1}$
$\mathbf{g}_j - \mathbf{h}_{0,j}$
0
$\cdots$
0
m^1
$\cdots$
m^{j-1}
m^j
m^{j+1}
$\cdots$
m^J

Figure 3.5. Illustration of the log-odds update in (3.13) for a single point observation. The sensor ray (blue) hits an obstacle (black) in cell m^j. The log-odds increment $\mathbf{g}_j - \mathbf{h}_{0,j}$ on each cell is shown in grayscale.

Fig. 3.5 illustrates the increment of the log-odds vector $\mathbf{h}_{t,j}$ for a single point $(\mathbf{p}_l, \mathbf{y}_l)$. The log-odds are increased more at m^j than other cells far away from the observed point. When ε in (3.12) is set to half the cell size, values of the cells beyond the observed point are unchanged. Fig. 3.6 shows the semantic class probability prediction for the example in Fig. 3.3 using the inverse observation model in Def. 2 and the log-odds update in (3.13).

Cost Encoder

We also develop a cost encoder that uses the semantic occupancy log odds $\mathbf{h}_t$ to define a cost function estimate $c_t(\mathbf{x}, \mathbf{u})$ at a given state-control pair $(\mathbf{x}, \mathbf{u})$. A convolutional neural network (CNN) [62] with parameters $\boldsymbol{\phi}$ can extract cost features from the multi-class occupancy map: $c_t = \mathbf{CNN}(\mathbf{h}_t; \boldsymbol{\phi})$. We adopt a fully convolutional network (FCN) architecture [9] to parameterize the cost function over the semantic class probabilities. The model is a multi-scale architecture that performs downsamples and upsamples to extract feature maps at different layers. Features from multiple scales ensure that the cost function is aware of both local and global context from the semantic map posterior. FCNs are also translation equivariant [35], ensuring that map regions of the same semantic class infer the same cost, irrespective of the specific locations of those regions. Our model architecture (illustrated in Fig. 3.8) consists of a series of convolutional layers with 32 channels, batch normalization [80] and ReLU layers, followed by a max-pooling layer with 2×2 window with stride 2. The feature maps go through another series of convolutional layers with 64 channels, batch normalization, ReLU and max-pooling layers before they are upsampled by reusing the max-pooling indices. The feature maps then go through two series of

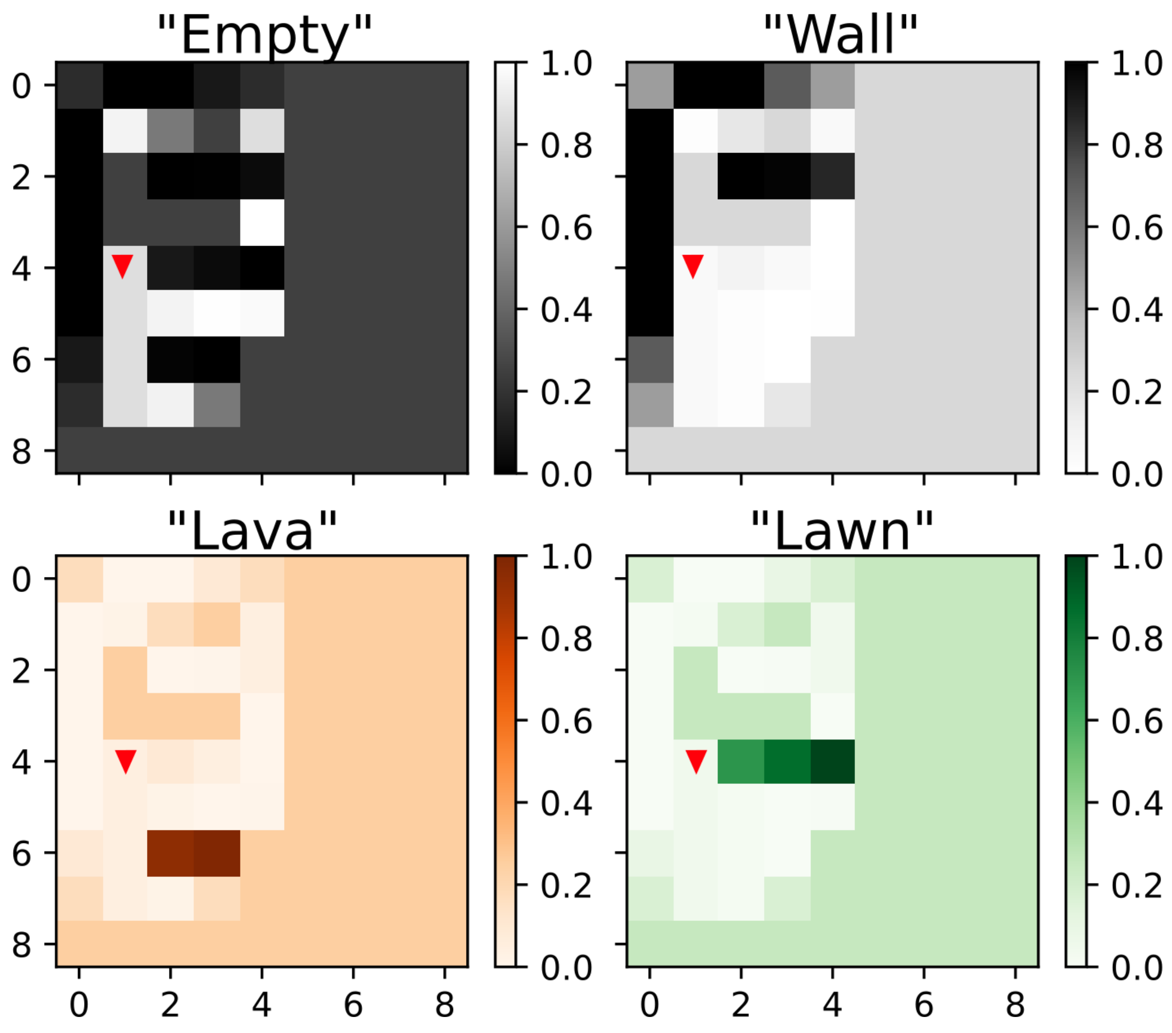


Figure 3.6. The semantic occupancy probability of each class for the example in Fig. 3.3. Using the map encoder described in Sec. 3.1.2, the semantic categories (wall, lawn, lava, etc.) can be identified correctly after training.

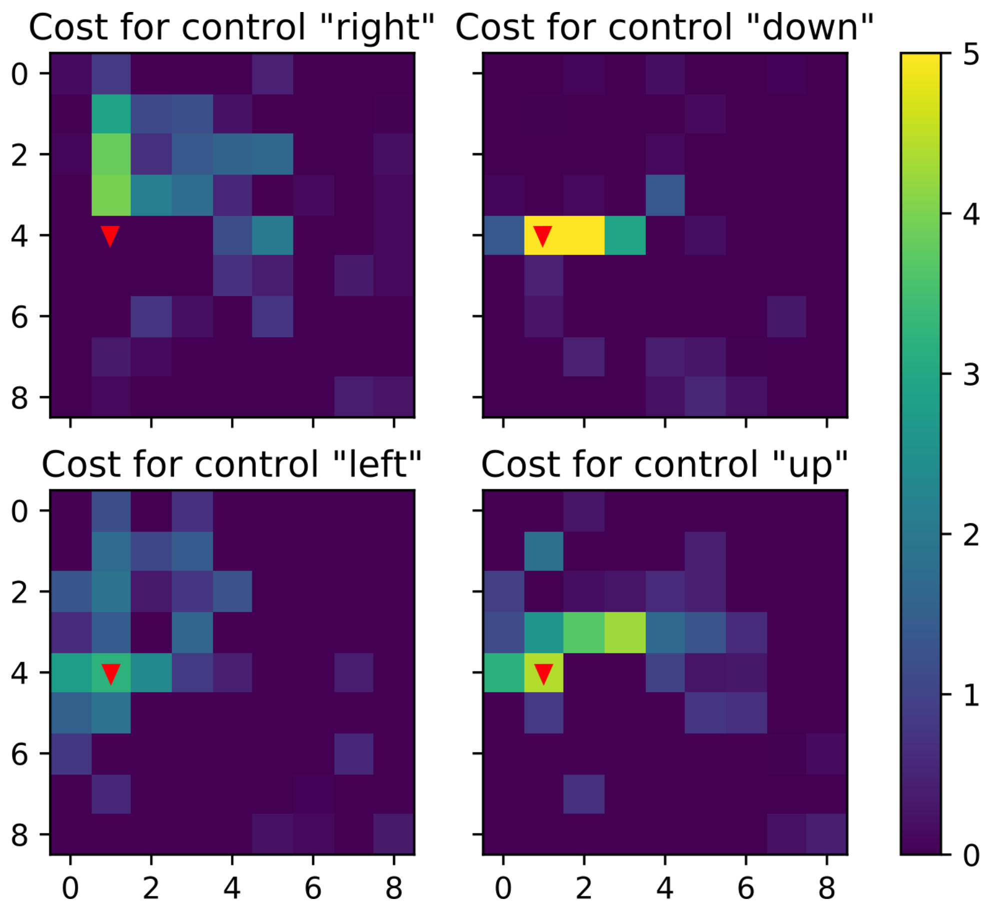


Figure 3.7. The learned cost function for the example in Fig. 3.3. The cost of control "right" is the smallest at the agent's location after training. The agent correctly predicts that it should move right and step on the lawn.

upsampling, convolution, batch normalization and ReLU layers to produce the final cost function c_t. We add a small positive constant to the ReLU output to ensure that $c_t > 0$ and there are no negative cycles or cost-free paths during planning.

In summary, the semantic map encoder (parameterized by $\{\mathbf{\Psi}_l\}_l$) takes the agent state history $\mathbf{x}_{1:t}$ and point cloud observation history $\mathbf{P}_{1:t}$ as inputs to encode a semantic map probability as discussed in Sec. 3.1.2. The FCN cost encoder (parameterized by $\boldsymbol{\phi}$) in turn defines a cost function from the extracted semantic features. The learnable parameters of the cost function, $c_t(\mathbf{x}, \mathbf{u}; \mathbf{P}_{1:t}, \boldsymbol{\theta})$, are $\boldsymbol{\theta} = \{\{\mathbf{\Psi}_l\}_l, \boldsymbol{\phi}\}$.

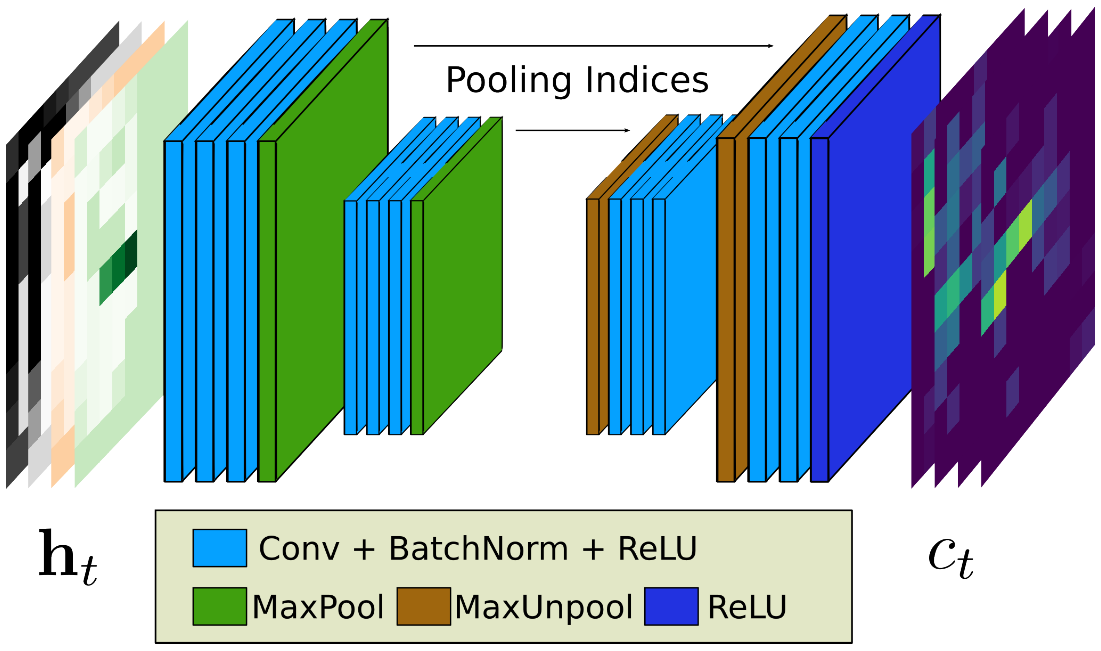


Figure 3.8. A fully convolutional encoder-decoder neural network similar to that in [9] is used as the cost encoder to learn features from semantic map $\mathbf{h}_t$ to cost function c_t.

Cost Learning via Differentiable Planning

We focus on optimizing the parameters $\boldsymbol{\theta}$ of the cost representation $c_t(\mathbf{x},\mathbf{u};\mathbf{P}_{1:t},\boldsymbol{\theta})$ developed in Sec. 3.1.2. Since the true cost c^* is not directly observable, we need to differentiate the loss function $\mathcal{L}(\boldsymbol{\theta})$ in (3.5), which, in turn, requires differentiating through the DSP problem in (3.3) with respect to the cost function estimate c_t.

Previous works rely on dynamic programming to solve the DSP problem in (3.3). For example, the VIN model [161] approximates T iterations of the value iteration algorithm by a neural network with T convolutional and minpooling layers. This allows VIN to be differentiable with respect to the stage cost c_t but it scales poorly with the size of the problem due to the full Bellman backups (convolutions and minpooling) over the state and control space. We observe that it is not necessary to determine the optimal cost-to-go $Q_t(\mathbf{x},\mathbf{u})$ at *every* state $\mathbf{x}\in\mathcal{X}$ and control $\mathbf{u}\in\mathcal{U}$. Instead of dynamic programming, a motion planning algorithm, such as a variant of A* [109] or RRT [101, 85], may be used to solve problem (3.3) efficiently and determine the optimal cost-to-go $Q_t(\mathbf{x},\mathbf{u})$ only over a subset of promising states. The subgradient method of

[150, 139] may then be employed to obtain the subgradient of $Q_t(\mathbf{x}_t, \mathbf{u}_t)$ with respect to c_t along the optimal path.

Deterministic Shortest Path

Given a cost estimate c_t, we use the A* algorithm (Alg. 1) to solve the DSP problem in (3.3) and obtain the optimal cost-to-go Q_t. The algorithm starts the search from the goal state $\mathbf{x}_g$ and proceeds backwards towards the current state $\mathbf{x}_t$. It maintains an *OPEN* set of states, which may potentially lie along a shortest path, and a *CLOSED* list of states, whose optimal value $\min_{\mathbf{u}} Q_t(\mathbf{x}, \mathbf{u})$ has been determined exactly. At each iteration, the algorithm pops a state $\mathbf{x}$ from *OPEN* with the smallest $g(\mathbf{x}) + \varepsilon h(\mathbf{x}_t, \mathbf{x})$ value, where $g(\mathbf{x})$ is an estimate of the cost-to-go from $\mathbf{x}$ to $\mathbf{x}_g$ and $h(\mathbf{x}_t, \mathbf{x})$ is a heuristic function that does not overestimate the true cost from $\mathbf{x}_t$ to $\mathbf{x}$ and satisfies the triangle inequality. We find all predecessor states $\mathbf{x}'$ and their corresponding control $\mathbf{u}'$ that lead to $\mathbf{x}$ under the known dynamics model $\mathbf{x} = f(\mathbf{x}', \mathbf{u}')$ and update their g values if there is a lower cost trajectory from $\mathbf{x}'$ to $\mathbf{x}_g$ through $\mathbf{x}$. The algorithm terminates when all neighbors of the current state $\mathbf{x}_t$ are in the *CLOSED* set. The following relations are satisfied at any time throughout the search:

$$Q_t(\mathbf{x},\mathbf{u}) = c_t(\mathbf{x},\mathbf{u}) + g(f(\mathbf{x},\mathbf{u})), \forall f(\mathbf{x},\mathbf{u}) \in \mathit{CLOSED},$$
$$Q_t(\mathbf{x},\mathbf{u}) \leq c_t(\mathbf{x},\mathbf{u}) + g(f(\mathbf{x},\mathbf{u})), \forall f(\mathbf{x},\mathbf{u}) \notin \mathit{CLOSED}.$$

The algorithm terminates only after all neighbors $f(\mathbf{x}_t, \mathbf{u})$ of the current state $\mathbf{x}_t$ are in *CLOSED* to guarantee that the optimal cost-to-go $Q_t(\mathbf{x}_t, \mathbf{u})$ at $\mathbf{x}_t$ is exact. A simple choice of heuristic that guarantees the above relations is $h(\mathbf{x}, \mathbf{x}') = 0$, which reduces A* to Dijkstra's algorithm. Alternatively, the cost encoder output may be designed to ensure that $c_t(\mathbf{x}, \mathbf{u}) \geq 1$, which allows using Manhattan distance, $h(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_1$, as the heuristic.

Finally, a Boltzmann policy $\pi_t(\mathbf{u} \mid \mathbf{x})$ can be defined using the g values returned by A* for any $\mathbf{x} \in \mathcal{X}$:

$$\pi_t(\mathbf{u} \mid \mathbf{x}) \propto \exp\left(-\frac{1}{\alpha}\left(c_t(\mathbf{x},\mathbf{u}) + g(f(\mathbf{x},\mathbf{u}))\right)\right). \tag{3.14}$$

The policy discourages controls that lead to states outside of *CLOSED* because $c_t(\mathbf{x},\mathbf{u}) + g(f(\mathbf{x},\mathbf{u}))$ overestimates $Q_t(\mathbf{x},\mathbf{u})$. For any unvisited states, the policy is uniform since g values are initialized to infinity. In practice, we only need to query the policy at the current state $\mathbf{x}_t$, which is always in *CLOSED*, for the loss function $\mathcal{L}(\boldsymbol{\theta})$ in (3.5) during training and policy inference during testing.

Algorithm 1. A* motion planning

```
procedure PLAN($\mathbf{x}_t, \mathbf{x}_g, c_t, h, \varepsilon$)
    $OPEN \leftarrow \{\mathbf{x}_g\}$, $CLOSED \leftarrow \{\}$
    $g(\mathbf{x}) \leftarrow \infty, \forall \mathbf{x} \in \mathcal{X}$, $g(\mathbf{x}_g) \leftarrow 0$
    while $\exists \mathbf{u} \in \mathcal{U}$ s.t. $f(\mathbf{x}_t,\mathbf{u}) \notin CLOSED$ do
        Remove $\mathbf{x}$ from $OPEN$ with smallest $g(\mathbf{x}) + \varepsilon h(\mathbf{x}_t,\mathbf{x})$ and insert in $CLOSED$
        for $(\mathbf{x}',\mathbf{u}') \in$ Predecessors($\mathbf{x}$) do
            if $\mathbf{x}' \notin CLOSED$ and $g(\mathbf{x}') > g(\mathbf{x}) + c_t(\mathbf{x}',\mathbf{u}')$ then
                $g(\mathbf{x}') \leftarrow g(\mathbf{x}) + c_t(\mathbf{x}',\mathbf{u}')$
                $CHILD(\mathbf{x}') \leftarrow \mathbf{x}$
                if $\mathbf{x}' \in OPEN$ then
                    Update priority of $\mathbf{x}'$ with $g(\mathbf{x}') + \varepsilon h(\mathbf{x}_t,\mathbf{x}')$
                else
                    $OPEN \leftarrow OPEN \cup \{\mathbf{x}'\}$
procedure PREDECESSORS($\mathbf{x}$)
    return $\{(\mathbf{x}',\mathbf{u}') \in \mathcal{X} \times \mathcal{U} \mid \mathbf{x} = f(\mathbf{x}',\mathbf{u}')\}$
```

Backpropagation through Planning

Having solved the DSP problem in (3.3) for a fixed cost function c_t, we now discuss how to optimize the cost parameters $\boldsymbol{\theta}$ such that the planned policy in (3.14) minimizes the loss in (3.5). Our goal is to compute the gradient $\frac{d\mathcal{L}(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$, using the chain rule, in terms of $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial Q_t(\mathbf{x}_t,\mathbf{u}_t)}$, $\frac{\partial Q_t(\mathbf{x}_t,\mathbf{u}_t)}{\partial c_t(\mathbf{x},\mathbf{u})}$, and $\frac{\partial c_t(\mathbf{x},\mathbf{u})}{\partial \boldsymbol{\theta}}$. The first gradient term can be obtained analytically from (3.5) and (3.4), as we show later, while the third one can be obtained via backpropagation (automatic differentiation) through the neural network cost model $c_t(\mathbf{x},\mathbf{u};\mathbf{P}_{1:t},\boldsymbol{\theta})$ developed in Sec. 3.1.2. We focus on computing the second gradient term.

We rewrite $Q_t(\mathbf{x}_t, \mathbf{u}_t)$ in a form that makes its subgradient with respect to $c_t(\mathbf{x}, \mathbf{u})$ obvious. Let $\mathcal{T}(\mathbf{x}_t, \mathbf{u}_t)$ be the set of trajectories, $\boldsymbol{\tau} = \mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}, \mathbf{u}_{t+1}, \ldots, \mathbf{x}_{T-1}, \mathbf{u}_{T-1}, \mathbf{x}_T$, of length $T - t + 1$ that start at $\mathbf{x}_t$, $\mathbf{u}_t$, satisfy transitions $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$, and terminate at $\mathbf{x}_T = \mathbf{x}_g$. Let $\boldsymbol{\tau}^* \in \mathcal{T}(\mathbf{x}_t, \mathbf{u}_t)$ be an optimal trajectory corresponding to the optimal cost-to-go $Q_t(\mathbf{x}_t, \mathbf{u}_t)$. Define a state-control visitation function which counts the number of times transition $(\mathbf{x}, \mathbf{u})$ appears in $\boldsymbol{\tau}$:

$$\mu_{\boldsymbol{\tau}}(\mathbf{x}, \mathbf{u}) := \sum_{k=t}^{T-1} \mathbb{1}_{(\mathbf{x}_k, \mathbf{u}_k) = (\mathbf{x}, \mathbf{u})}. \tag{3.15}$$

The optimal cost-to-go $Q_t(\mathbf{x}_t, \mathbf{u}_t)$ can be viewed as a minimum over trajectories $\mathcal{T}(\mathbf{x}_t, \mathbf{u}_t)$ of the inner product between the cost function c_t and the visitation function $\mu_{\boldsymbol{\tau}}$:

$$Q_t(\mathbf{x}_t, \mathbf{u}_t) = \min_{\boldsymbol{\tau}} \in \mathcal{T}(\mathbf{x}_t, \mathbf{u}_t) \sum_{\mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U}} c_t(\mathbf{x}, \mathbf{u}) \mu_{\boldsymbol{\tau}}(\mathbf{x}, \mathbf{u}), \tag{3.16}$$

where $\mathcal{X}$ can be assumed finite because both T and $\mathcal{U}$ are finite. We use the subgradient method [150, 139] to compute a subgradient of $Q_t(\mathbf{x}_t, \mathbf{u}_t)$ with respect to c_t.

Lemma 1. *Let $f(\mathbf{x}, \mathbf{y})$ be differentiable and convex in $\mathbf{x}$. Then, $\nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y}^*)$, where $\mathbf{y}^* := \arg\min_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$, is a subgradient of the piecewise-differentiable convex function $g(\mathbf{x}) := \min_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$.*

Applying Lemma 1 to (3.16) leads to the following subgradient of the optimal cost-to-go function:

$$\frac{\partial Q_t(\mathbf{x}_t, \mathbf{u}_t)}{\partial c_t(\mathbf{x}, \mathbf{u})} = \mu_{\boldsymbol{\tau}^*}(\mathbf{x}, \mathbf{u}) \tag{3.17}$$

which can be obtained along the optimal trajectory $\boldsymbol{\tau}^*$ by tracing the *CHILD* relations returned by Alg. 1. Fig. 3.9 shows an illustration of this subgradient computation with respect to the cost estimate in Fig. 3.7 for the example in Fig. 3.3. The result in (3.17) and the chain rule allow us to obtain a complete subgradient of $\mathcal{L}(\boldsymbol{\theta})$.

Proposition 2. *A subgradient of the loss function $\mathcal{L}(\boldsymbol{\theta})$ in* (3.5) *with respect to $\boldsymbol{\theta}$ can be obtained*

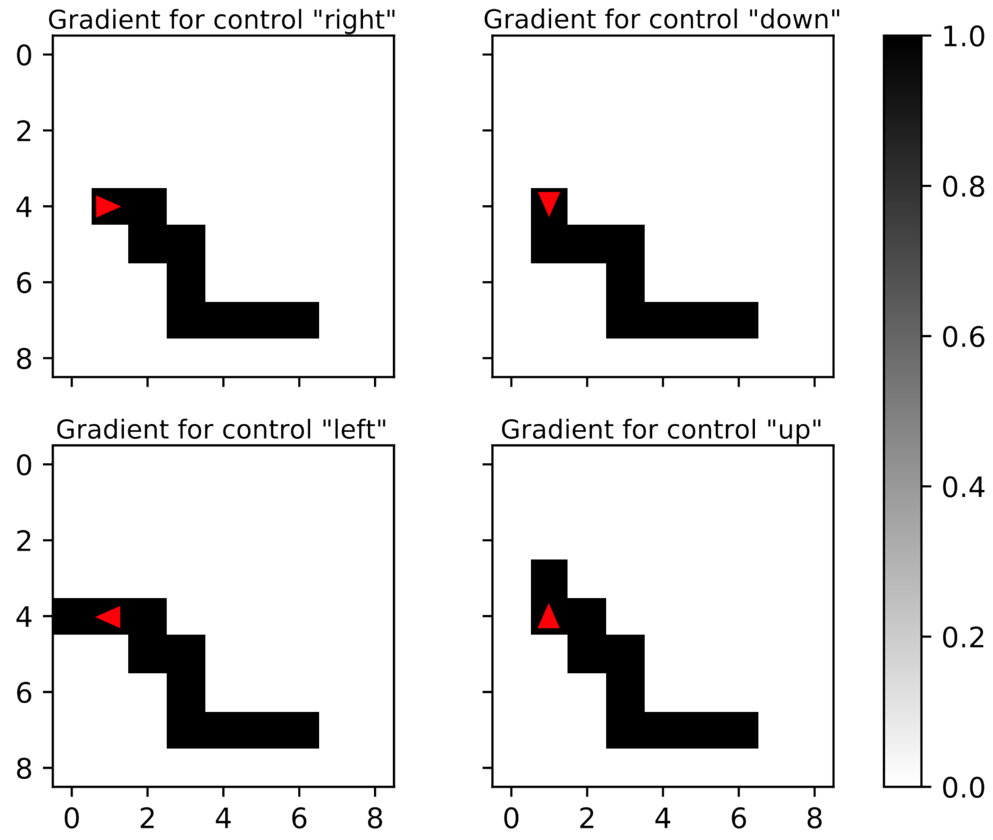


Figure 3.9. Subgradient of the optimal cost-to-go $Q_t(\mathbf{x}_t, \mathbf{u}_t)$ for each control $\mathbf{u}_t$ with respect to the cost $c_t(\mathbf{x}, \mathbf{u})$ in Fig. 3.7.

as:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\sum_{n=1}^{N}\sum_{t=1}^{T_n} \frac{d \log \pi_{t,n}(\mathbf{u}^*_{t,n} \mid \mathbf{x}_{t,n})}{d\boldsymbol{\theta}} = -\sum_{n=1}^{N}\sum_{t=1}^{T_n}\sum_{\mathbf{u}_{t,n}\in\mathcal{U}} \frac{d \log \pi_{t,n}(\mathbf{u}^*_{t,n} \mid \mathbf{x}_{t,n})}{dQ_{t,n}(\mathbf{x}_{t,n},\mathbf{u}_{t,n})}\frac{dQ_{t,n}(\mathbf{x}_{t,n},\mathbf{u}_{t,n})}{d\boldsymbol{\theta}}$$

$$= -\sum_{n=1}^{N}\sum_{t=1}^{T_n}\sum_{\mathbf{u}_{t,n}\in\mathcal{U}} \frac{1}{\alpha}\left(\mathbb{1}_{\{\mathbf{u}_{t,n}=\mathbf{u}^*_{t,n}\}} - \pi_{t,n}(\mathbf{u}_{t,n} \mid \mathbf{x}_{t,n})\right) \times \sum_{(\mathbf{x},\mathbf{u})\in\boldsymbol{\tau}^*} \frac{\partial Q_{t,n}(\mathbf{x}_{t,n},\mathbf{u}_{t,n})}{\partial c_t(\mathbf{x},\mathbf{u})}\frac{\partial c_t(\mathbf{x},\mathbf{u})}{\partial \boldsymbol{\theta}} \quad (3.18)$$

Algorithms

The computation graph implied by Prop. 2 is illustrated in Fig. 3.4. The graph consists of a cost representation layer and a differentiable planning layer, allowing end-to-end minimization of $\mathcal{L}(\boldsymbol{\theta})$ via stochastic subgradient descent. The training algorithm for solving Problem 1 is shown in Alg. 2. The testing algorithm that enables generalizing the learned semantic mapping and planning behavior to new sensory data in new environments is shown in Alg. 3.

Algorithm 2. Train cost parameters $\boldsymbol{\theta}$

```
Input: Dataset $\mathcal{D} = \{(\mathbf{x}_{t,n}, \mathbf{u}^*_{t,n}, \mathbf{P}_{t,n}, \mathbf{x}_{g,n})\}_{t=1,n=1}^{T_n,N}$
1: while $\boldsymbol{\theta}$ not converged do
2:     $\mathcal{L}(\boldsymbol{\theta}) \leftarrow 0$
3:     for $n = 1,\ldots,N$ and $t = 1,\ldots,T_n$ do
4:         Update $c_{t,n}$ using $\mathbf{x}_{t,n}$ and $\mathbf{P}_{t,n}$ as in Sec. 3.1.2
5:         Get $Q_{t,n}(\mathbf{x}_{t,n},\mathbf{u})$ from Alg. 1 with cost $c_{t,n}$
6:         Get $\pi_{t,n}(\mathbf{u} \mid \mathbf{x}_{t,n})$ in (3.4) from $Q_{t,n}(\mathbf{x}_{t,n},\mathbf{u})$
7:         $\mathcal{L}(\boldsymbol{\theta}) \leftarrow \mathcal{L}(\boldsymbol{\theta}) - \log \pi_{t,n}(\mathbf{u}^*_{t,n} \mid \mathbf{x}_{t,n})$
       Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla\mathcal{L}(\boldsymbol{\theta})$ via Prop. 2
Output: Trained cost function parameters $\boldsymbol{\theta}$
```

Sparse Tensor Implementation

In this section, we propose a sparse tensor implementation of the map and cost variables introduced in Sec. 3.1.2. The region explored during a single navigation trajectory is usually a small subset of the full environment due to the agent's limited sensing range. The map and cost variables $\mathbf{h}_t$, $\mathbf{g}_t$, $c_t(\mathbf{x},\mathbf{u})$ thus contains many 0 elements corresponding to "free" space or unexplored regions and only a small subset of the states in $c_t(\mathbf{x},\mathbf{u})$ are queried during planning

Algorithm 3. Test control policy π_t

```
Input: Start state $\mathbf{x}_s$, goal state $\mathbf{x}_g$, cost parameters $\boldsymbol{\theta}$
1: Current state $\mathbf{x}_t \leftarrow \mathbf{x}_s$
2: while $\mathbf{x}_t \neq \mathbf{x}_g$ and navigation not failed do
3:     Make an observation $\mathbf{P}_t$
4:     Update $c_t$ using $\mathbf{x}_t$ and $\mathbf{P}_t$ as in Sec. 3.1.2
5:     Get $Q_t(\mathbf{x}_t, \mathbf{u})$ from Alg. 1 with cost $c_t$
6:     Get $\pi_t(\mathbf{u} \mid \mathbf{x}_t)$ in (3.4) from $Q_t(\mathbf{x}_t, \mathbf{u})$
7:     $\mathbf{x}_t \leftarrow f(\mathbf{x}_t, \mathbf{u}_t)$ with $\mathbf{u}_t := \arg\max_{\mathbf{u}} \pi_t(\mathbf{u} \mid \mathbf{x}_t)$
Output: Navigation succeeds or fails at $\mathbf{x}_t$
```

and parameter optimization in Sec. 3.1.2. Representing these variables as dense matrices is computationally and memory inefficient. Instead, we propose an implementation of the map encoder and cost encoder that exploits the sparse structure of these matrices. [32] developed the Minkowski Engine, an automatic differentiation neural network library for sparse tensors. This library is tailored for our case as we require automatic differentiation for operations among the variables $\mathbf{h}_t$, $\mathbf{g}_t$, c_t in order to learn the cost parameters $\boldsymbol{\theta}$.

During training, we pre-compute the variable $\delta p_{t,l,j}$ over all points $\mathbf{p}_l$ from a point cloud $\mathbf{P}_t$ and all grid cells m^j. This results in a matrix $\mathbf{R}_t \in \mathbb{R}^{K \times J}$ where the entry corresponding to cell m^j stores the vector $\mathbf{y}_l \delta p_{t,l,j}$[1]. The matrix $\mathbf{R}_t$ is then converted to COOrdinate list (COO) format [162], specifying the nonzero indices $\mathbf{C}_t \in \mathbb{R}^{N_{nz} \times 1}$ and their feature values $\mathbf{F}_t \in \mathbb{R}^{N_{nz} \times K}$, where $N_{nz} \ll J$ if $\mathbf{R}_t$ is sparse. To construct $\mathbf{C}_t$ and $\mathbf{F}_t$, we append non-zero features $\mathbf{y}_l \delta p_{t,l,j}$ to $\mathbf{F}_t$ and their coordinates j in $\mathbf{R}_t$ to $\mathbf{C}_t$. The inverse observation model log-odds $\mathbf{g}_t$ can be computed from $\mathbf{C}_t$ and $\mathbf{F}_t$ via (3.11) and represented in COO format as well. Hence, a sparse representation of the semantic occupancy log-odds $\mathbf{h}_t$ can be obtained by accumulating $\mathbf{g}_t$ over time via (3.13).

We use the sparse tensor operations (e.g., convolution, batch normalization, pooling, etc.) provided by the Minkowski Engine in place of their dense tensor counterparts in the cost encoder defined in Sec. 3.1.2. For example, the convolution kernel does not slide sequentially over each entry in a dense tensor but is defined only over the indices in $\mathbf{C}_t$, skipping computations at the 0

[1]In our experiments, we found that storing only $\mathbf{y}_l$ at the cell m^j where p_l lies, instead of along the sensor ray, does not degrade performance.

elements. To ensure that the sparse tensors are compatible in the backpropagtion step of the cost parameter learning (Sec. 3.1.2), the analytic subgradient in (3.18) should also be provided in sparse COO format. We implement a custom operation in which the forward function computes the cost-to-go $Q_t(\mathbf{x}_t, \mathbf{u}_t)$ from $c_t(\mathbf{x}, \mathbf{u})$ via Alg. 1 and the backward function multiplies the sparse matrix $\frac{\partial Q_t(\mathbf{x}_t, \mathbf{u}_t)}{\partial c_t(\mathbf{x}, \mathbf{u}})$ with the previous gradient in the computation graph, $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial Q_t(\mathbf{x}_t, \mathbf{u}_t)}$, to get $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial c_t(\mathbf{x}, \mathbf{u})}$. The output gradient $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial c_t(\mathbf{x}, \mathbf{u})}$ is used as input to the downstream operations defined in Sec. 3.1.2 and Sec. 3.1.2 to update the cost parameters $\boldsymbol{\theta}$.

3.1.3 Results

MiniGrid Experiment

We first demonstrate our inverse reinforcement learning approach in a synthetic minigrid environment [28]. We consider a simplified setting to help visualize and understand the differentiable semantic mapping and planning components[2]. A more realistic autonomous driving setting is demonstrated in Sec. 3.1.3.

Table 3.1. Validation and test results for the 16×16 and 64×64 minigrid environments. We report the negative log-likelihood (*NLL*) and prediction accuracy (*Acc*) of the validation set expert controls and the trajectory success rate (*TSR*) and modified Hausdorff distance (*MHD*) between the agent and the expert trajectories on the test set. See Sec. 3.1.3 for precise definitions of the metrics.

	16×16				64×64			
Model	*NLL*	*Acc* (%)	*TSR* (%)	*MHD*	*NLL*	*Acc* (%)	*TSR* (%)	*MHD*
`DeepMaxEnt`	0.333	87.7	85.5	0.783	0.160	92.5	86.3	2.305
`Ours`	0.247	91.9	93.0	0.208	0.153	95.2	95.6	1.097

Environment: Grid environments of sizes 16×16 and 64×64 are generated by sampling a random number of random length rectangles with semantic labels from $\mathcal{K} := \{empty, wall, lava, lawn\}$. One such environment is shown in Fig. 3.10. The agent motion is modeled over a 4-connected grid such that a control $\mathbf{u}_t$ from $\mathcal{U} := \{up, down, left, right\}$ causes a transition from

[2]Our code for the minigrid experiments is open-sourced at https://github.com/tianyudwang/sirl.

$\mathbf{x}_t$ to one of the four neighboring tiles $\mathbf{x}_{t+1}$. A *wall* tile is not traversable and a transition to it does not change the agent's position.

Sensor: At each step t, the agent receives 72 labeled points $\mathbf{P}_t = \{\mathbf{p}_l, \mathbf{y}_l\}_l$, obtained from ray-tracing a 360° field of view at angular resolution of 5° with maximum range of 3 grid cells and returning the grid location $\mathbf{p}_l$ of the hit point and its semantic class encoded in a one-hot vector $\mathbf{y}_l$. See Fig. 3.3 for an illustration. The sensing range is smaller than the environment size, making the environment only partially observable at any given time.

Demonstrations: Expert demonstrations are obtained by running a shortest path algorithm on the true map $\mathbf{m}^*$, where the cost of arriving at an *empty*, *wall*, *lava*, or *lawn* tile is 1, 100, 10, 0.5, respectively. We generate 10000, 1000, and 1000 random map configurations for training, validation, and testing, respectively. Start and goal locations are randomly assigned and maps without a feasible path are discarded. To avoid overfitting, we use the model parameters that perform best in validation for testing.

Models

`DeepMaxEnt`: We use the `DeepMaxEnt` IRL algorithm of [174] as a baseline. `DeepMaxEnt` is an extension of the MaxEnt IRL algorithm [192], which uses a deep neural network to learn a cost function directly from LiDAR observations. In contrast to our model, `DeepMaxEnt` does not have an explicit map representation. The cost representation is a multi-scale FCN [174] adapted to the 16×16 and 64×64 domains. Value iteration over the cost matrix is approximated by a finite number of Bellman backup iterations, equal to the number of map cells. The original experiments in [174] use the mean and variance of the height of 3D LiDAR points in each cell, as well as a binary indicator of cell visibility, as input features to the FCN neural network. Since our synthetic experiments are in 2D, the point count in each grid cell is used instead of the height mean and variance. This is a fair adaptation since [174] argued that obstacles generally represent areas of larger height variance which corresponds to more points within obstacles cells for our observations. We compare against the original `DeepMaxEnt` model in Sec. 3.1.3.

`Ours`: Our model takes as inputs the semantic point cloud $\mathbf{P}_t$ and the agent position $\mathbf{x}_t$ at each time step and updates the semantic map probability via Sec. 3.1.2. The cost encoder goes through two scales of convolution and down(up)-sampling as introduced in Sec. 3.1.2. The models are trained using the Adam optimizer [91] in Pytorch [129]. The neural network model training and online inference during testing are performed on an Intel i7-7700K CPU and an NVIDIA GeForce GTX 1080Ti GPU.

Evaluation Metrics

The following metrics are used for evaluation: negative log-likelihood (*NLL*) and control accuracy (*Acc*) for the validation set and trajectory success rate (*TSR*) and modified Hausdorff distance (*MHD*) for the test set. Given learned cost parameters $\boldsymbol{\theta}^*$ and a validation set $\mathcal{D} = \left\{(\mathbf{x}_{t,n}, \mathbf{u}^*_{t,n}, \mathbf{P}_{t,n}, \mathbf{x}_{g,n})\right\}_{t=1,n=1}^{T_n,N}$, policies $\pi_{t,n}(\cdot \mid \mathbf{x}_{t,n}; \mathbf{P}_{1:t,n}, \boldsymbol{\theta}^*)$ are computed online via Alg. 1 at each demonstrated state $\mathbf{x}_{t,n}$ and are evaluated according to:

$$\begin{aligned} NLL(\boldsymbol{\theta}^*, \mathcal{D}) &= -\frac{1}{\sum_{n=1}^{N} T_n} \sum_{n=1,t=1}^{N,T_n} \log \pi_{t,n}(\mathbf{u}^*_{t,n} \mid \mathbf{x}_{t,n}; \mathbf{P}_{1:t,n}, \boldsymbol{\theta}^*) \\ Acc(\boldsymbol{\theta}^*, \mathcal{D}) &= \frac{1}{\sum_{n=1}^{N} T_n} \sum_{n=1,t=1}^{N,T_n} \mathbb{1}_{\left\{\mathbf{u}^*_{t,n} = \arg\max_{\mathbf{u}} \pi_{t,n}(\mathbf{u} \mid \mathbf{x}_{t,n}; \mathbf{P}_{1:t,n}, \boldsymbol{\theta}^*)\right\}}. \end{aligned} \tag{3.19}$$

In the test set, the agent is initialized at the starting pose and iteratively applies control inputs $\mathbf{u}_{t,n} = \arg\max_{\mathbf{u}} \pi_{t,n}(\mathbf{u} \mid \mathbf{x}_{t,n}; \mathbf{P}_{1:t,n}, \boldsymbol{\theta}^*)$ as described in Alg. 3. The agent trajectories can deviate from expert trajectories and the agent has to recover from states which were not encountered by the expert. To find whether the agent eventually reaches the goal, we report the success rate *TSR*, where success is defined as reaching the goal state $\mathbf{x}_{g,n}$ within twice the number of steps of the expert trajectory. In addition, *MHD* compares how far the agent trajectories τ_n^A deviate from the expert trajectories $\boldsymbol{\tau}_n^E$:

$$MHD(\left\{\boldsymbol{\tau}_n^A\right\}, \left\{\boldsymbol{\tau}_n^E\right\}) = \frac{1}{N} \sum_{n=1}^{N} \max\left\{\frac{1}{T^A} \sum_{t=1}^{T^A} d(\boldsymbol{\tau}_{t,n}^A, \boldsymbol{\tau}_n^E), \frac{1}{T^E} \sum_{t=1}^{T^E} d(\boldsymbol{\tau}_{t,n}^E, \boldsymbol{\tau}_n^A)\right\}, \tag{3.20}$$

where $d(\boldsymbol{\tau}_{t,n}^{A}, \boldsymbol{\tau}_{n}^{E})$ is the minimum Euclidean distance from the state $\boldsymbol{\tau}_{t,n}^{A}$ at time t in $\boldsymbol{\tau}_{n}^{A}$ to any state in $\boldsymbol{\tau}_{n}^{E}$.

MiniGrid Results

The results are shown in Table. 3.1. Our model outperforms `DeepMaxEnt` in every metric. Specifically, low *NLL* on the validation set indicates that map encoder and cost encoder in our model are capable of learning a cost function that matches the expert demonstrations. During testing in unseen environment configurations, our model also achieves a higher score in successfully reaching the goal. In addition, the difference in the agent trajectory and the expert trajectory is smaller, as measured by the *MHD* metric.

The outputs of our model components, i.e., map encoder, cost encoder and subgradient computation, are visualized in Fig. 3.10. The map encoder integrates past observations and holds a correct estimate of the semantic probability of each cell. The subgradients in the last column enable us to propagate the negative log-likelihood of the expert controls back to the cost model parameters. The cost visualizations indicate that the learned cost function correctly assigns higher costs to *wall* and *lava* cells (in brighter scale) and lower costs to *lawn* cells (in darker scale).

Inference Speed

The problem setting in this section requires the agent to replan at each step when a new observation $\mathbf{P}_t$ arrives and updates the cost function c_t. Our planning algorithm is computationally efficient because it searches only through a subset of promising states to obtain the optimal cost-to-go $Q_t(\mathbf{x}_t, \mathbf{u}_t)$. On the other hand, the value iteration in `DeepMaxEnt` has to perform Bellman backups on the entire state space even though most of the environment is not visited and the cost in these unexplored regions is inaccurate. Table. 3.2 shows the average inference speed to predict a new control $\mathbf{u}_t$ at each step during testing.

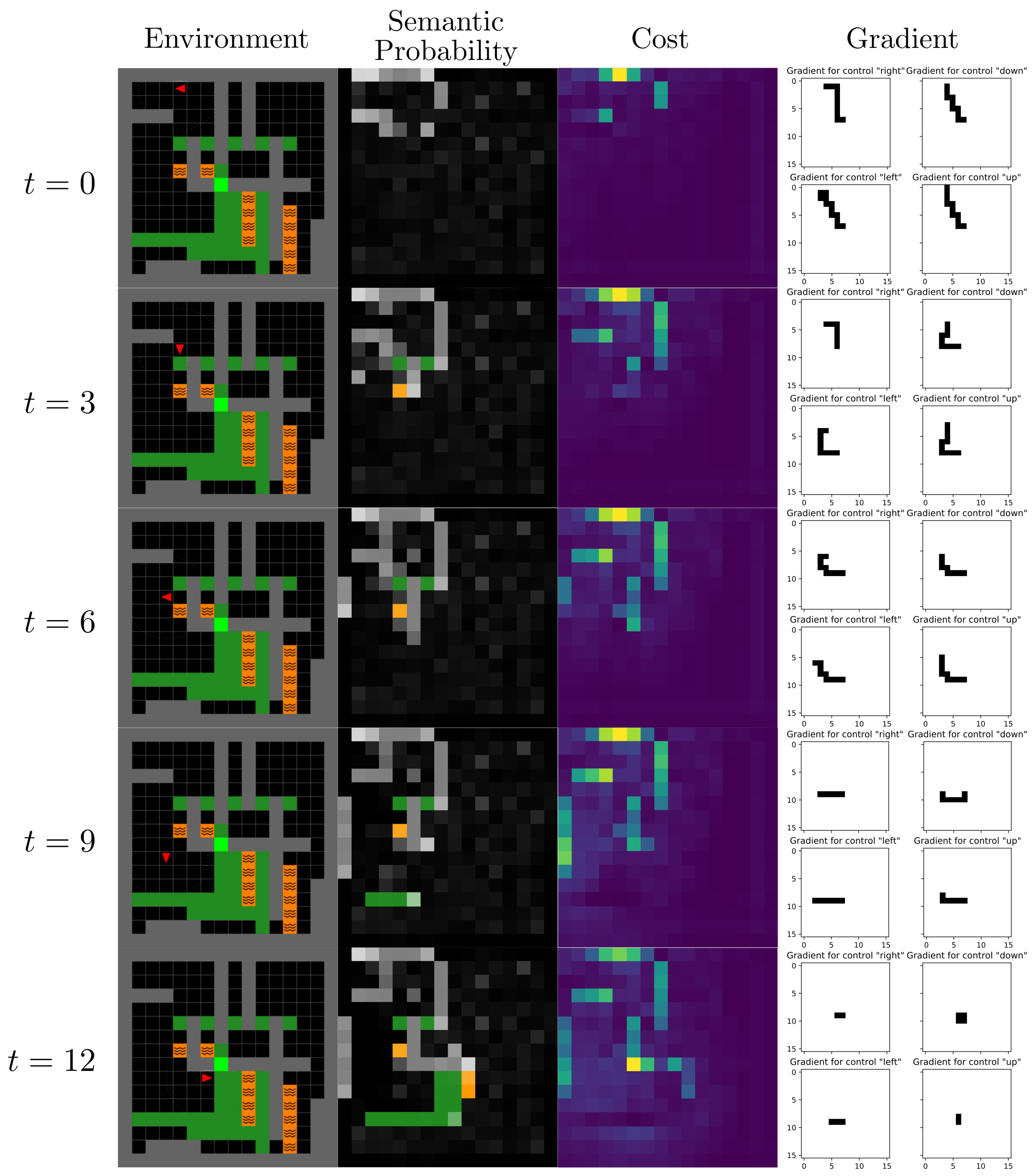


Figure 3.10. Examples of probabilistic multi-class occupancy estimation, cost encoder output, and subgradient computation. The first column shows the agent in the true environment at different time steps. The second column shows the semantic occupancy estimates of the different cells. The third column shows the predicted cost of arriving at each cell. Note that the learned cost function correctly assigns higher costs (in brighter scale) to *wall* and *lava* cells and lower costs (in darker scale) to *lawn* cells. The last column shows subgradients obtained via 3.17 during backpropagation to update the cost parameters.

Table 3.2. Average inference speed comparison between our model and `DeepMaxEnt` for predicting one control in testing.

Grid size	16×16	64×64
`DeepMaxEnt`	5.8 ms	19.7 ms
`Ours`	2.7 ms	3.1 ms

CARLA Experiment

Building on the insights developed in the 2D minigrid environment in Sec. 3.1.3, we design an experiment in a realistic autonomous driving simulation.

Environment: We evaluate our approach using the CARLA simulator (0.9.9) [41], which provides high-fidelity autonomous vehicle simulation in urban environments. Demonstration data is collected from maps $\{Town01, Town02, Town03\}$, while $Town04$ is used for validation and $Town05$ for testing. $Town05$ includes different street layouts (e.g., intersections, buildings and freeways) and is larger than the training and validation maps.

Sensors: The vehicle is equipped with a LiDAR sensor that has 20 meters maximum range and 360° horizontal field of view. The vertical field of view ranges from 0° (facing forward) to $-40°$ (facing down) with 5° resolution. A total of 56000 LiDAR rays are generated per scan $\mathbf{P}_t$ and point measurements are returned only if a ray hits an obstacle (see Fig. 3.11). The vehicle is also equipped with 4 semantic segmentation cameras that detect 13 different classes, including *road*, *road line*, *sidewalk*, *vegetation*, *car*, *building*, etc. The cameras face front, left, right, and rear, each capturing a 90° horizontal field of view (see Fig. 3.11). The semantic label of each LiDAR point is retrieved by projecting the point in the camera's frame and querying the pixel value in the segmented image.

Demonstrations: In each map, we collect 100 expert trajectories by running an autonomous navigation agent provided by the CARLA Python API. On the graph of all available waypoints, the expert samples two waypoints as start and goal and searches the shortest path as a list of waypoints. The expert uses a PID controller to generate a smooth and continuous trajectory to connect the waypoints on the shortest path. The expert respects traffic rules, such

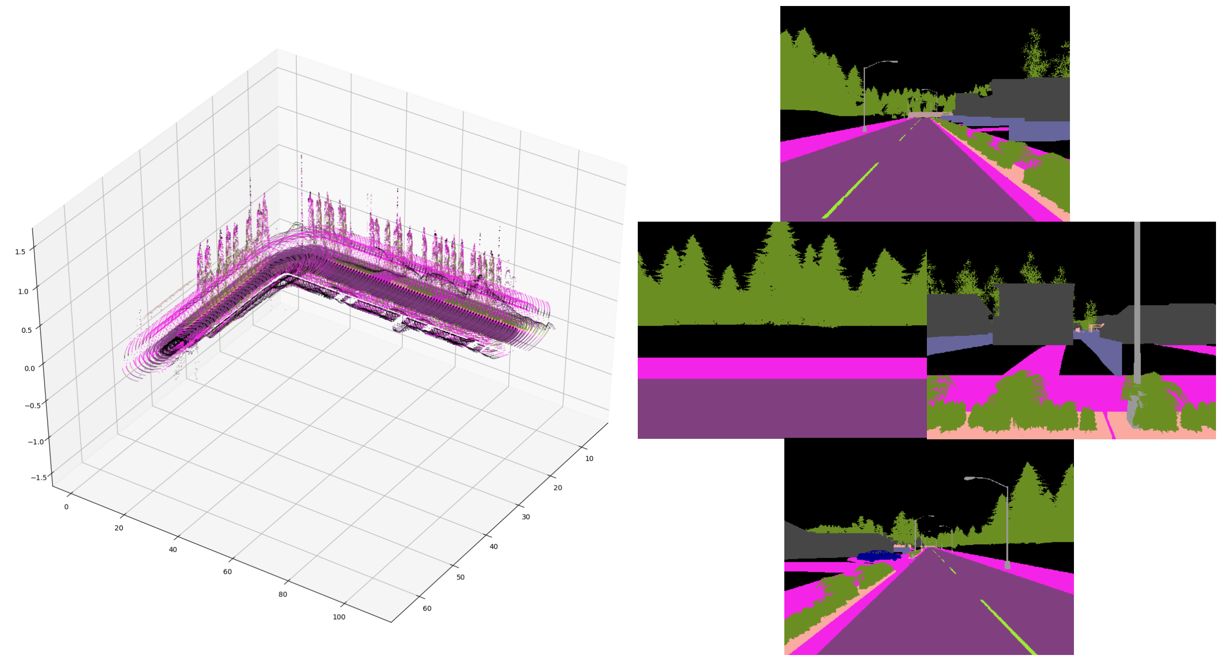


Figure 3.11. Example of 3D LiDAR points and semantic segmentation camera facing four directions. The LiDAR points are annotated with semantic class labels.

as staying on the road, and keeping in the current lane. The ground plane is discretized into a 256×256 grid of 0.5 meter resolution. Expert trajectories that do not fit in the given grid size are discarded. For planning purposes, the agent motion is modeled over a 4-connected grid with control space $\mathcal{U} := \{up, down, left, right\}$. A planned sequence of such controls is followed using the CARLA PID controller. Simulation features not related to the experiment are disabled, including spawning other vehicles and pedestrians, changing traffic signals and weather conditions, etc. Designing an agent that understands more complicated environment settings with other moving objects and changing traffic lights will be considered in future research.

Models and Metrics

`DeepMaxEnt`: We use the `DeepMaxEnt` IRL algorithm [174] with a multi-scale FCN cost encoder as a baseline again. Unlike the previous 2D experiment in Sec. 3.1.3, we use the input format from the original paper. Specifically, observed 3D point clouds are mapped into a 2D grid with three channels: the mean and variance of the height of the points as well as the cell visibility of each cell. This model does not utilize the point cloud semantic labels.

Table 3.3. Test results from the CARLA *Town05* environment, including the negative log-likelihood (*NLL*) and prediction accuracy (*Acc.*) of the validation set expert controls and the trajectory success rate (*TSR*) and modified Hausdorff distance (*MHD*) between the agent and the expert trajectories on the test set.

Model	*NLL*	*Acc* (%)	*TSR* (%)	*MHD*
`DeepMaxEnt`	0.673	85.3	89	4.331
`DeepMaxEnt + Semantics`	0.742	82.6	87	4.752
`Ours`	0.406	94.2	93	2.538

`DeepMaxEnt + Semantics`: The input features are augmented with additional channels that contain the number of points in a cell of each particular semantic class. This model uses the additional semantic information but does not explicitly map the environment over time.

`Ours`: We ignore the height information in the 3D point clouds $\mathbf{P}_{1:t}$ and maintain a 2D semantic map. The cost encoder is a two scale convolution and down(up)-sampling neural network, described in Sec. 3.1.2. Additionally, our model is implemented using sparse tensors, described in Sec. 3.1.2, to take advantage of the sparsity in the map $\mathbf{h}_t$ and cost c_t. The models are implemented using the Minkowski Engine [32] and the PyTorch library [129] and are trained with the Adam optimizer [91]. The neural network training and the online inference during testing are performed on an Intel i7-7700K CPU and an NVIDIA GeForce GTX 1080Ti GPU.

Metrics: The metrics, *NLL*, *Acc*, *TSR*, and *MHD*, introduced in Sec. 3.1.3, are used for evaluation.

CARLA Results

Table. 3.3 shows the performance of our model in comparison to `DeepMaxEnt` and `DeepMaxEnt + Semantics`. Our model learns to generate policies closest to the expert demonstrations in the validation map *Town*04 by scoring best in *NLL* and *Acc* metrics. During testing in map *Town*05, the models predict controls at each step online to generate the agent trajectory. `Ours` achieves the highest success rate of reaching the goal without hitting sidewalks and other obstacles. Among the successful trajectories, `Ours` is also closest to the expert by achieving the minimum *MHD*. The results demonstrate that the map encoder captures both geometric and

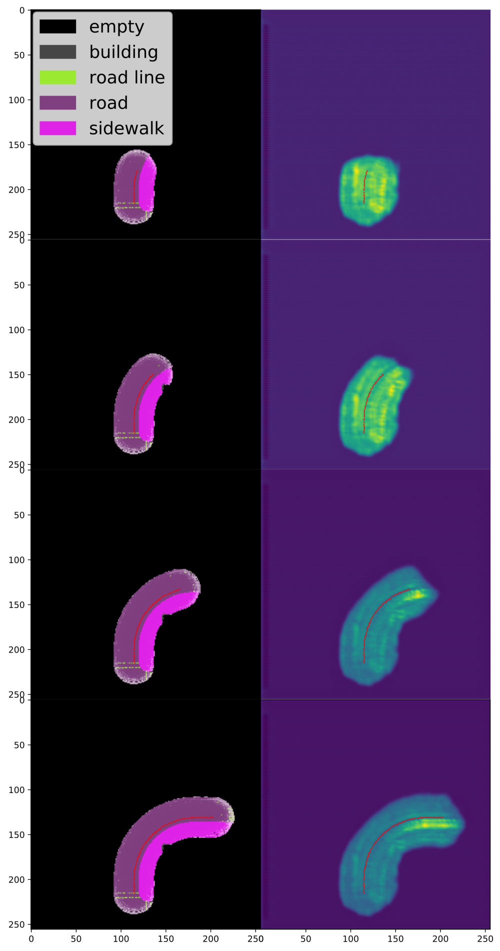


Figure 3.12. Examples of semantic occupancy estimation and cost encoding during different steps in a test trajectory marked in red. The left column shows the most probable semantic class of the map encoder and the right column shows the cost to arrive at each state. Our model correctly distinguishes the road from other categories (e.g., sidewalk, building, etc) and assigns lower cost to road than sidewalks.

semantic information, allowing accurate cost estimation and generation of trajectories that match the expert behavior. Fig. 3.12 shows an example of a generated trajectory during testing in the previously unseen *Town*05 environment (also see Extension 1). The map encoder predicts correct semantic class labels for each cell and the cost encoder assigns higher costs to sidewalks than the road. We notice that the addition of semantic information actually degrades the performance of `DeepMaxEnt`. We conjecture that the increase in the number of input channels, due to the addition of the number of LiDAR points per category, makes the convolutional neural network layers prone to overfit on the training set but generalize poorly on the validation and test sets.

Table 3.4. Runtime analysis of our model during testing in the CARLA simulator. We report per-step runtime averaged over 100 test trajectories.

Simulation	Data preprocessing	Model inference
39.3 ± 1.5 ms	14.9 ± 3.2 ms	6.4 ± 2.5 ms

Additional examples of agent trajectories and semantic mapping predictions are given in Online Resource 1. We also report runtime analysis for test-time model inference in Table 3.4. Each time step is divided into (1) simulator update, where the agent is set at new states and image and lidar observations are generated, (2) data preprocessing, where semantic labels are retrieved for point clouds and data are moved to GPU, and (3) model inference.

Evaluation with Dynamic Obstacles

In this section, we study the effects of dynamic obstacles in the scene on our model's map and cost encoders. We create three different scenarios where the agent vehicle has to overtake a lower speed non-player character (NPC vehicle). In Scenario 1, the NPC is spawned in the left lane to the agent and 20 meters ahead. The agent is expected to stay in its own lane when overtaking the NPC. In Scenario 2, the NPC is spawned in the same lane as the agent and 20 meters ahead. The agent has to move to its left lane to overtake the NPC. Scenario 3 is a mixture of the first two where the NPC could be either in the same lane or in the left lane to the agent. A visualization of the first two scenarios is shown in Fig. 3.13. Training and evaluation are conducted in the $Town05$ map since it contains multi-lane streets while other maps contain mostly single-lane streets. We sample 100 trajectories for testing within the top-left quadrant of the map and 200 trajectories for training from other quadrants as illustrated in Fig. 3.14. We train our model in all three scenarios and test it in the same scenario where it is trained. Each trajectory is discretized on a 128×128 grid of 1 meter resolution.

To effectively capture the most current information of the dynamic NPC vehicle, we

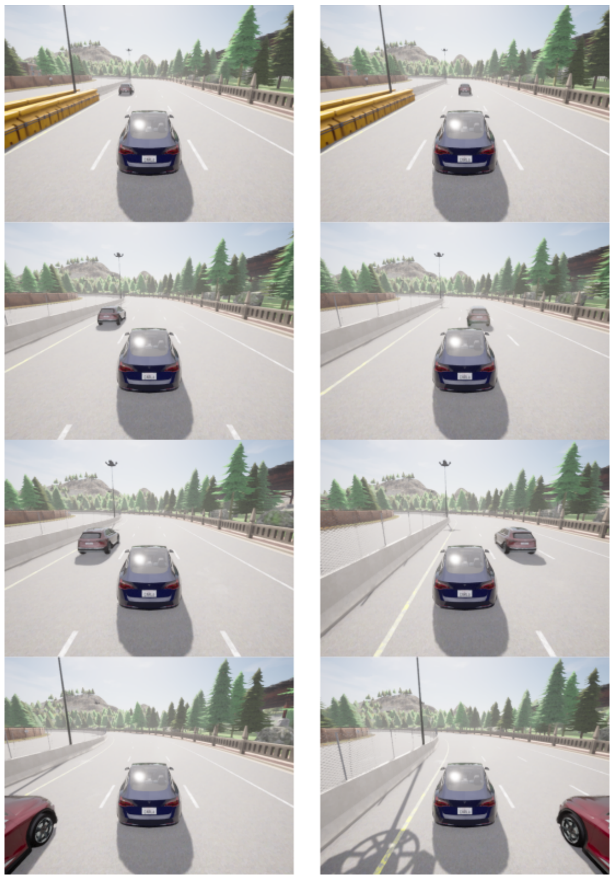


Figure 3.13. Two scenarios with dynamic obstacles. Left column (scenario 1): the agent vehicle (blue) keeps in its own lane when overtaking the NPC vehicle (red) in the left lane. Right column (scenario 2): when the NPC and agent vehicles are spawned in the same lane, the agent switches to the left lane to overtake.

multiply the grid log-odds $\mathbf{h}_{t,j}$ with a decay rate $\gamma \in \{1.0, 0.9, 0.8, 0.7\}$, i.e.,

$$h_{t,j}^k = \gamma h_{t-1,j}^k + \log \frac{p(\mathbf{P}_t \mid m^j = k, \mathbf{x}_t)}{p(\mathbf{P}_t \mid m^j = 0, \mathbf{x}_t)}. \tag{3.21}$$

The map encoder is the same as in previous experiments when $\gamma = 1.0$, while when $\gamma < 1$ past observation information is slowly removed. Note that we use the same decay rate across all semantic classes since we do not assume prior knowledge of which classes are dynamic or static. Alternatively, it is possible to use a different decay rate for each class, $h_{t-1,j}^k \rightarrow \gamma^k h_{t-1,j}^k$, or set γ^k as a learnable parameter to be optimized with the overall objective in (3.5). The semantic probabilities and cost encoder output of the same trajectory with different decay rates is shown

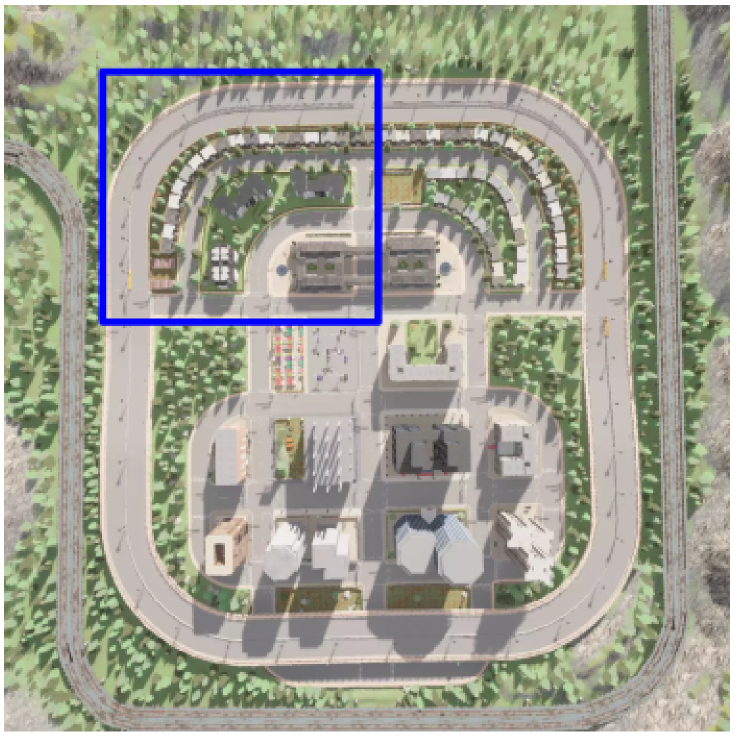


Figure 3.14. Bird's-eye view of the *Town*05 map. The top-left quadrant is resevered for testing while training trajectories are sampled from other regions.

in Fig. 3.15.

We report the results of our model with different decay rates in each scenario in Table 3.5. In addition to the *TSR* and *MHD* metrics, we report the collision rate (*CR*) between the agent and the NPC vehicles in the test trajectories. We find that *CR* is higher in Scenario 2 than in Scenario 1, which is expected as lane changing is a harder task when a moving NPC vehicle is present. The performance in the mixed scenario is on par with that in Scenario 2, suggesting that our policy class in (3.4) may not capture a multi-modal distribution in the demonstrated behaviors effectively. Within each scenario, we find that the model generally works better when the decay rate γ is close to 1.0. We suspect that since both vehicles are moving forward in the same direction, it does not hurt to map the NPC's past locations. However, when γ is small, forgetting the NPC's history makes its semantic probability smaller (as shown in Fig. 3.15), and thus the agent has a higher chance of colliding into the NPC vehicle. Finally, we find that MHD is consistent across all settings which shows that the agent trajectories are close to the expert's,

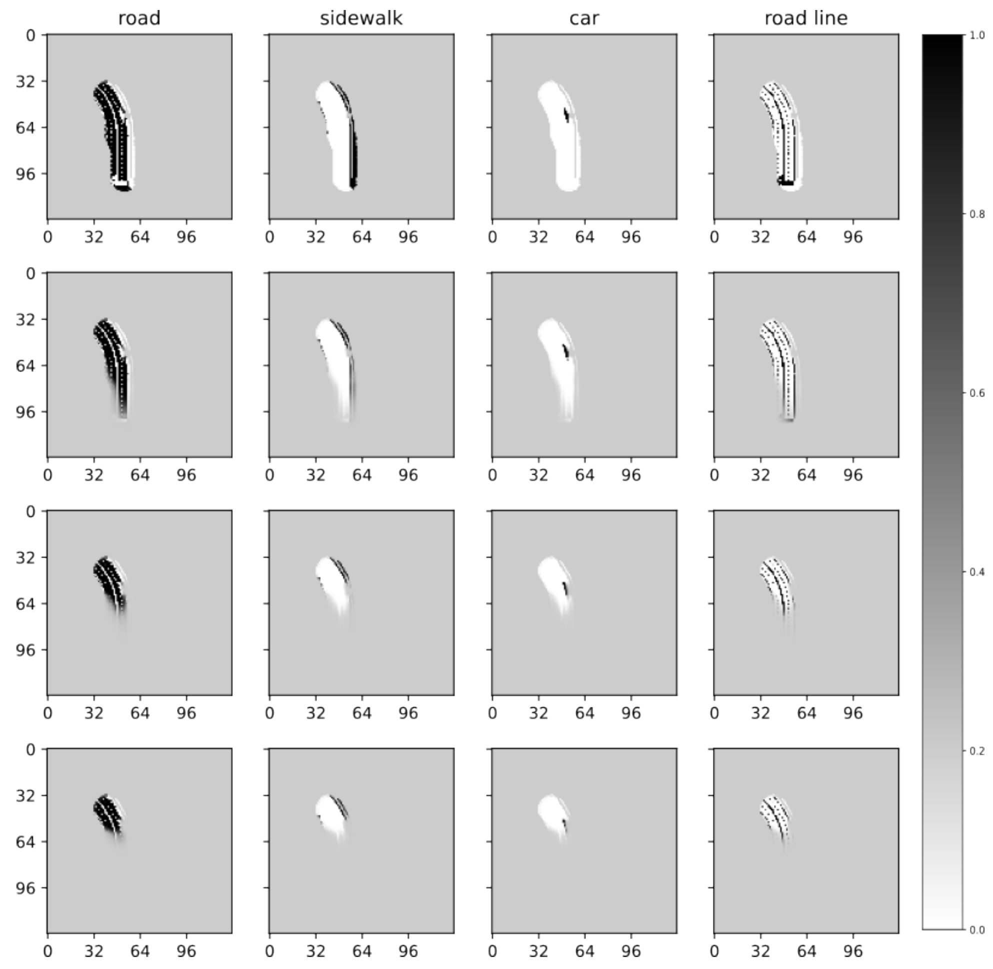


Figure 3.15. Semantic probability of each class with different decay rate $\gamma \in \{1, 0.9, 0.8, 0.7\}$. when they are successful.

Evaluation with Noisy Semantic Observations

In this section, we study how noisy semantic observations can affect downstream cost prediction and policy inference. First, we consider noise added to the contours of each segmentation region. We replace each pixel value in the original 600×800 semantic segmentation image with a random pixel within its local 5×5 pixel window. This makes the segment boundaries blurry while the interior of each semantic region is unchanged (see Fig. 3.16 (b)). With this noise model, only 0.2% of the lidar points are labeled incorrectly. We considered two additional noise models in which 2% and 20% of all pixels are randonmly changed to an incorrect label chosen among the remaining semantic labels (see Fig. 3.16 (c) and (d)).

Table 3.5. Test results with dynamic obstacles from CARLA *Town05* map, including trajectory success rate (*TSR*), collision rate (*CR*), and modified Hausdorff distance (*MHD*) between the agent and the expert trajectories on the test set.

	Scenario 1 (no lane change)			Scenario 2 (lane change required)			Scenario 3 (mixed scenario)		
Decay rate γ	*TSR* (%)	*CR* (%)	*MHD*	*TSR* (%)	*CR* (%)	*MHD*	*TSR* (%)	*CR* (%)	*MHD*
1.0	92	2	2.878	84	12	2.528	84	8	2.708
0.9	92	3	2.795	88	11	2.446	84	9	2.690
0.8	90	2	2.721	78	15	2.512	80	11	2.912
0.7	86	7	3.224	73	24	2.885	78	16	2.948

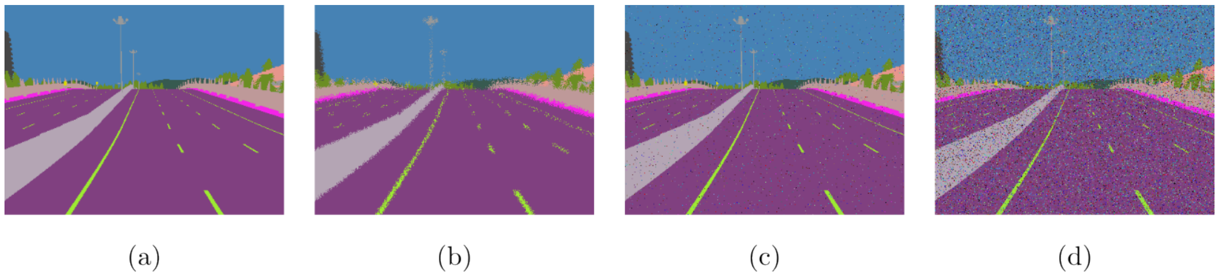


Figure 3.16. Noisy semantic segmentation observations: (a) original image, (b) each pixel is replaced with a random pixel within its local 5×5 pixel window (c) 2% of all pixels are randomly changed, (d) 20% of all pixels are randomly changed.

We find that these noise models have very little influence on the policy inference. To understand this, we study how much the map encoder output changes when using noisy semantic images. We calculate the total variation distance between the semantic map probabilities obtained from the original and perturbed semantic images. Specifically, let $\mathbb{P}_a := \mathbb{P}(m^j = k \mid \mathbf{x}_{1:T}, \mathbf{P}_{1:T})$ be the semantic posterior probability of a trajectory using the original semantic segmentation images and, correspondingly, let $\mathbb{P}_b$, $\mathbb{P}_c$, $\mathbb{P}_d$ denote the posteriors using perturbed images from Fig. 3.16. The total variation distance between two discrete probability measures is

$$TV(\mathbb{P}_a, \mathbb{P}_b) := \frac{1}{2} \sum_k |\mathbb{P}_a(m^j = k) - \mathbb{P}_b(m^j = k)| \tag{3.22}$$

Table 3.6 and Fig. 3.17 show the maximum and a histogram, respectively, of the total variation across all grid cells. These results show that our map encoder is robust to noise even when 20%

Table 3.6. Maximum total variation distance between the original and perturbed semantic probabilities across all grid cells.

$\max_{m_j} TV(\mathbb{P}_a, \mathbb{P}_b)$	$\max_{m_j} TV(\mathbb{P}_a, \mathbb{P}_c)$	$\max_{m_j} TV(\mathbb{P}_a, \mathbb{P}_d)$
0.002	0.185	0.511

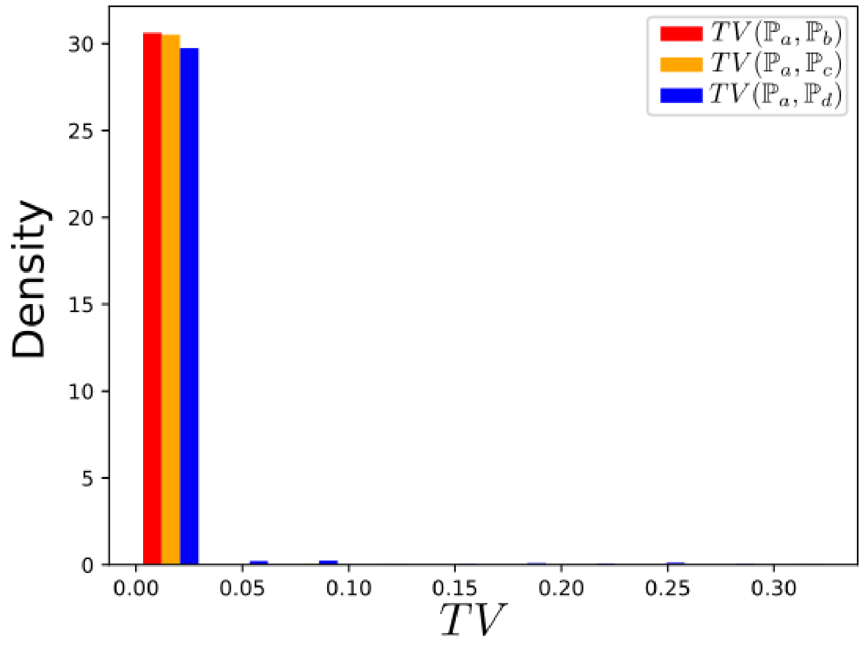


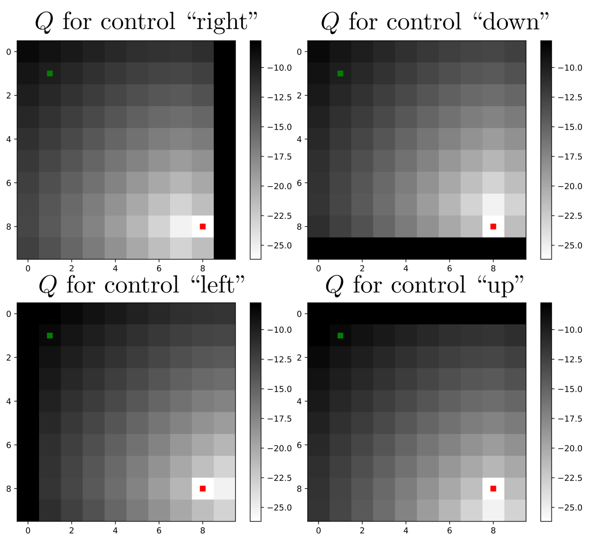
Figure 3.17. Histogram of the total variation distance between the semantic probabilities from the original and the perturbed semantic images. Even with 20% incorrectly labeled pixels in the semantic segmentation images, most semantic probabilities are unaffected.

of the labels in the semantic images are wrong.

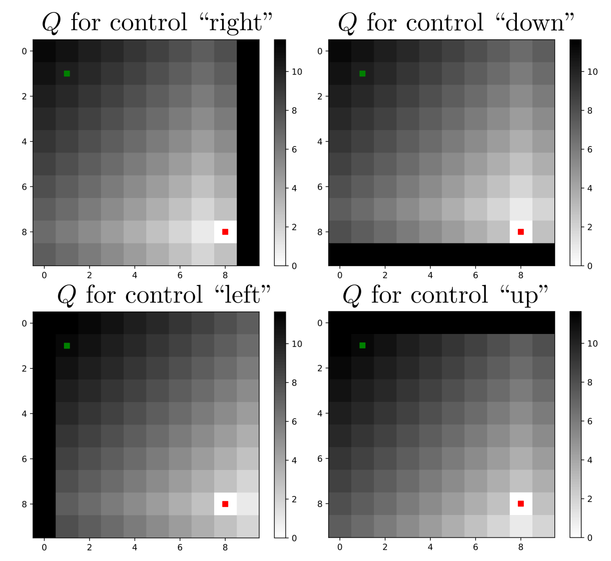
3.1.4 Derivations

For completeness, we present a comparison and derivation between our model and the MaxEnt formulation.

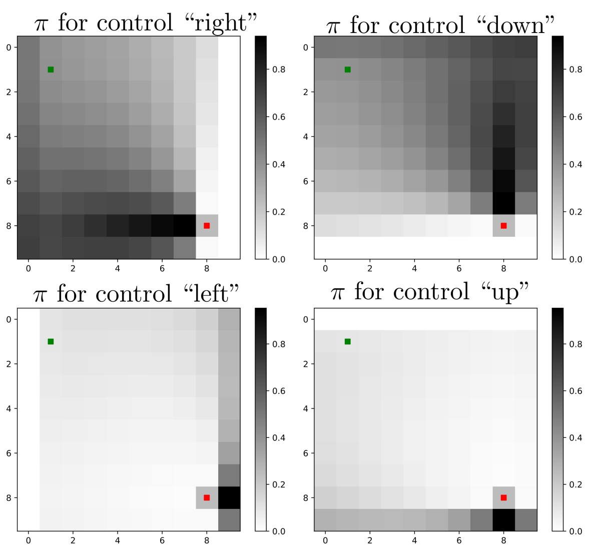
This appendix compares the MaxEnt expert model of [192] to the expert model proposed in Sec. 3.1.1. The MaxEnt model has been widely studied in the context of reinforcement learning and inverse reinforcement learning [69, 49, 104]. On the other hand, while a Boltzmann policy is a well-known method for exploration in reinforcement learning, it has not been used to model expert or learner behavior in inverse reinforcement learning.



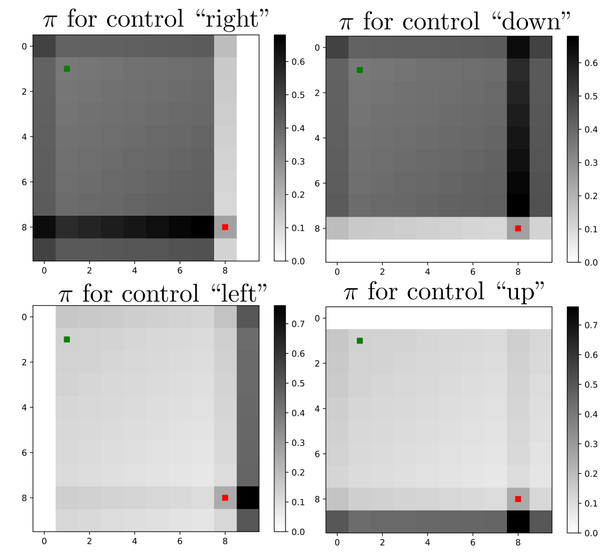
(a) Value function Q_{ME} for MaxEnt policy



(b) Value function Q_{BM} for Boltzmann policy



(c) Maxent policy π_{ME}



(d) Boltzmann policy π_{BM}

Figure 3.18. Value functions corresponding to the MaxEnt and Boltzmann policies in infinite horizon setting with discount $\gamma = 0.95$. The environment only has obstacles around the outer boundary. The start state is marked in green and the goal in red. The controls are $\{right, down, left, up\}$ at each state with constant true cost of 0 to arrive at the goal (which is an absorbing state), 1 to any state except the goal in the grid and infinity to any obstacle outside the border. Darker color indicates higher cost-to-go values to reach the goal (top two rows) or higher probability of choosing a control (bottom two rows). Although the absolute values of Q_{ME} and Q_{BM} are different, both have similar relative value differences across the controls, providing well-performing policies π_{ME} and π_{BM}.

The work of [69] shows that both a Boltzmann policy and the MaxEnt policy are special cases of an energy-based policy:

$$\pi(\mathbf{u}_t \mid \mathbf{x}_t) \propto \exp(-E(\mathbf{x}_t, \mathbf{u}_t)) \tag{3.23}$$

with appropriate choices of the energy function E. We study the two policies in the discounted infinite-horizon setting, as this is the most widely used setting for the MaxEnt model. Extensions to first-exit and finite-horizon formulations are possible. Consider a Markov decision process with finite state space $\mathcal{X}$, finite control space $\mathcal{U}$, transition model $p(\mathbf{x}' \mid \mathbf{x}, \mathbf{u})$, stage cost $c(\mathbf{x}, \mathbf{u})$, and discount factor $\gamma \in (0,1)$.

Proposition 3 ([69, Thm. 1])**.** *Define the maximum entropy Q-value as:*

$$Q_{ME}(\mathbf{x}_t, \mathbf{u}_t) := c(\mathbf{x}_t, \mathbf{u}_t) + \min_{\pi} \mathbb{E}_{\pi,p}\left[\sum_{k=t+1}^{\infty} \gamma^{k-t}\left(c(\mathbf{x}_k, \mathbf{u}_k) - \alpha \mathcal{H}(\pi(\cdot \mid \mathbf{x}_k))\right)\right], \tag{3.24}$$

where $\mathcal{H}(\pi(\cdot \mid \mathbf{x})) = -\sum_{\mathbf{u}\in\mathcal{U}} \pi(\mathbf{u} \mid \mathbf{x}) \log \pi(\mathbf{u} \mid \mathbf{x})$ is the Shannon entropy of $\pi(\cdot \mid \mathbf{x})$. Then, the maximum entropy policy satisfies:

$$\pi_{ME}(\mathbf{u}_t \mid \mathbf{x}_t) \propto \exp\left(-\frac{1}{\alpha} Q_{ME}(\mathbf{x}_t, \mathbf{u}_t)\right). \tag{3.25}$$

Similarly, define the usual Q-value as:

$$Q_{BM}(\mathbf{x}_t, \mathbf{u}_t) := c(\mathbf{x}_t, \mathbf{u}_t) + \min_{\pi} \mathbb{E}_{\pi,p}\left[\sum_{k=t+1}^{\infty} \gamma^{k-t} c(\mathbf{x}_k, \mathbf{u}_k)\right] \tag{3.26}$$

and the Boltzmann policy associated with it as:

$$\pi_{BM}(\mathbf{u}_t \mid \mathbf{x}_t) \propto \exp\left(-\frac{1}{\alpha} Q_{BM}(\mathbf{x}_t, \mathbf{u}_t)\right). \tag{3.27}$$

The value functions Q_{ME} and Q_{BM} can be seen as the fixed points of the following

Bellman contraction operators:

$$\mathcal{T}_{ME}[Q](\mathbf{x}_t, \mathbf{u}_t) := c(\mathbf{x}_t, \mathbf{u}_t) - \gamma\alpha\mathbb{E}_p \left[\log \sum_{\mathbf{u}_{t+1} \in \mathcal{U}} \exp\left(-\frac{1}{\alpha} Q(\mathbf{x}_{t+1}, \mathbf{u}_{t+1})\right)\right] \tag{3.28}$$

$$\mathcal{T}_{BM}[Q](\mathbf{x}_t, \mathbf{u}_t) := c(\mathbf{x}_t, \mathbf{u}_t) + \gamma\mathbb{E}_p \left[\min_{\mathbf{u}_{t+1} \in \mathcal{U}} Q(\mathbf{x}_{t+1}, \mathbf{u}_{t+1})\right]. \tag{3.29}$$

In the latter, the Q values are bootstrapped with a "hard" min operator, while in the former they are bootstrapped with a "soft" min operator given by the log-sum-exponential operation. The form of the Bellman equations resembles the online SARSA update and offline Q-learning update in reinforcement learning. Consider temporal difference control with transitions $(\mathbf{x}, \mathbf{u}, c, \mathbf{x}', \mathbf{u}')$ using SARSA backups:

$$Q(\mathbf{x}, \mathbf{u}) \leftarrow Q(\mathbf{x}, \mathbf{u}) + \eta\,[c(\mathbf{x}, \mathbf{u}) + \gamma Q(\mathbf{x}', \mathbf{u}') - Q(\mathbf{x}, \mathbf{u})]$$

and Q-learning backups:

$$Q(\mathbf{x}, \mathbf{u}) \leftarrow Q(\mathbf{x}, \mathbf{u}) + \eta\,[c(\mathbf{x}, \mathbf{u}) + \gamma \min_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}') - Q(\mathbf{x}, \mathbf{u})]$$

where η is a step-size parameter. If we additionally assume that the controls are sampled from the energy-based policy in (3.23) defined by Q, the SARSA algorithm specifies the MaxEnt policy, while the Q-learning algorithm specifies the Boltzmann policy.

We show a visualization of the MaxEnt and Boltzmann policies, π_{ME}, π_{BM}, as well as their corresponding value functions Q_{ME}, Q_{BM}, in the infinite horizon setting with discount $\gamma = 0.95$ and $\alpha = 1$. The 4-connected grid environment in Fig. 3.18 has obstacles only along the outside border. The true cost is 0 to arrive at the goal (which is an absorbing state), 1 to any state (except the goal) inside the grid, infinity to any obstacle outside the border. Note that Q_{ME} and Q_{BM} are very different in absolute value. In fact, Q_{ME} is negative for all states due to the additional entropy term in (3.24). However, the relative value differences across the controls are similar and, thus, both policies π_{ME} and π_{BM} generate desirable paths from start to goal.

3.2 Inferring Logic from Demonstrations

3.2.1 Problem Formulation

Environment and Agent Models

The agent's interaction with the environment is modeled as an L-MDP [39].

Definition 4. A labeled Markov decision process is a tuple $\{\mathcal{X}, \mathcal{U}, \mathbf{x}_0, f, c, \mathcal{AP}, \ell\}$, where $\mathcal{X}, \mathcal{U}$ are finite sets of states and controls, $\mathbf{x}_0 \in \mathcal{X}$ is an initial state, $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is a deterministic transition function, and $c : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ assigns a non-negative cost when control $\mathbf{u} \in \mathcal{U}$ is applied at state $\mathbf{x} \in \mathcal{X}$. A finite set of atomic propositions $\mathcal{AP}$ provides logic statements that must be true or false (e.g., “the agent is 1 meter away from the closest obstacle” or “the agent possesses a key”). A labeling function $\ell : \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{AP}}$ assigns a set of atomic propositions that evaluate true for a given state transition.

We assume that the state $\mathbf{x}$ is fully observable and captures both endogenous variables for the agent, such as position and orientation, and exogenous variables, such as an environment containing objects of interest as illustrated in Fig. 3.2. The transition function $f(\mathbf{x}, \mathbf{u})$ specifies the change of state $\mathbf{x}$ when control $\mathbf{u}$ is executed, and $c(\mathbf{x}, \mathbf{u})$ assigns a non-negative cost to this transition. The alphabet of the L-MDP is the set of labels $\Sigma = 2^{\mathcal{AP}}$ that can be assigned to the transitions. The labeling function $\sigma = \ell(\mathbf{x}, \mathbf{u})$ provides the atomic propositions $\sigma \in \Sigma$ which are satisfied during the transition $f(\mathbf{x}, \mathbf{u})$. The set of words on Σ is denoted by Σ^* and consists of all strings $\sigma_{0:T} = \sigma_0 \ldots \sigma_T$ for $\sigma_t \in \Sigma$ and $T \in \mathbb{N}$. We assume that the transition f and labeling ℓ are known. However, the cost function c is unknown and needs to be inferred from expert demonstrations.

Expert Model

The agent needs to execute a task, whose success is evaluated based on the word $\sigma_{0:T} \in \Sigma^*$ resulting from the agent's actions. We model the quality of the task execution by a function $h : \Sigma^* \to \mathbb{R}$. An execution $\sigma_{0:T}$ is deemed successful if $h(\sigma_{0:T}) \geq \xi$ for a known performance threshold ξ, and unsuccessful otherwise. As argued in the introduction, defining the function h explicitly is challenging in many applications. Instead, we consider a training set $\mathcal{D} = \left\{(\mathbf{x}^n_{0:T_n}, \mathbf{u}^n_{0:T_n}, s^n)\right\}_{n=1}^N$ of N demonstrations of the same task in different environment configurations provided by an expert. Each demonstration n contains the controls $\mathbf{u}^n_{0:T_n} = \mathbf{u}^n_0 \ldots \mathbf{u}^n_{T_n}$ executed by the expert, the resulting agent-environment states $\mathbf{x}^n_{0:T_n} = \mathbf{x}^n_0 \ldots \mathbf{x}^n_{T_n}$, and the success level $s^n \in \mathbb{R}$ of the execution, measured by $h(\sigma^n_{0:T_n})$, where $\sigma^n_t = \ell(\mathbf{x}^n_t, \mathbf{u}^n_t)$ is the label encountered by the expert at time t. We assume that the expert knows the *true* task h and the *true* cost c and can solve a finite-horizon first-exit deterministic optimal control problem [16] over the L-MDP:

$$\begin{aligned} Q^*(\mathbf{x}, \mathbf{u}) &:= \min_{T, \mathbf{u}_{1:T}} \sum_{t=0}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \\ &\text{s.t. } \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t),\ \mathbf{x}_0 = \mathbf{x},\ \mathbf{u}_0 = \mathbf{u}, \sigma_t = \ell(\mathbf{x}_t, \mathbf{u}_t),\ h(\sigma_{0:T}) \geq \xi, \end{aligned} \tag{3.30}$$

where $Q^*(\mathbf{x}, \mathbf{u})$ is the optimal value function. Since (3.30) is a deterministic optimal control problem, there exists an open-loop control sequence which is optimal, i.e., achieves the same cost as an optimal closed-loop policy function [16, Chapter 6]. However, we consider experts that do not necessarily choose strictly rational controls. Instead, we model the expert behavior using a stochastic Boltzmann policy over the optimal values $\pi^*(\mathbf{u}|\mathbf{x}) \propto \exp\left(-\frac{1}{\eta} Q^*(\mathbf{x}, \mathbf{u})\right)$, where $\eta \in (0, \infty)$ is a temperature parameter representing a continuous spectrum of rationality. For example, $\eta \to 0$ means that the expert takes strictly optimal controls while $\eta \to \infty$ means random controls are selected. The Boltzmann expert model was previously introduced and studied in [120, 138, 171]. It provides an exponential preference for controls that incur low long-term costs. This expert model also allows efficient policy search, as shown in Sec. 3.2.2, and computation of the policy gradient with respect to the cost needed to optimize the cost parameters, as shown in

Sec. 3.2.2.

The agent needs to infer the unknown task model h and unknown cost function c from the expert demonstrations $\mathcal{D} = \left\{ (\mathbf{x}^n_{0:T_n}, \mathbf{u}^n_{0:T_n}, s^n) \right\}_{n=1}^{N}$.

Problem 2. Given the demonstrations $\mathcal{D}$ and labeling $\sigma^n_t = \ell(\mathbf{x}^n_t, \mathbf{u}^n_t)$, optimize the parameters ψ of an approximation h_ψ of the unknown task function h to minimize the mean squared error:

$$\min_{\psi} \mathcal{L}_h(\psi) := \frac{1}{N} \sum_{n=1}^{N} \left(h_\psi(\sigma^n_{0:T_n}) - s^n \right)^2. \tag{3.31}$$

Similarly, the agent needs to obtain an approximation c_θ with parameters θ of the unknown cost function c. This allows the agent to obtain a control policy:

$$\pi_\theta(\mathbf{u}|\mathbf{x}) \propto \exp\left(-\frac{1}{\eta} Q_\theta(\mathbf{x}, \mathbf{u}) \right), \tag{3.32}$$

approximating the expert model using a value function Q_θ computed according to (3.30) with c and h replaced by c_θ and h_ψ, respectively.

Problem 3. Given the demonstrations $\mathcal{D}$, optimize the parameters θ of an approximation c_θ of the unknown cost function c such that the log-likelihood of the demonstrated controls $\mathbf{u}^n_t$ is maximized under the agent policy in (3.32):

$$\min_{\theta} \mathcal{L}_c(\theta) := -\sum_{n=1}^{N} \mathbb{1}_{\{s^n \geq \xi\}} \sum_{t=0}^{T_n} \log \pi_\theta(\mathbf{u}^n_t | \mathbf{x}^n_t), \tag{3.33}$$

where $\mathbb{1}$ is an indicator function and ξ is the known task satisfaction threshold.

3.2.2 Learning Task Logic as Weighted Finite Automata

We first discuss how to learn a task model h_ψ from demonstrations $\mathcal{D}$ in Sec. 3.2.2. Next, in Sec. 3.2.2, we learn a cost model c_θ by solving the optimal control problem in (3.30) to obtain an agent policy π_θ. Finally, in Sec. 3.2.2, we show how to backpropagate the policy loss $\mathcal{L}_c(\theta)$ in (3.33) through the optimal control problem to update the cost parameters θ.

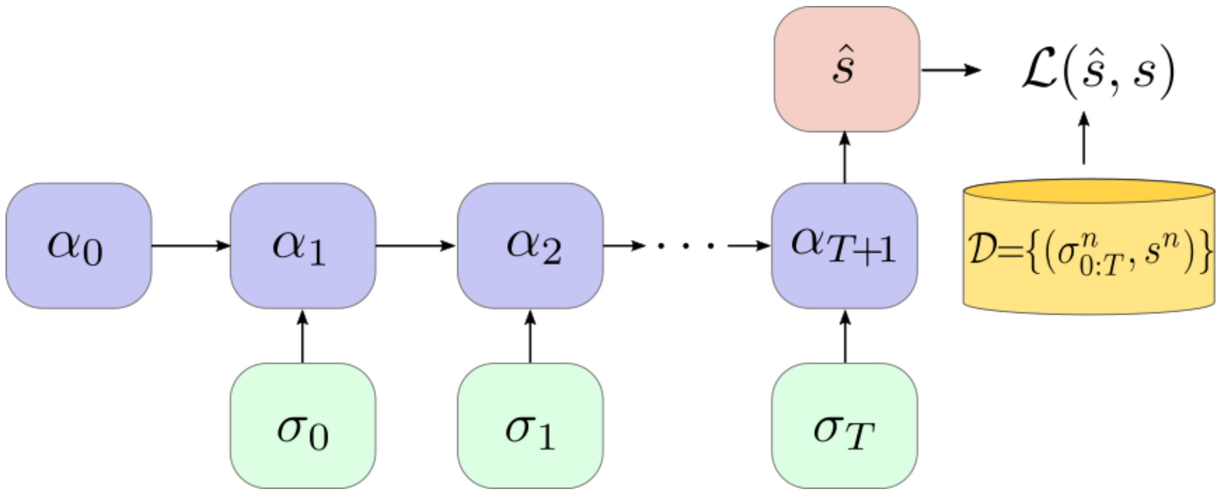


Figure 3.19. Inferring the hidden state progression $\boldsymbol{\alpha}_t$ from events σ_t can be acheived by an RNN with initial hidden state $\boldsymbol{\alpha}_0$, hidden state transition $\boldsymbol{\alpha}_{t+1} = g_1(\sigma_t, \boldsymbol{\alpha}_t, \mathbf{W})$ and output $\hat{s} = h_\psi(\sigma_{0:T}) = g_2(\boldsymbol{\alpha}_{T+1}, \boldsymbol{\beta})$, where g_1, g_2 are nonlinear functions. The weights $\psi = (\boldsymbol{\alpha}_0, \mathbf{W}, \boldsymbol{\beta})$ can be learned via the loss $\mathcal{L}(\hat{s}, s)$ in (3.31) between RNN outputs $\hat{s}$ and demonstration scores s.

Spectral Learning of Task Specifications

Fitting a single cost neural network c_θ that is capable of generalizing to various environment configurations and tasks is difficult when state and control spaces are large and the task horizon is long. An alternative is to consider the cost function and its corresponding policy only for small segments of the task, associated with different subtasks. This idea is based on the observation that task specifications often have a compositional logic structure. For example, the demonstrated trajectory in the DoorKey environment in Fig. 3.2 can be decomposed into three segments, each denoted by a high-level state $\boldsymbol{\alpha}$. The transitions between the high-level states are triggered by events like σ_0: a key is picked up, and σ_1: a door is opened. Note that there is no direct transition between $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_3$ because the door cannot be opened without possessing a key. Such high-level state abstraction and transitions are commonly learned via recurrent neural network (RNN) or memory architectures [74, 116]. For example, to solve Problem 2, we can use an RNN h_ψ in Fig. 3.19 with initial hidden state $\boldsymbol{\alpha}_0$, hidden state transition $\boldsymbol{\alpha}_{t+1} = g_1(\sigma_t, \boldsymbol{\alpha}_t, \mathbf{W})$ and output function $h_\psi(\sigma_{0:T}) = g_2(\boldsymbol{\alpha}_{T+1}, \boldsymbol{\beta})$, where g_1, g_2 are nonlinear functions and $\psi = (\boldsymbol{\alpha}_0, \mathbf{W}, \boldsymbol{\beta})$ are learnable weights. Instead of an RNN model, in this work, we propose to use a weighted finite automaton (WFA) [13] to represent h_ψ. A WFA is less expressive than an RNN [135] but can be trained more effectively from small demonstration dataset. Moreover, a WFA generalizes deterministic and nondeterministic finite automata, which are commonly used to model logic task specifications for autonomous agents [98, 44, 97, 45]. Hence, a WFA model is sufficiently expressive to represent a complex task and allows one to focus on a temporal abstraction without reliance on the low-level system dynamics.

Definition 5. A weighted finite automaton (WFA) with m states is a tuple $\psi = \left\{\boldsymbol{\alpha}_0, \boldsymbol{\beta}, \{\mathbf{W}_\sigma\}_{\sigma\in\Sigma}\right\}$ where $\boldsymbol{\alpha}_0, \boldsymbol{\beta} \in \mathbb{R}^m$ are initial and final weight vectors and $\mathbf{W}_\sigma \in \mathbb{R}^{m\times m}$ are transition matrices associated with each symbol $\sigma \in \Sigma$. A WFA ψ represents a function $h_\psi : \Sigma^* \to \mathbb{R}$ by $h_\psi(\sigma_{0:T}) = \boldsymbol{\alpha}_0^\top \mathbf{W}_{\sigma_0}\mathbf{W}_{\sigma_1}\ldots\mathbf{W}_{\sigma_T}\boldsymbol{\beta}$.

A WFA represents the task progress for a given word $\sigma_{0:t}$ via $h_\psi(\sigma_{0:t}) = \boldsymbol{\alpha}_0^\top \mathbf{W}_{\sigma_0}\mathbf{W}_{\sigma_1}\ldots \mathbf{W}_{\sigma_t}\boldsymbol{\beta}$, where the high-level state at time $t+1$ is $\boldsymbol{\alpha}_{t+1} = \left(\boldsymbol{\alpha}_0^\top \mathbf{W}_{\sigma_0}\mathbf{W}_{\sigma_1}\ldots\mathbf{W}_{\sigma_t}\right)^\top$. When the WFA is learned correctly, its prediction for an expert word should approximate the expert score s, i.e., $h_\psi(\sigma_{0:T}) \approx s$. This can be used to guide a task planning algorithm by providing a task satisfaction criterion. A trajectory with corresponding word $\sigma_{0:T}$ is identified as successful if the WFA prediction passes the known performance threshold introduced in Sec. 3.2.1, i.e., $h_\psi(\sigma_{0:T}) = \boldsymbol{\alpha}_{T+1}^\top\boldsymbol{\beta} \geq \xi$.

Our approach to learn a minimal WFA is based on the spectral learning method developed by [13]. The spectral method makes use of a Hankel matrix $\mathbf{H}_h \in \mathbb{R}^{\Sigma^*\times\Sigma^*}$ associated with the function $h : \Sigma^* \to \mathbb{R}$, which is a bi-infinite matrix with entries $\mathbf{H}_h(u,v) = h(uv)$ for $u, v \in \Sigma^*$. We assume the class of functions h that can be represented by a WFA are rational power series functions and their associated Hankel matrix $\mathbf{H}_h$ has finite rank [15, 143]. It can be showns that under certain assumptions WFA are expressively equivalent to monadic second-order logic. The quantitative property of WFA allows us to model the performance score s of the demonstrated trajectories.

Assumption 1. The Hankel matrix $\mathbf{H}_h$ associated with the true task specification h has finite rank.

In practice, only finite sub-blocks of the Hankel matrix, constructed from the expert demonstrations $\mathcal{D}$, can be considered. Given a basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ where $\mathcal{P}, \mathcal{S} \subset \Sigma^*$ are finite sets of prefixes and suffixes respectively, define $\mathbf{H}_{\mathcal{B}}$ and $\{\mathbf{H}_\sigma\}_{\sigma \in \Sigma}$ as the finite sub-blocks of $\mathbf{H}_h$ such that $\mathbf{H}_{\mathcal{B}}(u,v) = h(uv)$, $\mathbf{H}_\sigma(u,v) = h(u\sigma v)$, $\forall u \in \mathcal{P}, v \in \mathcal{S}$. The foundation of the spectral learning method is summarized in the following theorem.

Theorem 1 ([13]). *Given a basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ such that the empty string $\lambda \in \mathcal{P} \cap \mathcal{S}$ and $rank(\mathbf{H}_h) = rank(\mathbf{H}_{\mathcal{B}})$, for any rank m factorization $\mathbf{H}_{\mathcal{B}} = \mathbf{PS}$ where $\mathbf{P} \in \mathbb{R}^{|\mathcal{P}| \times m}$ and $\mathbf{S} \in \mathbb{R}^{m \times |\mathcal{S}|}$, the WFA $\{\boldsymbol{\alpha}_0, \boldsymbol{\beta}, \{\mathbf{W}_\sigma\}\}$ is a minimal WFA representing h, where $\boldsymbol{\alpha}_0^\top = \mathbf{P}(\lambda, :)$ is the row vector of $\mathbf{P}$ corresponding to prefix λ, $\boldsymbol{\beta} = \mathbf{S}(:, \lambda)$ is the column vector of $\mathbf{S}$ corresponding to suffix λ, and $\mathbf{W}_\sigma = \mathbf{P}^\dagger \mathbf{H}_\sigma \mathbf{S}^\dagger$, $\forall \sigma \in \Sigma$.*

A basis can be chosen empirically from demonstrations $\mathcal{D}$. For example, we can choose a basis that includes all prefixes and suffixes that appear in the words $\left\{\sigma_{0:T_n}^n\right\}$ or one with desired cardinality for the most frequent prefixes and suffixes. Given a basis, the Hankel blocks $\mathbf{H}_{\mathcal{B}}$, $\{\mathbf{H}_\sigma\}$ are constructed from $\mathcal{D}$. For example, given a word and its score $(\sigma_{0:T}, s)$, we set the entries $\mathbf{H}_{\mathcal{B}}(\lambda, \sigma_{0:T})$, $\mathbf{H}_{\mathcal{B}}(\sigma_0, \sigma_{1:T})$, $\ldots$, $\mathbf{H}_{\mathcal{B}}(\sigma_{0:T}, \lambda)$, and $\mathbf{H}_\sigma(\sigma_{0:t-1}, \sigma_{t+1:T})$, where $\sigma = \sigma_t$ with value s. To find a low rank factorization of $\mathbf{H}_{\mathcal{B}}$, we use truncated singular value decomposition, $\mathbf{H}_{\mathcal{B}} = \mathbf{U}_m \boldsymbol{\Lambda}_m \mathbf{V}_m^\top$ where $\boldsymbol{\Lambda}_m$ is a diagonal matrix of the m largest singular values and $\mathbf{U}_m, \mathbf{V}_m$ are the corresponding column vectors, and set $\mathbf{P} = \mathbf{U}_m$ and $\mathbf{S} = \boldsymbol{\Lambda}_m \mathbf{V}_m^\top$. Finally, the vectors and matrices $\psi = \{\boldsymbol{\alpha}_0, \boldsymbol{\beta}, \{\mathbf{W}_\sigma\}\}$ of the WFA can be obtained from $\mathbf{P}$, $\mathbf{S}$, $\{\mathbf{H}_\sigma\}$ using Theorem 1.

Planning in a Product WFA-MDP System

Given a learned WFA representation h_ψ and an initial cost estimate c_θ, we propose a planning algorithm to solve the deterministic optimal control problem in (3.30) and obtain a control policy $\pi_\theta(\mathbf{u}|\mathbf{x})$ as in (3.32). To determine the termination condition for the problem in (3.30), we define the product of the WFA, modeling the task, and the L-MDP, modeling the agent-environment interactions.

Definition 6. Given an L-MDP $\{\mathcal{X},\mathcal{U},\mathbf{x}_0,f,c,\mathcal{AP},\ell\}$ and a WFA $\{\boldsymbol{\alpha}_0,\boldsymbol{\beta},\{\mathbf{W}_\sigma\}\}$, a product WFA-MDP model is a tuple $\{\mathcal{S},\mathcal{U},\mathbf{s}_0,T,\mathcal{S}_F,c,\mathcal{AP},\ell\}$ where $\mathcal{S}=\mathcal{X}\times\mathbb{R}^m$ is the product state space, $\mathbf{s}_0=(\mathbf{x}_0,\boldsymbol{\alpha}_0)$ is the initial state, and $\mathcal{S}_F=\left\{(\mathbf{x},\boldsymbol{\alpha})\in\mathcal{S}\mid\boldsymbol{\alpha}^\top\boldsymbol{\beta}\geq\xi\right\}$ are the final states. The function $T:\mathcal{S}\times\mathcal{U}\to\mathcal{S}$ is a deterministic transition function such that $T((\mathbf{x}_t,\boldsymbol{\alpha}_t),\mathbf{u}_t)=(\mathbf{x}_{t+1},\boldsymbol{\alpha}_{t+1})$ where $\mathbf{x}_{t+1}=f(\mathbf{x}_t,\mathbf{u}_t)$, emitting symbol $\sigma_t=\ell(\mathbf{x}_t,\mathbf{u}_t)$ and causing transition $\boldsymbol{\alpha}_{t+1}=\mathbf{W}_{\sigma_t}^\top\boldsymbol{\alpha}_t$.

To obtain the agent policy in (3.32) for any state $\mathbf{x}_t\in\mathcal{X}$ and control $\mathbf{u}_t\in\mathcal{U}$, our goal is to compute the optimal cost-to-go values for the WFA-MDP model:

$$Q_\theta(\mathbf{s}_t,\mathbf{u}_t)=c_\theta(\mathbf{x}_t,\mathbf{u}_t)+V_\theta(T(\mathbf{s}_t,\mathbf{u}_t))=c_\theta(\mathbf{x}_t,\mathbf{u}_t)+\min_{T,\mathbf{u}_{t+1:T}}\sum_{k=t+1}^{T}c_\theta(\mathbf{x}_k,\mathbf{u}_k)\qquad(3.34)$$

where $\mathbf{s}_{t+1}=T(\mathbf{s}_t,\mathbf{u}_t)$ and $\boldsymbol{\alpha}_{T+1}^\top\boldsymbol{\beta}\geq\xi$. We have rewritten the terminal state condition as $\boldsymbol{\alpha}_{T+1}^\top\boldsymbol{\beta}\geq\xi$, where we keep track of the task hidden state $\boldsymbol{\alpha}_t$ using the WFA-MDP transition function T. Our key observation is that (3.34) is a deterministic shortest path problem and $V_\theta(T(\mathbf{s}_t,\mathbf{u}_t))$ can be obtained via any shortest path algorithm, such as Dijkstra [38], A* [109] or RRT* [86]. When we use a shortest path algorithm to update the cost-to-go values of successor states $\mathbf{s}_{t+1}=T(\mathbf{s}_t,\mathbf{u}_t)$, we concurrently compute the corresponding WFA state $\boldsymbol{\alpha}_{t+1}=\mathbf{W}_{\sigma_t}^\top\boldsymbol{\alpha}_t$ where $\sigma_t=\ell(\mathbf{x}_t,\mathbf{u}_t)$. A goal state $\mathbf{s}_{T+1}$ is reached when its WFA state $\boldsymbol{\alpha}_{T+1}$ satisfies $\boldsymbol{\alpha}_{T+1}^\top\boldsymbol{\beta}\geq\xi$. The agent policy π_θ in (3.32) with respect to the current cost estimate c_θ can be obtained from the cost-to-go values Q_θ in (3.34) computed by the shortest path algorithm.

Optimizing Cost Parameters

We discuss how to differentiate the loss function $\mathcal{L}_c(\theta)$ in (3.33) with respect to θ through the deterministic shortest path problem defined by the product WFA-MDP model. [171] introduce a sub-gradient descent approach to differentiate the log likelihood of expert demonstrations from the Bolzman policy in (3.32) through the optimal cost-to-go values in (3.34). The cost parameters can be updated by (stochastic) subgradient descent at each iteration k

with learning rate $\gamma^{(k)}$, $\theta^{(k+1)} = \theta^{(k)} - \gamma^{(k)}\nabla\mathcal{L}_c(\theta^{(k)})$. Intuitively, the subgradient descent makes the trajectory starting with a demonstrated control more likely, while those with other controls less likely. The analytic subgradient computation is presented below.

Proposition 4. *[171, Proposition 1] Consider an expert transition* $(\mathbf{x}_t, \mathbf{u}_t)$. *Define* $\boldsymbol{\tau}(\mathbf{x}_t, \mathbf{u})$ *as the optimal path starting from state* $\mathbf{x}_t$ *and any control* $\mathbf{u} \in \mathcal{U}$ *that achieves* $Q_\theta(\mathbf{x}_t, \mathbf{u})$ *in* (3.34) *under cost estimate* c_θ. *A subgradient of the agent policy* (3.32) *evaluated at expert transition* $(\mathbf{x}_t, \mathbf{u}_t)$ *with respect to cost parameters* θ *can be obtained via the chain rule as:*

$$\begin{aligned}\frac{\partial \log \pi_\theta(\mathbf{u}_t \mid \mathbf{x}_t)}{\partial \theta} &= \sum_{\mathbf{u}' \in \mathcal{U}} \frac{d \log \pi_\theta(\mathbf{u}_t \mid \mathbf{x}_t)}{d Q_\theta(\mathbf{x}_t, \mathbf{u}')} \frac{\partial Q_\theta(\mathbf{x}_t, \mathbf{u}')}{\partial \theta} \\ &= \sum_{\mathbf{u}' \in \mathcal{U}} \frac{1}{\eta} \left(\mathbb{1}_{\{\mathbf{u}'=\mathbf{u}_t\}} - \pi_\theta(\mathbf{u}_t \mid \mathbf{x}_t)\right) \times \sum_{(\mathbf{x},\mathbf{u}) \in \boldsymbol{\tau}(\mathbf{x}_t, \mathbf{u}')} \frac{\partial Q_\theta(\mathbf{x}_t, \mathbf{u}')}{\partial c_\theta(\mathbf{x}, \mathbf{u})} \frac{\partial c_\theta(\mathbf{x}, \mathbf{u})}{\partial \theta} \end{aligned} \tag{3.35}$$

Substituting (3.35) in the gradient of $\mathcal{L}_c(\theta)$ in (3.33), Proposition 4 provides an explicit subgradient computation to allow backpropagation with respect to θ through the value function Q_θ of the deterministic shortest path problem in (3.34). The subgradient only affects the cost parameters through the optimal trajectories $\boldsymbol{\tau}(\mathbf{x}_t, \mathbf{u}')$, $\forall \mathbf{u}' \in \mathcal{U}$ for expert transitions $(\mathbf{x}_t, \mathbf{u}_t)$ which can be retrieved from any optimal planning algorithm applied in Sec 3.2.2. Thereafter, the cost parameters can be optimized depending on the specific form of $\frac{\partial c_\theta(\mathbf{x}, \mathbf{u})}{\partial \theta}$.

Our complete approach WFA-IRL is summarized in Fig. 3.20. We first solve Problem 2 to find a WFA h_ψ which models the demonstrated task. The learned WFA provides termination conditions for a deterministic shortest path problem in the product WFA-MDP. Cost parameters are optimized by backpropagating the loss in (3.33) through the planning algorithm.

Neural Network Cost Representation

We use a neural network, shown in Fig. 3.21 to learn a nonlinear cost function c_θ, mapping from each state-control pair to a non-negative cost value. The cost neural network is separated into two parts. The first part is a feature extractor which processes each input type accordingly.

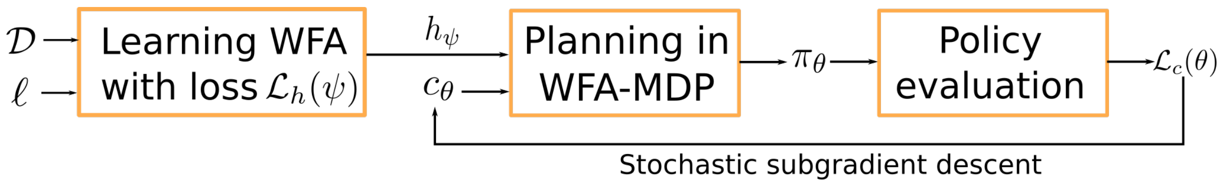


Figure 3.20. WFA-IRL architecture for joint learning of a task specification h_ψ and cost function c_θ. Given demonstrations $\mathcal{D}$ and a labeling function ℓ, we learn the unknown task specification with a weighted finite automaton. We construct a product WFA-MDP space from the learned WFA $\psi = \left\{\boldsymbol{\alpha}_0, \boldsymbol{\beta}, \{\mathbf{W}_\sigma\}_{\sigma\in\Sigma}\right\}$ to solve a deterministic shortest path problem with cost estimate c_θ. The agent policy π_θ is compared with the demonstrated controls to backpropagate the loss $\mathcal{L}_c(\theta)$ with respect to θ.

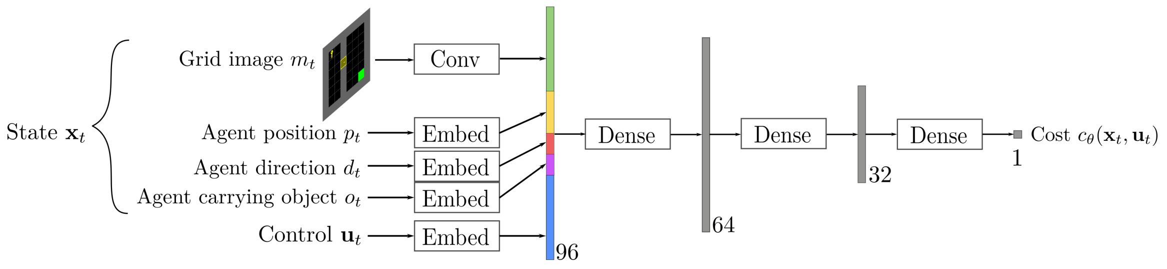


Figure 3.21. Neural network architecture for the transition cost $c_\theta(\mathbf{x}_t, \mathbf{u}_t)$. The state $\mathbf{x}_t$ consists of the grid image $\mathbf{m}_t$, the agent position $\mathbf{p}_t$, direction d_t, object it is carrying o_t. The grid m_t is fed through a convolutional neural network (Conv), while the discrete variables $\mathbf{p}_t$, d_t, o_t and the control $\mathbf{u}_t$ are converted to embedding vectors (Embed) to provide latent representations for learning the cost function. The concatenated vector of Conv and Embed layer outputs is passed through three fully-connected layers (Dense) to obtain $c_\theta(\mathbf{x}_t, \mathbf{u}_t)$.

The grid image is passed through a convolutional neural network, consisting of 3 stacks of convolution + ReLU layers with $\{16, 32, 64\}$ filters of size 2. The agent position, direction, object carried and control are discrete variables and are passed through embedding layers to produce high-dimensional feature vectors. The embedding dimensions are $\{128, 64, 64, 128\}$ respectively. The outputs from each feature extractor are flattened and concatenated to construct a latent vector representing the state-control pair in feature space. In the second part, a fully-connected neural network maps the latent vector to a scalar output for cost prediction. The 3 fully-connected layers have sizes $\{64, 32, 1\}$ with ReLU activation function. The cost neural network architecture is trained using Proposition 4 in PyTorch [129] with the Adam optimizer [91].

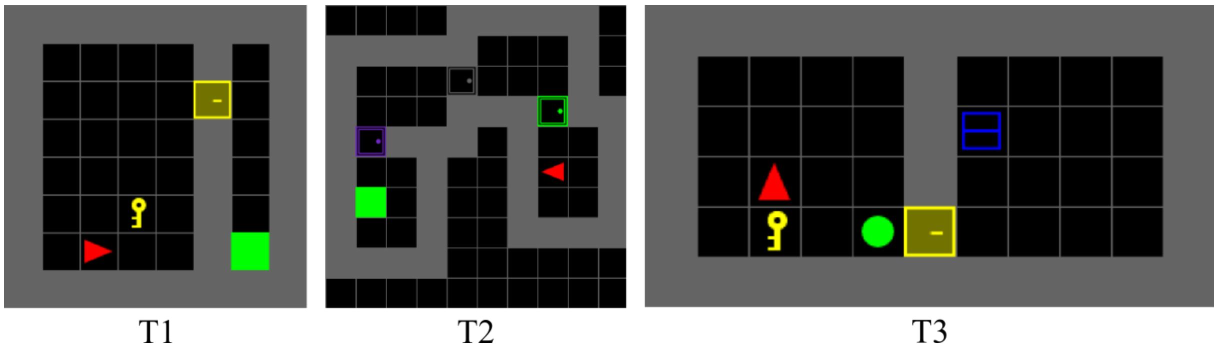


Figure 3.22. MiniGrid environments [28] of our experiments. In T1 (*MiniGrid-DoorKey-8x8-v0*) the agent must pick up *Key* to unlock *Door* and reach *Goal* in the other room. In T2 (*MiniGrid-MultiRoom-N4-S5-v0*) it has to open a series of *Door*s to reach *Goal* in the last room. In T3 (*MiniGrid-BlockedUnlockPickup-v0*) it has to move away a blocking *Ball*, unlock *Door* with *Key* and pick up *Box*. The state $\mathbf{x}_t$ includes the grid image $\mathbf{m}_t \in \{Wall, Key, Door, Box, Ball, Empty\}^{H\times W}$, the agent position $\mathbf{p}_t \in \{1,\dots,H\} \times \{1,\dots,W\}$, direction $d_t \in \{Up, Left, Down, Right\}$, and the object carried $o_t \in \{Key, Ball, Box, Empty\}$. The control space $\mathcal{U}$ is defined as turn left/right, move forward, pick up/drop/toggle an object.

3.2.3 Results

We consider three MiniGrid tasks shown in Fig. 3.22 whose atomic propositions are shown in Table 3.7. The task specifications can be expressed in terms of these propositions, e.g., one possible trajectory that fulfills task T1 is to evaluate the propositions p_1, p_2, p_3 as true sequentially.

Table 3.7. Atomic propositions used in each task.

	T1	T2	T3
p_1	*Key* is picked up	*Door* 1 is open	*Ball* is 2 steps away from *Door*
p_2	*Door* is open	*Door* 2 is open	*Key* is picked up
p_3	Agent reaches *Goal*	*Door* 3 is open	*Door* is open
p_4	———	Agent reaches *Goal*	*Box* is picked up

Demonstrations

An expert trajectory is collected by iteratively rolling out the controls sampled from the expert policy π^* at each state $\mathbf{x}$, where $Q^*(\mathbf{x}, \mathbf{u})$ in (3.30) is computed via the Dijkstra's

Table 3.8. Results on MiniGrid environment tasks. In each entry, Green / Orange are results trained from demonstrations $\mathcal{D}_1$ with expert policy temperature $\eta = 0$ and $\mathcal{D}_2$ with $\eta = 0.5$, respectively. Top: Best Scikit-SpLearn hyperparameters that solve Problem 2 for each task. Bottom: Mean episode returns (or negative cumulative true cost, higher is better) are reported across 64 randomly generated test environments.

	T1	T2	T3
`rank`	5/9	6/9	7/11
`rows`	4/5	5/5	6/6
`cols`	4/5	5/6	6/7

	T1	T2	T3
BC	0.364/0.253	0.338/0.307	0.284/0.192
GAIL	0.483/0.429	0.274/0.185	0.342/0.257
WFA-IRL(**ours**)	**0.797**/**0.708**	**0.776**/**0.642**	**0.733**/**0.602**
WFA-IRL w/o WFA	0.683/0.514	0.652/0.488	0.566/0.390
Expert	0.798/0.718	0.776/0.668	0.734/0.639
Optimal	0.798	0.776	0.734
Random	0.000	0.000	0.000

algorithm with cost of 1 for any feasible transition. For each task, we consider two sets of expert demonstrations $\mathcal{D}_1$ and $\mathcal{D}_2$, each with 32 trajectories collected from expert policies with temperatures $\eta \in \{0, 0.5\}$. The expert trajectories in $\mathcal{D}_1$ and $\mathcal{D}_2$ are strictly optimal and suboptimal, respectively, and are labeled with score $s = 1$. In each set we also add 128 failed trajectories with score $s = 0$ from a random exploration policy (effectively setting expert policy temperature $\eta \to \infty$). The full demonstration set is used in each case to learn a WFA representation h_ψ of the task via the spectral method in Sec. 3.2.2 while only the successful trajectories are used to learn the cost function c_θ, as in Sec. 3.2.2 and 3.2.2.

Our Method and Baselines

Our method WFA-IRL uses a neural network architecture to represent the cost function. For a detailed description, please refer to Appendix 3.2.2. We use the spectral learning algorithm in the Scikit-SpLearn toolbox [6] to learn the parameters ψ of the WFA from the expert demonstrations $\mathcal{D}$. In the implementation, we first compress the demonstration words $\sigma^n_{0:T_n}$ where consecutive identical symbols are removed. This greatly reduces the complexity of learning the WFA while keeping the symbol sequences unchanged. The hyperparameters for the spectral learning method are the `rank` of the automaton (m in Theorem 1) and the sizes (`rows` and `cols`) of the prefix and suffix basis ($\mathcal{B} = (\mathcal{P}, \mathcal{S})$ in Theorem 1), which determine the size of the Hankel matrix estimated empirically from demonstrations. The complexity is $O(n \times \texttt{rank} \times \texttt{rows} \times \texttt{cols})$ since we iterate through the ranks to find a minimal WFA and count the prefix and suffix frequencies for a given word. The spectral method can learn almost perfectly with near zero loss in (3.31) for all tasks and the best hyperparameter configurations are shown in Table 3.8. We observe that larger WFA capacity is required to learn from suboptimal trajectories and thus more diverse words from $\mathcal{D}_2$.

As baselines, we first ablate the WFA component in our method to understand its effects. Instead of planning in the product WFA-MDP space and checking the WFA termination condition in (3.34), the agent without WFA component simply plans in the original MDP and checks whether a goal state is achieved. Additionally, we compare our method with standard imitation learning and inverse reinforcement learning algorithms, including behavioral cloning (BC) [141] and GAIL [77][3]. The value/policy functions in these baselines follow our cost neural network architecture to fit the state-control input format and to compare fairly in representation power across methods. This includes the policy network in BC, the discriminator in GAIL, the policy and value networks in PPO [145], used as generator in GAIL. Only the size of the last fully-connected layer is modified depending on whether it is action or value prediction. GAIL is known to achieve stable performance in fixed horizon environments while the MiniGrid environments terminate as soon as the agent fulfills the tasks. We fix this issue by adding a virtual absorbing state as suggested in [96] when training GAIL.

MiniGrid Results

We report the average performance of each method in Table 3.8 by testing on 64 new environment configurations generated randomly for each task. First, we observe that our method can achieve almost perfect performance when trained on $\mathcal{D}_1$. This is expected since the learned WFA strictly chooses planned trajectories whose words would match the optimal behavior. Interestingly, the learned WFA could make the agent suboptimal if the optimal word in testing is

[3]The implementations are adapted from the imitation learning library [170]

not seen in training, as shown in Fig. 3.23. Next, our method matches the expert performance well using either $\mathcal{D}_1$ or $\mathcal{D}_2$ and outperforms BC and GAIL (even without WFA). This demonstrates that using planning to solve tasks that encode logical structures performs better than a reactive policy employed by BC. Moreover, the performance gap between our method and ours without WFA shows that learning logic specifications explicitly with a WFA can further improve the policy. On the other hand, we find the performance of GAIL is limited as PPO cannot easily generate successful samples similar to the demonstrations (notice that the random policy never succeeds) to improve the cost discriminator and, in turn, the generator itself. We visualize the agent policy in Fig. 3.24 and observe that our method has a stronger bias on controls that follow the learned logical sequences.

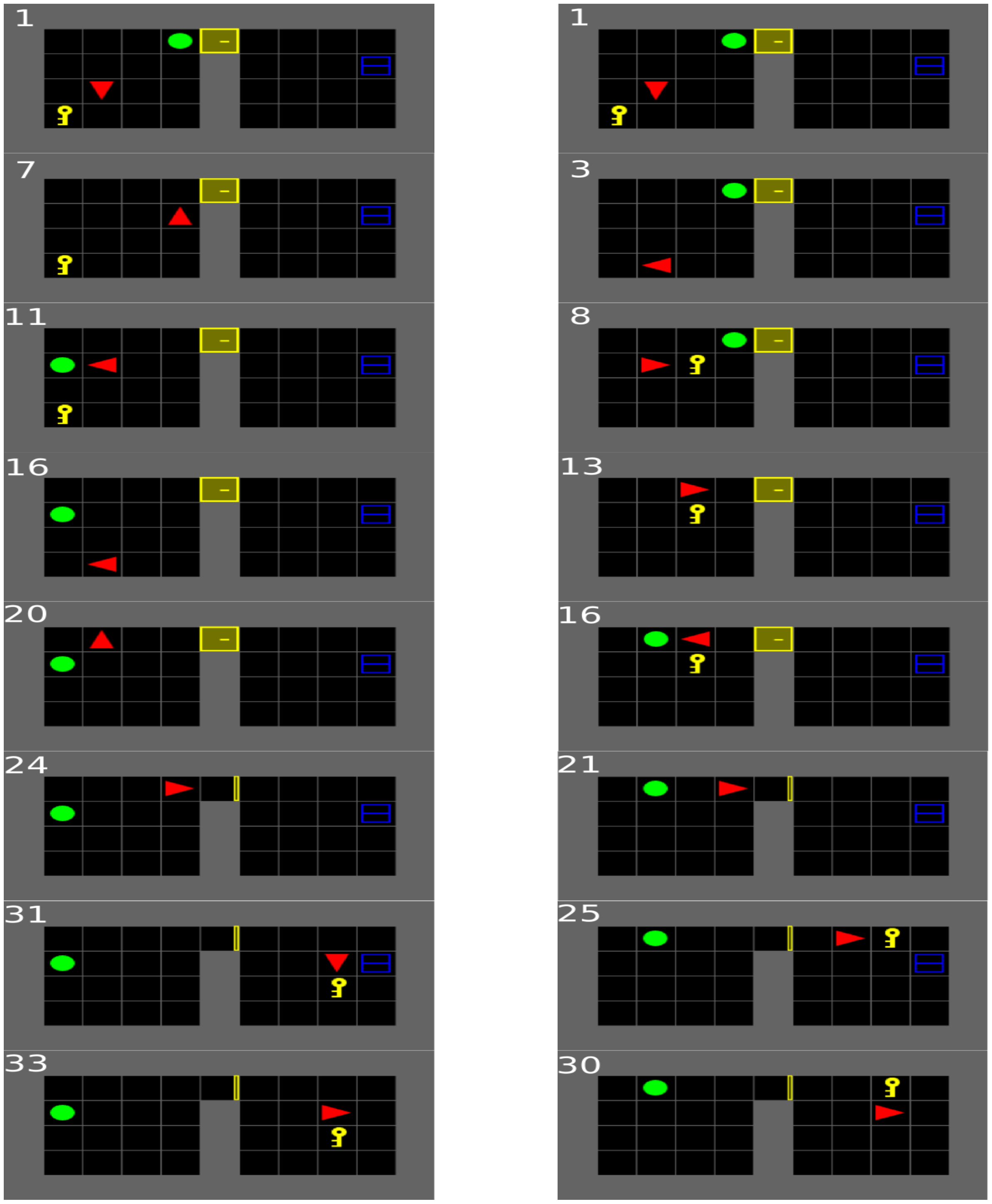


Figure 3.23. Agent trajectory (left column) trained with $\mathcal{D}_1$ and expert trajectory (right column) in task T3 during testing. The agent WFA only learns words that appear in demonstration $\mathcal{D}_1$, which always moves *Ball* away from *Door* first before picking up *Key*. In testing it fails to recognize a lower cost trajectory of a different word sequence, where *Key* is carried closer to *Door* before moving away *Ball*.

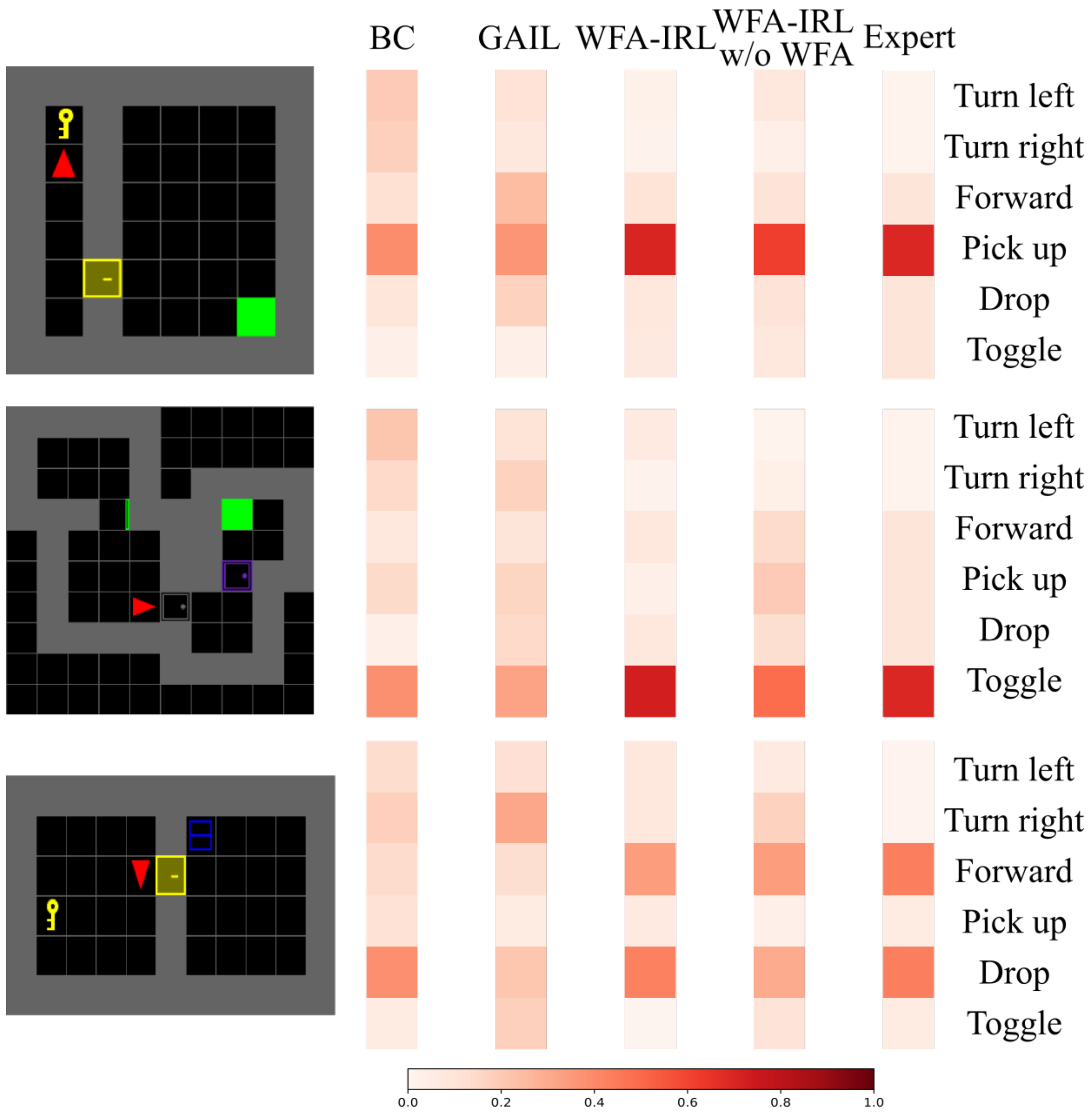


Figure 3.24. Visualization of policy probabilities of each method trained on $\mathcal{D}_2$ at a critical state in T2. Our method shows a stronger preference towards controls (toggle door) that can make task progress.

3.3 Summary

This chapter introduces inverse reinforcement learning approaches to learn navigation policies from expert demonstrations. In Sec. 3.1, we introduce cost functions that can learn from semantic features and in Sec. 3.2 we encode high-level task logic as weighted finite automata. Furthermore, we propose a differentiable motion planning algorithm that efficiently optimizes the cost function parameters using an objective function that maximizes the likelihood of the expert demonstrated behavior. The proposed approaches allow us to generalize the navigation policy to unseen environments with dynamic obstacles by understanding semantic entities and solving long-horizon, sequential and compositional planning problems.

3.4 Acknowledgements

Chapter 3.1, in part, is a reprint of the following papers. The dissertation author is the primary author of these papers.

- T. Wang, V. Dhiman and N. Atanasov, "Learning Navigation Cost from Demonstrations in Partially Observable Environments," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4434-4440, 2020.
- T. Wang, V. Dhiman and N. Atanasov, "Learning Navigation Costs from Demonstrations with Semantic Observations," *Learning for Dynamics and Control*, pp. 245–255, 2020.
- T. Wang, V. Dhiman and N. Atanasov, "Inverse Reinforcement Learning for Autonomous Navigation via Differentiable Semantic Mapping and Planning," *Autonomous Robots*, 47(6), 809-830.

Chapter 3.2, in part, is a reprint of the material as it appears in T. Wang and N. Atanasov, "Inverse Reinforcement Learning of Autonomous Behaviors Encoded as Weighted Finite Automata", *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp 7429-7435, 2022. The dissertation author is the primary author of this paper.

Chapter 4

Simulation to Real Generalization for Robot Manipulation

In the previous chapter, we have introduced inverse reinforcement learning approaches for imitating robot navigation behaviors from expert demonstrations. In this chapter, we focus on applying inverse reinforcement learning methods on robot manipulators. Robot manipulation systems typically consist of a robot arm and an end-effector and classical control methods are used in assembling and packaging in manufacture, planting and harvesting in agriculture, etc. However, robot manipulators cannot easily achieve versatile skills like washing dishes, peeling a banana or arranging furniture. In recent years, deep reinforcement learning and imitation learning methods have been applied to learn complex manipulation skills and adapt to diverse environments and tasks from experience and demonstrations. While typical RL agents require millions of environment interactions to specialize is a single task, we consider learning internal robot representations such that the policy is not re-trained for different robots performing the same task. In particular, we will consider domain adaptation for policy generalization across different types of robots and transfer learning from simulation to real world.

Consider a manipulation task requiring a robot arm to control a gripper (see Fig. 4.1). A canonical parametrization of the state and action space utilizes joint angles, angular velocities or torques. Our insight is that a successful trajectory in the joint configuration space may be complicated but its embedding in the end effector (e.g., fingers or gripper) configuration space is relatively simple. We postulate that discovering such a latent space through joint optimization with the task objective can facilitate training convergence and policy adaptation across different types of robot arms in both simulation and real world settings.

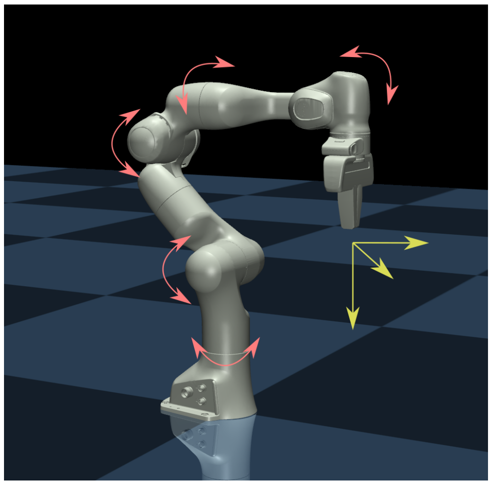


Figure 4.1. Learning a policy for robot joint torques (red) to achieve an expert-demonstrated manipulation task is challenging due to the high dimensional configuration space. We propose to abstract the action space to the gripper pose movement (yellow) to allow imitation learning in a lower-dimensional action space.

Learning invariant feature representations have shown promising for solving downstream RL and domain transfer tasks. Guo et al. [65] and Hafner et al. [71] independently propose to learn a latent recurrent state model from pixel observations and bootstrap latent dynamics to learn latent predictions of future observations. Pari et al. [127] use a BYOL-style [63] self-supervised learning framework to learn latent representations from visual observations offline and then find nearest neighbors from demonstrations to predict actions. Zhang et al. [187] use virtual reality teleoperation system to align human demonstrations with robot arms for imitation learning. Wang et al. [169] consider visual planning problem for robot manipulation where a causal InfoGAN model [100] generates future visual observations with latent planning and a learned inverse dynamics tracks the predicted observation sequence. Nair at el. [119] pre-train latent visual representations from large dataset using time-contrastive learning [148] and video-language alignment [118] to be used in downstream robot manipulation tasks. Das et al. [37] use visual keypoints as latent features to imitate human demonstrations for robot manipulation tasks.

Transfer learning is considered a challenging direction in reinforcement learning. Zhang et al. [185] propose to learn invariant representations through bisimulation metrics [47] where states are considered similar if they have similar immediate rewards and state distributions under the same action sequence. Wulfmeier et al. [173] finetune a source policy on the target robot of the same type but different dynamics by encouraging similar state distribution. However, it still performs RL on the target environment which assumes access to the reward function and environment interactions. Zakka et al. [183] use temporal consistency constraint from paired source and target samples for cross-embodiment imitation learning. Hejna et al. [75] consider cross-morphology transfer learning by finetuning hierarchical policies with KL constraint. Zhang et al. [186] learn a direct state and action correspondence between source and target domains with a cycle consistency constraint. However, they do not construct a latent space so correspondence has to be re-trained if a third domain is introduced. Stadie et al. [155] consider imitation learning under viewpoint mismatch where feature extractor learns to be invariant to viewpoint changes. Kim et al. [90] also considers matching state distributions via generative adversarial networks across domains in an imitation learning setting. Our work does not require expert demonstrations in the target domain. Yin et al. [179] learn a latent invariant representation for a robot with different physical parameters (e.g. link length). Their method cannot be applied if source and target robots have different morphologies (e.g. different number of links). Yoneda et al. [180] aligns latent features from source and target samples with adversarial training and dynamics consistency constraint. While their approach only considers visual adaptation of the same robot, we consider a more general setting of aligning robots of different dynamics and morphology.

Different from common manipulation tasks like grasping and opening drawers, dynamic manipulation abandons the quasi-static assumption of interaction. It leverages object dynamics, such as inertia and momentum, to handle tasks requiring high-speed actions and extended robot workspace, like throwing and catching [114, 56, 147, 93]. Traditional systems for such tasks often rely on handcrafted models of system dynamics, which may fall short when dealing with difficult-to-estimate parameters or new objects. To address this issue, recent works [94, 59, 184, 29] employ data-driven approaches to optimize control commands using partial dynamics models. For example, Chi *et al.* [29] propose an iterative residual policy to solve tasks with complex dynamics; Zeng *et al.* [184] use end-to-end training to learn stable grasps that generate predictable throws. However, these works focus on low-DoF manipulators rather than high-DoF dexterous hands, which introduce additional complexities due to intricate hand-object interactions. [130] uses Population Based Training (PBT) to scale up the training of dexterous manipulation in simulation, and it also trains a task that throws and catches using two hands. However, while our methodologies primarily focus on sim2real transfer, [130] does not include the real-world experiments. In this study, we use a learning-based approach to tackle dynamic problems, specifically throwing and catching, with multi-finger dexterous hands. We explore the impact of initial dexterous grasps on throwing performance and investigate strategies to bridge the sim2real gap in the context of dynamic manipulation.

In recent years, the robotics community has increasingly focused on dexterous manipulation due to its great flexibility and human-like dexterity. Researchers have developed methods using dexterous hands for tasks such as grasping [177, 19, 67, 33, 134], in-hand rotating [178, 72, 132, 2, 26], and manipulating deformable objects [10, 48, 108, 79]. Similar to us, DexPoint [133] achieves generalizable manipulation for grasping and door opening by training on multiple objects with a Allegro hand. However, complex tasks like throwing and catching objects require a bimanual robot system to achieve human-level manipulation skills. Researchers have investigated bimanual manipulation through task planning [167, 193, 175], and reinforcement learning [30, 3, 87]. However, most previous work focused on using two parallel jaw grippers for quasi-static interaction, leaving dexterous bimanual manipulation largely unexplored. Few studies have delved into this area, mainly in simulation without real robot validation [23, 27, 182]. In this section, we take a step forward by developing a bimanual dexterous manipulation system capable of throwing and catching various objects. Our work also demonstrates that simulation training without real-world data can still tackle this challenging task even with great sim2real gap.

We introduce three approaches for learning robot manipulations skills in this chapter. First, we propose *LAtent Policies for Adversarial Imitation Learning* (*LAPAL*), which trains an action encoder-decoder model to provide a latent structured action space for efficient adversarial imitation learning. We demonstrate that the action encoder-decoder model can be trained offline with expert demonstrations only to learn latent action representations, and trained online to learn latent representations that align with the imitation learning task objective. Next, we generalize *LAPAL* by learning both state and action encoder-decoder models. We propose to project robots of different embodiments into a common latent representation space using adversarial distribution matching and cycle consistency training objectives. Finally, we demonstrate a simulation to real transfer approach for learning bimanual throw and catch skill. In summary, the contributions of this chapter are:

- We propose *LAPAL*, an action encoder-decoder model which learns a structured latent action representation via adversarial imitation learning. The latent action space can be aligned with expert demonstrations or imitation learning task objective for efficient training and finetuning the action encoder-decoder model allows us to generalize policy across different robot manipulators.
- We propose cross embodiment robot manipulation skill transfer, where robots of different dynamics and morphology can be projected into a common latent space using adversarial distribution matching and cycle consistency constraint. The latent policy trained from a source robot can be transferred to a target robot without finetuning with target domain reward function or expert demonstrations.
- We propose an approach to enable simulation to real world transfer for bimanual robot manipulation. We demonstrate that our multi-finger robot hands learn robust policies to throw and catch diverse types of objects at high speed in real world.

4.1 Latent Policies for Adversarial Imitation Learning

4.1.1 Problem Formulation

Consider a continuous-space control task formulated as a discrete-time Markov decision process (MDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, T, r, \gamma\}$, where $\mathbf{s} \in \mathcal{S} \subseteq \mathbb{R}^n$ is the state, $\mathbf{a} \in \mathcal{A} \subseteq \mathbb{R}^m$ is the action, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ is an unknown transition function from state $\mathbf{s}$ to state $\mathbf{s}'$ under action $\mathbf{a}$, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function and $\gamma \in (0,1)$ is a discount factor. In IRL, the reward function r is not known to the agent. Instead, it infers a reward function from a set of expert demonstrations $\{(\mathbf{s}_t, \mathbf{a}_t)\}_t$ sampled from an expert policy $\pi_E : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$.

We denote the state-action marginal of the trajectory distribution induced by a policy π in $\mathcal{M}$ as $\rho_\pi(\mathbf{s}, \mathbf{a})$. The imitation learning objective can be formulated as minimizing a general f-divergence between the state-action occupancy measures under a parameterized agent policy π_θ, and the expert policy π_E,

$$\min_\theta D_f \left[\rho_{\pi_\theta}(\mathbf{s}, \mathbf{a}) \;||\; \rho_{\pi_E}(\mathbf{s}, \mathbf{a})\right], \tag{4.1}$$

where $D_f[p||q] := \int f(\frac{p(\mathbf{s},\mathbf{a})}{q(\mathbf{s},\mathbf{a})}) q(\mathbf{s}, \mathbf{a}) d\mathbf{s} d\mathbf{a})$ measures the difference between two probability distributions $p, q \in \mathcal{P}(\mathcal{S} \times \mathcal{A})$ in the space of density functions over $\mathcal{S} \times \mathcal{A}$, using a convex generator function $f : [0, \infty) \mapsto (-\infty, \infty]$. For example, AIRL [54] optimizes the Kullback-Leibler divergence ($f(x) = x \ln x$), and GAIL [77] optimizes the Jensen-Shannon divergence ($f(x) = \frac{1}{2}(x-1)\ln x$) [77, 61]. In this work, we consider the Jensen-Shannon divergence but the development can be generalized to other types of divergence functions as well. GAIL iteratively trains a discriminator D_ϕ, and a generator policy π_θ, with the following mini-max objective

[77, 61]:

$$\begin{aligned}\min_{\theta}\max_{\phi} J(\theta,\phi) &= \min_{\theta} D_{JS}\left[\rho_{\pi_\theta}(\mathbf{s},\mathbf{a}) \,||\, \rho_{\pi_E}(\mathbf{s},\mathbf{a})\right] \\ &= \min_{\theta}\max_{\phi} \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\pi_E}\left[\log D_\phi(\mathbf{s},\mathbf{a})\right] + \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\pi_\theta}\left[\log(1-D_\phi(\mathbf{s},\mathbf{a}))\right] \qquad (4.2)\end{aligned}$$

where $D_\phi : \mathcal{S}\times\mathcal{A}\to[0,1]$ is the discriminator function classifying the probability of a given state-action pair from the expert. In this section, we will focus on solving (4.2) by introducing a latent policy defined on a learned latent action space.

4.1.2 Learning Latent Action Representation

We consider lifting the original MDP to a new MDP, $\bar{\mathcal{M}} = \left\{\mathcal{S},\bar{\mathcal{A}},\bar{T},\bar{r},\gamma\right\}$ where $\bar{\mathcal{A}}$ is a (lower-dimensional) latent action space, and $\bar{T} : \mathcal{S}\times\bar{\mathcal{A}}\times\mathcal{S}\to[0,1]$ and $\bar{r} : \mathcal{S}\times\bar{\mathcal{A}}\to\mathbb{R}$ are the transition probability and reward function defined in the latent space. We assume there exist optimal action encoder-decoder functions $g : \mathcal{A}\to\bar{\mathcal{A}}$ and $h : \bar{\mathcal{A}}\to\mathcal{A}$, such that $\bar{T}(\mathbf{s},g(\mathbf{a})) = T(\mathbf{s},\mathbf{a}), \bar{r}(\mathbf{s},g(\mathbf{a})) = r(\mathbf{s},\mathbf{a}), \forall \mathbf{s}\in\mathcal{S},\mathbf{a}\in\mathcal{A}$. We further assume that the expert policy support lies within the image of the latent action decoder, i.e., $supp(\pi_E)\subset h(\bar{\mathcal{A}})\subset\mathcal{A}$. The insight is that the expert policy only operates on a subset of the original configuration space. Therefore, learning a policy on such subspace is more efficient and it prevents explorations on the original action space where the optimal policy would never visit. In this section, we present a method to learn the latent action representations from expert demonstrations. In the following sections, we show that the learned action encoder-decoder functions can be fixed and (4.2) reduces to solving an imitation learning problem in latent MDP $\bar{\mathcal{M}}$, or they can be merged within the adversarial learning algorithm with their parameters optimized by gradients of (4.2).

Following the motivation in Fig. 4.1, we propose to learn a latent action space such as the space of the robot's end-effector pose movements from it orginal action space of joint torques. To determine a latent action, the end-effector pose movement not only depends on the joint torques applied but also the current robot joint configuration state. Similarly, finding the joint torques such that the end-effector achieves the desired movement is also conditioned on the robot joint state. Inspired by these observations, we propose to use a conditional variational autoencoder (CVAE) [153] to model the action encoder-decoder functions and learn a latent represetation of the original action space. The action encoder g_{ω_1} maps an action $\mathbf{a}$ to a latent distribution $\Phi(\cdot \mid \mathbf{s})$, e.g. diagonal Gaussian, conditioned on the state $\mathbf{s}$, and samples a latent action $\bar{\mathbf{a}}$ from Φ. The action decoder $h_{\omega_2} : \mathcal{S} \times \bar{\mathcal{A}} \to \mathcal{A}$ maps a latent action to the original action space, conditioned on the state $\mathbf{s}$. The loss function to train CVAE is

$$\mathcal{L}_{CVAE}(\mathbf{s},\mathbf{a}) = ||\mathbf{a} - h_{\omega_2}(\mathbf{s},\bar{\mathbf{a}})||_2^2 + \beta D_{KL}[\Phi(\bar{\mathbf{a}} \mid \mathbf{s}) \mid\mid p(\bar{\mathbf{a}} \mid \mathbf{s})] \tag{4.3}$$

where we assume a state-independent prior $p(\bar{\mathbf{a}} \mid \mathbf{s}) = p(\bar{\mathbf{a}})$. The first reconstruction loss term ensures that we learn a latent action space that is consistent with the original action space and the second KL loss term regularizes the encoded latent posterior distribution. The coefficient β is an adjustable hyperparameter that balances latent model capacity and reconstruction accurary [76]. We train the CVAE model with expert demonstrations only so that the learned action space captures latent representations of the expert policy and not those of any random exploration policy for the reasons described in the previous paragraph.

4.1.3 Task-agnostic and Task-aware Action Embedding for Adversarial Imitation Learning

Task-agnostic Action Embedding

In this section, we assume the action encoder-decoder functions $g_{\omega_1}, h_{\omega_2}$ are trained as in Sec. 4.1.2 and the parameters ω_1, ω_2 are fixed. Given $g_{\omega_1}, h_{\omega_2}$, we can induce a latent MDP $\bar{\mathcal{M}}$ as described in Sec. 4.1.2. We consider an imitation learning problem in the latent MDP $\bar{\mathcal{M}}$ and the mini-max objective anagolous to (4.2) is

$$\begin{aligned}\min_{\bar{\theta}}\max_{\phi} J(\bar{\theta},\phi) &= \min_{\bar{\theta}} D_{JS}\left[\rho_{\bar{\pi}_{\bar{\theta}}}(\mathbf{s},\bar{\mathbf{a}}) \,||\, \rho_{\bar{\pi}_E}(\mathbf{s},\bar{\mathbf{a}})\right] \\ &= \min_{\bar{\theta}}\max_{\phi} \mathbb{E}_{(\mathbf{s},\bar{\mathbf{a}})\sim\bar{\pi}_E}\left[\log D_{\phi}(\mathbf{s},\bar{\mathbf{a}})\right] + \mathbb{E}_{(\mathbf{s},\bar{\mathbf{a}})\sim\bar{\pi}_{\bar{\theta}}}\left[\log(1-D_{\phi}(\mathbf{s},\bar{\mathbf{a}}))\right] \end{aligned} \tag{4.4}$$

Here, we abuse the notation $(\mathbf{s},\bar{\mathbf{a}})\sim\bar{\pi}_E$ to denote that we have converted an expert transition $(\mathbf{s},\mathbf{a})\sim\pi_E$ into the latent action space via $\bar{\mathbf{a}} = g_{\omega_1}(\mathbf{s},\mathbf{a})$. The latent agent policy $\bar{\pi}_{\bar{\theta}} : \mathcal{S}\times\bar{\mathcal{A}}\to[0,1]$ predicts a latent action $\bar{\mathbf{a}}$ at state $\mathbf{s}$, which is converted back to the original action space, $\mathbf{a} = h_{\omega_2}(\mathbf{s},\bar{\mathbf{a}})$, before applying it to the agent.

For discriminator optimization, we compute the gradient of the objective with respect to discriminator parameters ϕ:

$$\nabla_{\phi} J(\bar{\theta},\phi) = \mathbb{E}_{(\mathbf{s},\bar{\mathbf{a}})\sim\bar{\pi}_E}\left[\nabla_{\phi}\log D_{\phi}(\mathbf{s},\bar{\mathbf{a}})\right] + \mathbb{E}_{(\mathbf{s},\bar{\mathbf{a}})\sim\bar{\pi}_{\bar{\theta}}}\left[\nabla_{\phi}\log(1-D_{\phi}(\mathbf{s},\bar{\mathbf{a}}))\right]. \tag{4.5}$$

We can exchange the expectation with differentiation since the expectation terms do not depend on ϕ. The expert latent actions are converted from the original state-action pair via the action encoder $\bar{\mathbf{a}} = g_{\omega_1}(\mathbf{s},\mathbf{a})$ and the agent latent action is sampled from the latent policy $\bar{\pi}$.

For generator optimization, we denote the discriminator D_{ϕ} evaluated at the optimal parameter ϕ^* as $D^* := D_{\phi^*}$. Since $\bar{\theta}$ appears in the probability distribution and not inside the expectation in the second term of (4.4), a gradient estimator can be obtained from the policy gradient theorem [159, 144],

$$\begin{aligned}\nabla_{\bar{\theta}} D_{JS}\left[\rho_{\bar{\pi}_{\bar{\theta}}}(\mathbf{s},\bar{\mathbf{a}}) \,||\, \rho_{\bar{\pi}_E}(\mathbf{s},\bar{\mathbf{a}})\right] &= \nabla_{\bar{\theta}}\mathbb{E}_{(\mathbf{s},\bar{\mathbf{a}})\sim\bar{\pi}_{\bar{\theta}}}\left[\log(1-D^*(\mathbf{s},\bar{\mathbf{a}}))\right] \\ &= \mathbb{E}_{(\mathbf{s},\bar{\mathbf{a}})\sim\bar{\pi}_{\bar{\theta}}}[\nabla_{\bar{\theta}}\log\bar{\pi}_{\bar{\theta}}(\bar{\mathbf{a}}\mid\mathbf{s})Q^{\bar{\pi}}(\mathbf{s},\bar{\mathbf{a}})] \end{aligned} \tag{4.6}$$

where $Q^{\bar{\pi}}(\mathbf{s},\bar{\mathbf{a}}) = \mathbb{E}_{(\mathbf{s}_t,\bar{\mathbf{a}}_t)\sim\bar{\pi}_{\bar{\theta}}}[\sum_{t=0}^{\infty}\gamma^t\log(1-D^*(\mathbf{s}_t,\bar{\mathbf{a}}_t))\mid\mathbf{s}_0=\mathbf{s},\bar{\mathbf{a}}_0=\bar{\mathbf{a}}]$ is the value function of $\bar{\pi}$ starting from $(\mathbf{s},\bar{\mathbf{a}})$. Effectively, the policy optimization step is to apply on-policy reinforcement

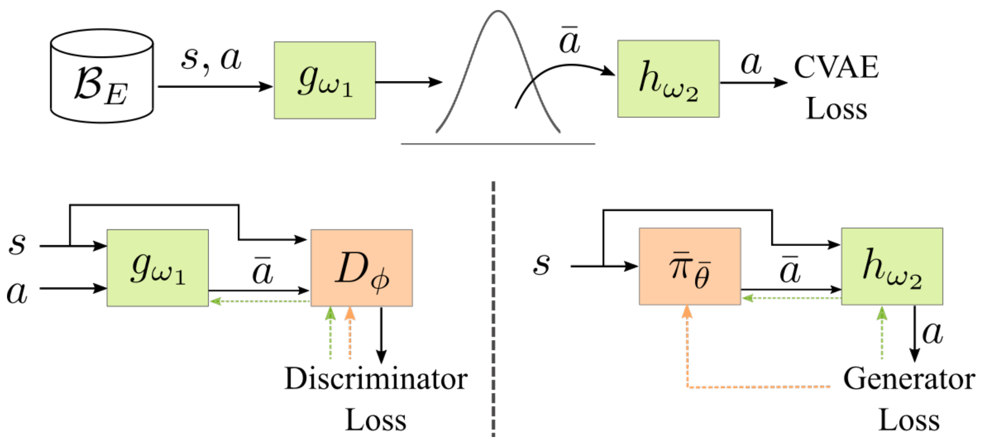


Figure 4.2. LAPAL overview. We first train action encoder-decoder functions $g_{\omega_1}, h_{\omega_2}$ with a conditional variational autoencoder (CVAE) on expert demonstrations $\mathcal{B}_E$ to extract latent action representation. In adversarial imitation learning, we iteratively train a discriminator D_ϕ that classifies state and latent action pairs $(\mathbf{s}, \bar{\mathbf{a}})$ and train a generator $\bar{\pi}_{\bar{\theta}}$ that predicts latent actions from states. For task-agnostic LAPAL, we update the discriminator and generator parameters $\phi, \bar{\theta}$ by formulating a imitation learning objective in the latent space (orange dashed line). For task-aware LAPAL, the action encoder-decoder functions can be jointly optimized with the discriminator and policy (green dashed line).

learning with reward $\bar{r}(\mathbf{s}, \bar{\mathbf{a}}) = -\log(1 - D^*(\mathbf{s}, \bar{\mathbf{a}}))$ using on-policy samples from $\bar{\pi}_{\bar{\theta}}$. In practice, off-policy reinforcement learning algorithms, e.g., soft actor-critic (SAC) [70], can be applied to replace the expectation under the policy distribution with expectation under the agent replay buffer distribution [95, 18]. It is no longer guaranteed that the agent visitation distribution will match that of the expert but allows off-policy training for sample efficiency. In addition, since we optimize the discriminator and generator iteratively, we do not obtain the optimal discriminator D^* but rather use the reward $\bar{r}(\mathbf{s}, \bar{\mathbf{a}}) = -\log(1 - D_\phi(\mathbf{s}, \bar{\mathbf{a}}))$ defined over the discriminator parameters in the current iteration.

To sum up, in task-agnostic imitation learning, we assume that the action encoder-decoder functions $g_{\omega_1}, h_{\omega_2}$ are trained and their parameters are fixed. We apply GAIL in the latent MDP $\bar{\mathcal{M}}$ where the discriminator D_ϕ classifies $(\mathbf{s}, \bar{\mathbf{a}})$, and the policy $\bar{\pi}_\theta$ is trained with reward $\bar{r}(\mathbf{s}, \bar{\mathbf{a}}) = -\log(1 - D_\phi(\mathbf{s}, \bar{\mathbf{a}}))$ using SAC.

Task-aware Action Embedding

In this section, we consider the case where the action encoder-decoder parameters ω_1, ω_2 are trainable, and solve the imitation learning optimization problem for the original MDP $\mathcal{M}$. We rewrite the minimax objective in (4.2) as

$$\begin{aligned}
&\min_{\bar{\theta},\omega_2}\max_{\phi,\omega_1} J(\bar{\theta},\phi,\omega_1,\omega_2) \\
&\quad= \min_{\bar{\theta},\omega_2}\max_{\phi,\omega_1} \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\pi_E}\left[\log D_\phi(\mathbf{s}, g_{\omega_1}(\mathbf{s},\mathbf{a}))\right] + \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\pi_{\bar{\theta},\omega_2}}\left[\log(1-D_\phi(\mathbf{s}, g_{\omega_1}(\mathbf{s},\mathbf{a})))\right] \quad (4.7)
\end{aligned}$$

where we have written the latent action as $\bar{\mathbf{a}} = g_{\omega_1}(\mathbf{s},\mathbf{a})$ and the latent policy samples $(\mathbf{s},\bar{\mathbf{a}}) \sim \pi_{\bar{\theta}}$ followed by action decoding $\mathbf{a} = h_{\omega_2}(\mathbf{s},\bar{\mathbf{a}})$ as $(\mathbf{s},\mathbf{a}) \sim \pi_{\bar{\theta},\omega_2}$ to explicitly bring out the dependency on the parameters ω_1, ω_2.

For the discriminator update, we now compute the gradient with respect to both ϕ and ω_1 since they affect the classification probability of a state-action pair $(\mathbf{s},\mathbf{a})$,

$$\begin{aligned}
&\nabla_{\phi,\omega_1} J(\bar{\theta},\phi,\omega_1,\omega_2) \\
&\quad= \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\pi_E}\left[\nabla_{\phi,\omega_1}\log D_\phi(\mathbf{s}, g_{\omega_1}(\mathbf{s},\mathbf{a}))\right] + \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\pi_{\bar{\theta},\omega_2}}\left[\nabla_{\phi,\omega_1}\log(1-D_\phi(\mathbf{s}, g_{\omega_1}(\mathbf{s},\mathbf{a})))\right].
\end{aligned} \quad (4.8)$$

For the generator update, we again notice that the policy parameters $\bar{\theta}, \omega_2$ only appear in the probability distribution and not inside the expectation. The policy gradient theorem can still be applied to obtain a gradient estimator,

$$\begin{aligned}
\nabla_{\bar{\theta},\omega_2} D_{JS}\left[\rho_{\pi_{\bar{\theta},\omega_2}}(\mathbf{s},\mathbf{a}) \,||\, \rho_{\pi_E}(\mathbf{s},\mathbf{a})\right] &= \nabla_{\bar{\theta},\omega_2}\mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\pi_{\bar{\theta},\omega_2}}\left[\log(1-D^*(\mathbf{s}, g_{\omega_1}(\mathbf{s},\mathbf{a})))\right] \\
&= \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\pi_{\bar{\theta},\omega_2}}\left[\nabla_{\bar{\theta},\omega_2}\log\pi_{\bar{\theta},\omega_2}(\mathbf{a}\mid\mathbf{s})Q^\pi(\mathbf{s},\mathbf{a})\right],
\end{aligned} \quad (4.9)$$

where $Q^\pi_{\bar{\theta},\omega_2}(\mathbf{s},\mathbf{a}) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\pi_{\bar{\theta},\omega_2}}[\sum_{t=0}^\infty \gamma^t \log(1-D^*(\mathbf{s}_t, g_{\omega_1}(\mathbf{s}_t,\mathbf{a}_t))) \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}]$ is the value function of $\pi_{\bar{\theta},\omega_2} = h_{\omega_2} \circ \bar{\pi}_{\bar{\theta}}$ starting from $(\mathbf{s},\mathbf{a})$. In practice, we can optimize the agent policy

$\pi_{\bar{\theta},\omega_2}$ using any off-policy reinforcement learning algorithm, e.g. SAC, with reward function $r(\mathbf{s},\mathbf{a}) = -\log(1-D_\phi(\mathbf{s}, g_{\omega_1}(\mathbf{s},\mathbf{a}))$ as described in the previous section.

The architechture for task-agnostic and task-aware LAPAL is illustrated in Fig. 4.2 and the complete algorithm is summarized in Algorithm 4.

Algorithm 4. Latent Policies for Adversarial Imitation Learning (LAPAL)

```
1: **Input**: Expert demonstration buffer $\mathcal{B}_E$
2: Randomly initialize action encoder-decoder $\{g_{\omega_1}, h_{\omega_2}\}$, discriminator $D_\phi$, latent policy $\bar{\pi}_{\bar{\theta}}$,
   and empty agent replay buffer $\mathcal{B}_\pi$
3: Train action encoder-decoder $g_{\omega_1}, h_{\omega_2}$ with CVAE loss (4.3) using expert transitions from
   $\mathcal{B}_E$
4: **for** number of training iterations **do**
5:     **for** number of experience collection steps **do**
6:         Sample latent action $\bar{\mathbf{a}}_t \sim \bar{\pi}_{\bar{\theta}}(\cdot \mid \mathbf{s}_t)$
7:         Decode action $\mathbf{a}_t = h_{\omega_2}(\mathbf{s}_t, \bar{\mathbf{a}}_t)$
8:         Add $(\mathbf{s}_t, \mathbf{a}_t)$ to agent replay buffer $\mathcal{B}_\pi$
9:         Step environment for next state $\mathbf{s}_{t+1}$
10:    **for** number of adversarial training steps **do**
11:        Sample a minibatch $\mathcal{B}$ from joint buffer $\mathcal{B}_E \cup \mathcal{B}_\pi$
12:        Encode actions $\bar{\mathbf{a}} = g_{\omega_1}(\mathbf{s},\mathbf{a})$ for $(\mathbf{s},\mathbf{a}) \in \mathcal{B}$
13:        **if** using task-agnostic LAPAL **then**
14:            Update $\phi$ with gradient (4.5)
15:            Update $\bar{\theta}$ with reward $\bar{r}(\mathbf{s},\bar{\mathbf{a}}) = -\log(1-D_\phi(\mathbf{s},\bar{\mathbf{a}}))$ using SAC [70]
16:        **else if** using task-aware LAPAL **then**
17:            Update $\{\phi, \omega_1\}$ with gradient (4.8)
18:            Update $\{\bar{\theta}, \omega_2\}$ with reward $r(\mathbf{s},\mathbf{a}) = -\log(1-D_\phi(\mathbf{s},\bar{\mathbf{a}}))$ using SAC [70]
```

Characterizing LAPAL Latent Policy

In this section, we discuss a theoretical aspect of our method where the objective function that we optimize is a lower bound to the original imitation learning algorithm objective. Given a fixed action embedding function $f: \mathcal{A} \to \bar{\mathcal{A}}$, we induce a latent MDP $\bar{\mathcal{M}}$. The imitation learning objective function in $\mathcal{M}$ is given as $D_f\left[\rho_{\pi_\theta}(\mathbf{s},\mathbf{a}) \mid\mid \rho_{\pi_E}(\mathbf{s},\mathbf{a})\right]$ while that in $\bar{\mathcal{M}}$ is $D_f\left[\rho_{\bar{\pi}_{\bar{\theta}}}(\mathbf{s},\bar{\mathbf{a}}) \mid\mid \rho_{\bar{\pi}_E}(\mathbf{s},\bar{\mathbf{a}})\right]$. The data processing theorem states that the latter is a lower bound for the former,

$$D_f\left[\rho_{\bar{\pi}}(\mathbf{s},\bar{\mathbf{a}}) \mid\mid \rho_{\bar{\pi}_E}(\mathbf{s},\bar{\mathbf{a}})\right] \le D_f\left[\rho_{\pi}(\mathbf{s},\mathbf{a}) \mid\mid \rho_{\pi_E}(\mathbf{s},\mathbf{a})\right]. \tag{4.10}$$

In other words, any processing of the ground truth state-action $(\mathbf{s}, \mathbf{a})$ makes it more difficult to determine whether it came from the expert or the agent policy. As a result, the latent policy $\bar{\pi}$ that we solve in $\bar{\mathcal{M}}$ can be suboptimal in $\mathcal{M}$ after action decoding, $\pi = h \circ \bar{\pi}$. Empirically we find that our task-agnostic LAPAL model can achieve a suboptimal performance in low-dimensional systems where the action dimension is already small and embedding action space might lose important information for imitating expert policy. For high-dimensional systems, the learned latent policy still matches the expert performance.

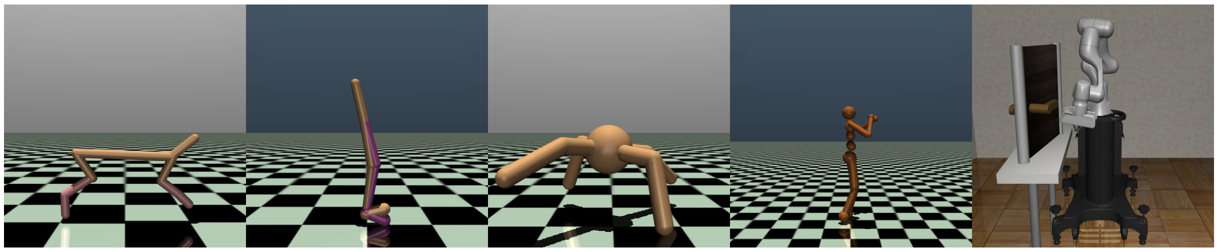


Figure 4.3. Benchmark environments from MuJoCo and robosuite: (left to right) HalfCheetah-v3, Walker2d-v3, Ant-v3, Humanoid-v3, Door.

4.1.4 Experiments

Benchmark Tasks for Imitation Learning

We use four continuous control locomotion environments from MuJoCo [165, 21] and one manipulation environment from robosuite [191], presented in Fig. 4.3. For the locomotion environments, the task is to run at high speed without falling or exerting too much control effort. For the Door manipulation environment, the task is to open the door using a robot arm with a two-finger gripper. The original state and action dimensions are provided in Table 4.1.

We evaluate task-agnostic and task-aware LAPAL against GAIL [77]. Each algorithm is provided with 64 expert demonstrations, collected from a policy trained with soft actor-critic (SAC) on the ground truth reward function. We use the same set of hyperparameters across GAIL and task-agnostic/aware LAPAL and the neural network architectures are shown in Table 4.2. The generator policy SAC is adapted from [136] and its critic and actor networks share weights

Table 4.1. Environment state and action space dimensions

Environment	State dimension	Action dimension
HalfCheetah-v3	17	6
Walker2d-v3	17	6
Ant-v3	111	8
Humanoid-v3	376	17
Door (Panda/Sawyer)	46	8

Table 4.2. Neural network configurations

Module	Hidden layer size	Activation	Learning rate
Action encoder	(256, 256)	Leaky ReLU	3e-4
Action decoder	(256, 256)	Leaky ReLU	3e-4
Discriminator	(256, 256)	Tanh	3e-5
Generator actor	(256, 256, 256)	ReLU	3e-4
Generator critic	(256, 256)	ReLU	3e-4

for the first two layers. For LAPAL models, we add a Tanh activation layer to the action decoder output to match the action space bounds. The networks are trained in PyTorch [129] with the Adam optimizer [91]. The latent action dimension is set to 4 for both task-agnostic and task-aware LAPAL to infer the latent task structure of these robotic locomotion and manipulation tasks.

Fig. 4.4 shows the learning curves of each method. For low-dimensional systems (HalfCheetah-v3, Walker2d-v3), task-agnostic LAPAL is on par or worse than the baseline GAIL since the original action space dimension is small enough and further compressing the latent space could result in suboptimal performance as discussed in Sec. 4.1.3. For high dimensional systems (Ant-v3, Humanoid, Door), both task-agnostic and task-aware LAPAL demonstrate faster convergence than GAIL. Specifically in the complex Humanoid-v3 environment, our models can achieve the expert baseline performance, while GAIL fails to recover an optimal policy in the original action space without explicit regularization techniques like gradient penalty [64, 123] and spectral normalization [117].

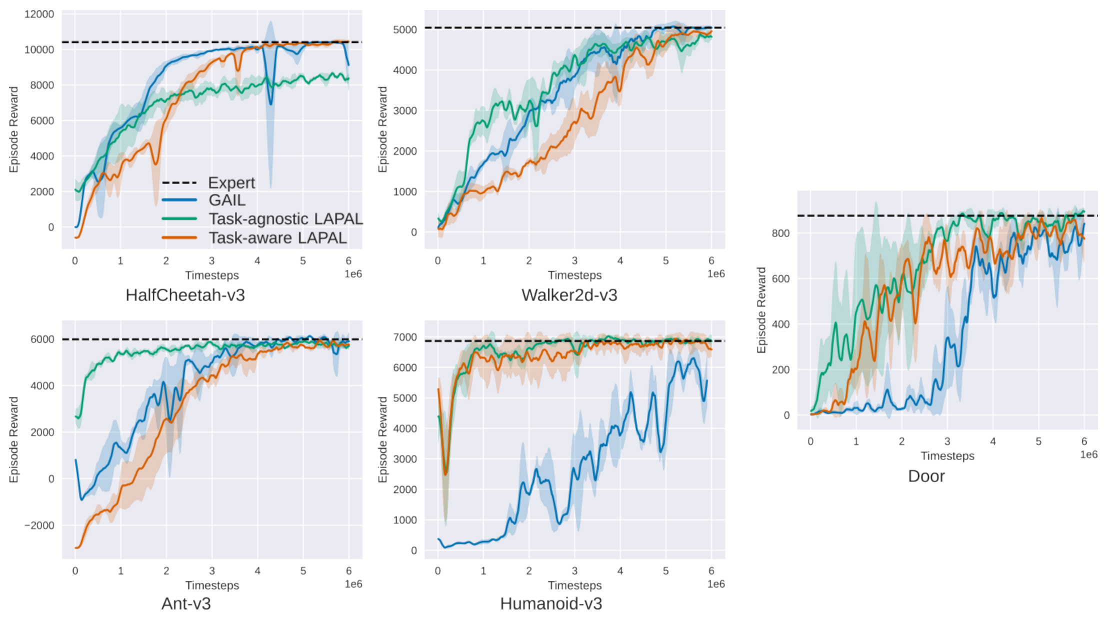


Figure 4.4. Benchmark results for MuJoCo and robosuite tasks. Each algorithm is averaged over 3 random seeds and the shaded area indicates standard deviation. LAPAL is on par with GAIL in low-dimensional problems such as Walker2d-v3 and HalfCheetah-v3 but converges faster for high-dimensional problems like Ant-v3, Humanoid and Door. In high-dimensional Humanoid-v3, GAIL fails to recover the optimal policy without addition regularization, e.g. gradient penalty and spectral normalization, while LAPAL converges quickly and asymptotically.

Ablation Studies

We perform ablation on the Humanoid-v3 task to investigate the influence of the latent action dimension d_a on task-agnostic LAPAL. Humanoid-v3 has the largest action space dimension among the tasks studied in this section. Fig. 4.5 shows the learning curves of task-agnostic LAPAL with $d_a = 4, 8$ and 16. We provide 16 expert trajectories instead of 64 as in the previous benchmark evaluation to emphasize the difference between our ablated models and the baseline GAIL. The performance of task-agnostic LAPAL is close to optimal when $d_a = 4$ or 8 but drops about 10% when $d_a = 16$. In comparison, the baseline GAIL applied in the original action space with 17 dimensions only reaches 60% of the expert performance when few demonstrations are provided. This shows that AIL methods can suffer from high action space dimensionality.

We also study the effect of spectral normalization on each model in Fig. 4.5. Empirically, we find that spectral normalization [117] improves the model performance while penalty gradient [64] does not and is omitted here. With spectral normalization, GAIL is able to reach expert

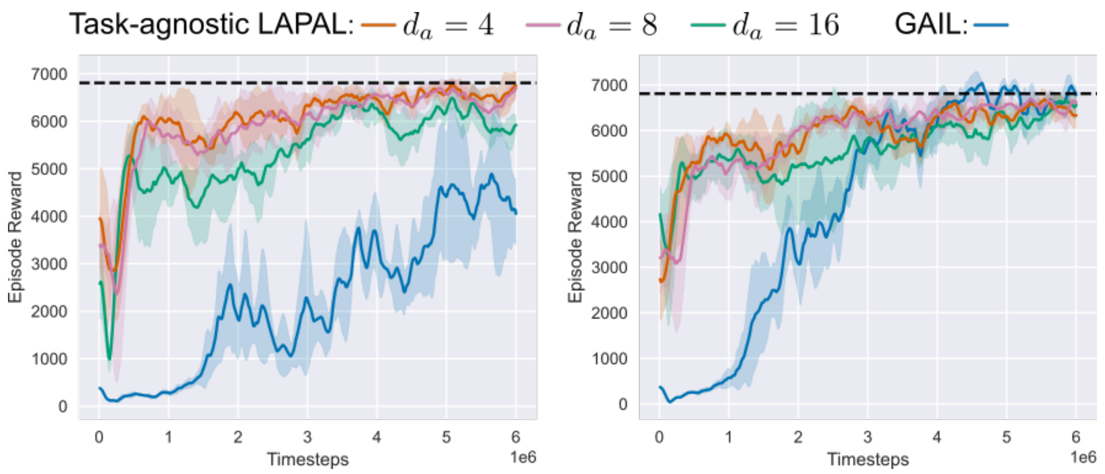


Figure 4.5. Ablation analysis of task-agnostic LAPAL with latent action dimension d_a in Humanoid-v3. Each model is averaged over 5 random seeds and the shaded area indicates standard deviation. Without spectral normalization (left), task-agnostic LAPAL with large $d_a = 16$ performs worse than those with $d_a = 4$ or 8, while GAIL using the original action space ($d_a = 17$) has the lowest return. With spectral normalization (right), all models reach expert performance but the effect of the action dimension size is still apparent in the training speed.

Table 4.3. Average return of each policy over 16 episodes and 5 random seeds in the Sawyer robot target environment $\mathcal{M}_t$. The source policy is trained with a Panda robot in a source environment $\mathcal{M}_s$ and directly applied to $\mathcal{M}_t$. Behavioral cloning (BC) and GAIL are trained with demonstrations from $\mathcal{M}_t$. The transferred policy is a composition of the latent policy $\bar{\pi}_s$ trained in $\mathcal{M}_s$ and the action decoder h_t trained in $\mathcal{M}_t$.

	Expert policy	Source policy	BC	GAIL	Transferred policy
Average return	857 ± 21	35 ± 6	657 ± 159	752 ± 53	732 ± 46

level and the differences in task-agnostic LAPAL with varying d_a are small. However, using a lower-dimensional action embedding still leads to faster training. Additionally, we notice that in the previous experiments, the performance of both task-agnostic and task-aware LAPAL degrades in the beginning of adversarial training. We suspect that this is due to pre-training the action encoder-decoder with expert demonstrations. When the weights of the action encoder-decoder are optimized for the CVAE loss (4.3), they might not be a good initialization for training the discriminator and generator under the adversarial loss. With spectral normalization applied, the performance drop of task-agnostic LAPAL is mitigated as it stablizes gradients during

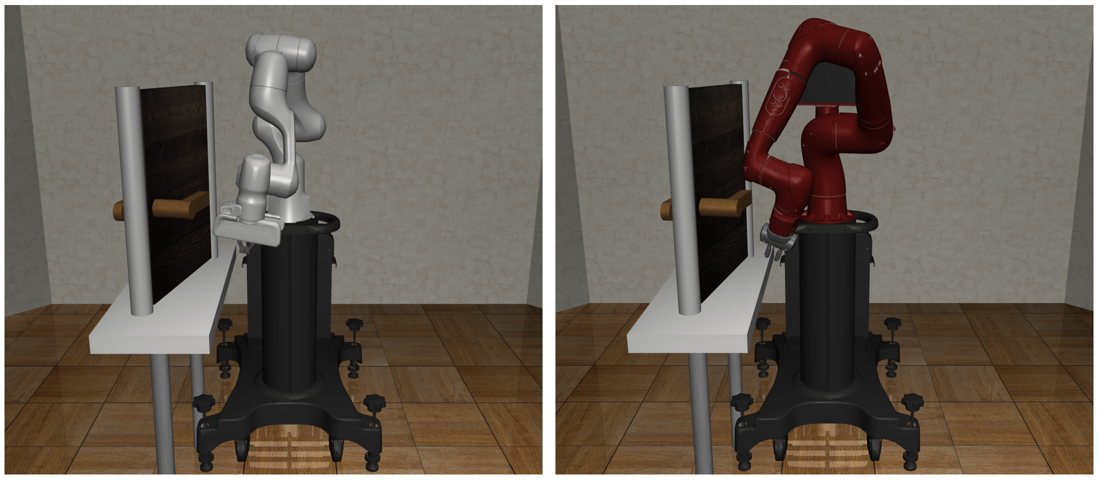


Figure 4.6. Zero-shot transfer learning for task-agnostic LAPAL from Panda robot (left) to Sawyer robot (right) in Door environment.

discriminator training.

Transfer Learning in Robosuite Door

Our task-agnostic LAPAL model is suited for transfer learning problems in robotic tasks. We consider transferring skills acquired from one robot in a source environment $\mathcal{M}_s$ to another robot in a task environment $\mathcal{M}_t$. Specifically, we train task-agnostic LAPAL in the robosuite Door environment with a Panda robot arm and deploy it on a Sawyer robot arm in the target environment (see Fig. 4.6). The two robots have the same degrees of freedom but the joint configurations (geometry, friction, damping) and gripper models are different. Directly applying a policy trained from one robot to another for the same task does not work.

We first apply task-agnostic LAPAL in $\mathcal{M}_s$ to acquire a latent policy $\bar{\pi}_s$ for an "open door" skill. We then use 64 expert demonstrations in $\mathcal{M}_t$ to train a new action encoder-decoder $\{g_t, h_t\}$ as described in Sec. 4.1.2 for the Sawyer arm. Combining the latent policy $\bar{\pi}_s$ in $\mathcal{M}_s$ and the new action decoder h_t we obtain a transferred policy for $\mathcal{M}_t$. We consider this as zero-shot transfer since it does not require additional online interactions with $\mathcal{M}_t$. On the other hand, training GAIL directly in the target environment $\mathcal{M}_t$ will require collecting online samples from $\mathcal{M}_t$. Table 4.3 shows that the transferred policy obtains an mean return of 732. For baseline comparisons, the expert policy on $\mathcal{M}_t$ has mean return 857 but directly applying the expert policy from $\mathcal{M}_s$ to $\mathcal{M}_t$ only achieves 35. Without considering transferability or leveraging any source task data, applying behavioral cloning (BC) and GAIL in $\mathcal{M}_t$ achieves mean returns of 657 and 752, respectively. The transferred policy from LAPAL is better than BC alone in $\mathcal{M}_t$ and almost matches GAIL trained in $\mathcal{M}_t$. This illustrates that LAPAL learns an informative latent policy that may be generalized to different robot types.

4.2 Cross Embodiment Robot Manipulation Skill Transfer from Cycle Consistency

4.2.1 Problem Formulation

A Markov decision process (MDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, r, T, \gamma\}$ consists of a continuous state space $\mathcal{S}$, a continuous action space $\mathcal{A}$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a probabilistic transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$, and a discount factor $\gamma \in [0,1]$. We consider a source MDP $\mathcal{M}^s = \{\mathcal{S}^s, \mathcal{A}^s, r^s, T^s, \gamma\}$ and a target MDP $\mathcal{M}^t = \{\mathcal{S}^t, \mathcal{A}^t, r^t, T^t, \gamma\}$. In general, the state and action spaces of the source and target MDPs are different. We aim to align the source and the target domains by defining a latent-space MDP $\mathcal{M}^z = \{\mathcal{S}^z, \mathcal{A}^z, r^z, T^z, \gamma\}$. We introduce a state encoder $F^s : \mathcal{S}^s \to \mathcal{S}^z$ and an action encoder $G^s : \mathcal{S}^s \times \mathcal{A}^s \to \mathcal{A}^z$ to map source state-action pairs to the latent MDP, as well as decoders $\tilde{F}^s : \mathcal{S}^z \to \mathcal{S}^s$ and $\tilde{G}_s : \mathcal{S}^s \times \mathcal{A}^z \to \mathcal{A}^s$ to map latent state-action pairs back to the source MDP. Similarly, we define state-action encoders and decoders between the target and the latent MDPs $F^t, \tilde{F}^t, G^t, \tilde{G}^t$. We assume that random transitions $\mathcal{D}^s = \left\{(\mathbf{s}^s_k, \mathbf{a}^s_k, \mathbf{s}^s_{k+1})\right\}$ and $\mathcal{D}^t = \left\{(\mathbf{s}^t_k, \mathbf{a}^t_k, \mathbf{s}^t_{k+1})\right\}$ are available from the source and target domains. Our goal is to learn the state-action encoders and decoders $F^{\{s,t\}}, \tilde{F}^{\{s,t\}}, G^{\{s,t\}}, \tilde{G}^{\{s,t\}}$ such that a source policy parameterized through the latent space, $\pi^s(\mathbf{s}^s) = \tilde{G}_s(\mathbf{s}^s, \pi^z(F^s(\mathbf{s}^s)))$, can be transferred to the target domain by keeping the latent policy π^z fixed and only replacing the embedding functions, i.e., $\pi^t(\mathbf{s}^t) = \tilde{G}_t(\mathbf{s}^t, \pi^z(F^t(\mathbf{s}^t)))$. In this section, the latent policy $\pi^z : \mathcal{S}^z \to \mathcal{A}^z$ is assumed to be deterministic. The encoders and decoders provide a common latent space to align different robot emobodiments. We can reuse the latent policy when a new target robot is introduced without learning the target policy from scratch.

4.2.2 Cross Embodiment Representation Alignment

In this section, we first define source encoders and decoders and train a latent space policy. Next, we align the target domain samples to the latent space constructed in the first stage, so that the latent policy can be combined with target domain projection functions to construct a

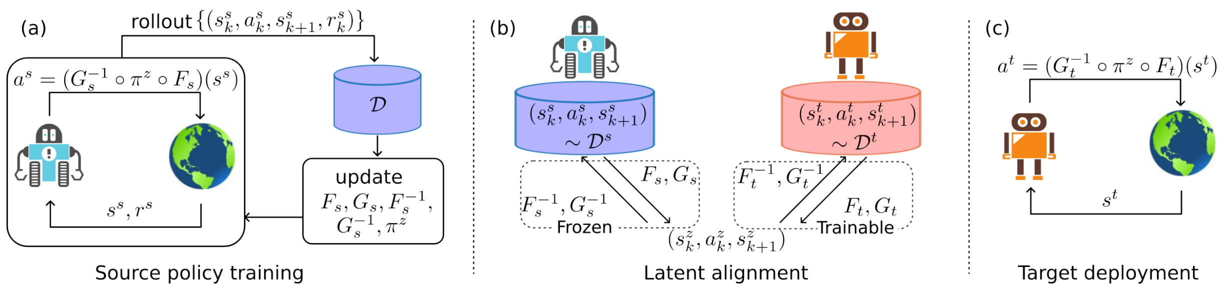

(a)
rollout $\{(s^s_k, a^s_k, s^s_{k+1}, r^s_k)\}$
$a^s = (G_s^{-1} \circ \pi^z \circ F_s)(s^s)$
$\mathcal{D}$
s^s, r^s
update $F_s, G_s, F_s^{-1}, G_s^{-1}, \pi^z$
Source policy training
(b)
$(s^s_k, a^s_k, s^s_{k+1}) \sim \mathcal{D}^s$
$(s^t_k, a^t_k, s^t_{k+1}) \sim \mathcal{D}^t$
F_s, G_s
F_s^{-1}, G_s^{-1}
Frozen
F_t^{-1}, G_t^{-1}
F_t, G_t
Trainable
$(s^z_k, a^z_k, s^z_{k+1})$
Latent alignment
(c)
$a^t = (G_t^{-1} \circ \pi^z \circ F_t)(s^t)$
s^t
Target deployment

Figure 4.7. Approach overview: (a) The source robot learns encoders and decoders $F_s, G_s, \tilde{F}_s, \tilde{G}_s$ for state-action projections between its own space and a latent space. The source robot learns a latent policy π^z simultaneously with encoders and decoders with RL. (b) During latent alignment, the source encoder decoder functions are frozen while the target encoder decoder are trained to match latent distributions as well as to satisfy cycle consistency and latent dynamics constraints. (c) During target deployment, we composite the target encoder and decoder functions trained in (b) with the latent policy trained in (a).

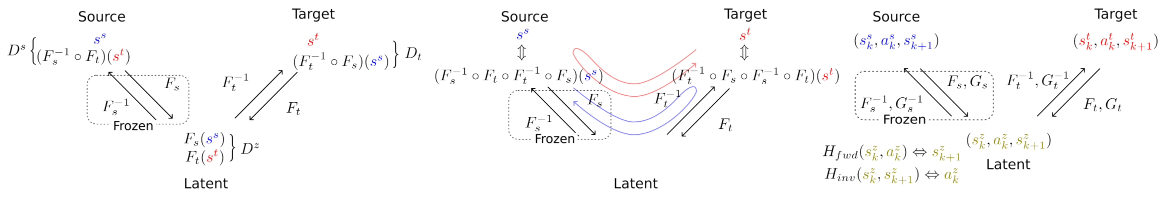

Source
Target
D^s
s^s
$(F_s^{-1} \circ F_t)(s^t)$
s^t
$(F_t^{-1} \circ F_s)(s^s)$
D_t
F_s
F_s^{-1}
Frozen
F_t^{-1}
F_t
$F_s(s^s)$
$F_t(s^t)$
D^z
Latent
Source
Target
s^s
s^t
$(F_s^{-1} \circ F_t \circ F_t^{-1} \circ F_s)(s^s)$
$(F_t^{-1} \circ F_s \circ F_s^{-1} \circ F_t)(s^t)$
F_s
F_s^{-1}
Frozen
F_t^{-1}
F_t
Latent
Source
Target
$(s^s_k, a^s_k, s^s_{k+1})$
$(s^t_k, a^t_k, s^t_{k+1})$
F_s, G_s
F_s^{-1}, G_s^{-1}
Frozen
F_t^{-1}, G_t^{-1}
F_t, G_t
$H_{fwd}(s^z_k, a^z_k) \Leftrightarrow s^z_{k+1}$
$H_{inv}(s^z_k, s^z_{k+1}) \Leftrightarrow a^z_k$
$(s^z_k, a^z_k, s^z_{k+1})$
Latent

Figure 4.8. Overview of latent alignment losses: (left) adversarial loss to match distributions in each domain, (middle) cycle consistency loss that regularizes samples to be close to themselves when translated to the other domains and back, (right) latent dynamics loss to enforce forward and inverse latent dynamics functions.

target domain policy. An overview of our approach is shown in Fig. 4.7.

Latent Policy Training with Source Domain Projection

Training a source domain policy directly does not help build a representation that allows generalization to new target domains. We parameterize a source policy in $\mathcal{M}^s$ through a latent policy in $\mathcal{M}^z$ using a state encoder F^s and an action decoder $\tilde{G}_s$: $\pi^s(\mathbf{s}^s) = \tilde{G}_s(\mathbf{s}^s, \pi^z(F^s(\mathbf{s}^s)))$. Hence, instead of directly predicting a source action from a source state, we project the source state $\mathbf{s}^s$ to a latent state $\mathbf{s}^z = F^s(\mathbf{s}^s)$, use the latent policy to predict a latent action $\mathbf{a}^z = \pi^z(\mathbf{s}^z)$, and project the latent action back to a source action, i.e., $\mathbf{a}^s = \tilde{G}^s(\mathbf{s}^s, \mathbf{a}^z)$. We propose to combine three types of objectives to learn the encoders, decoders and latent policy as discussed below.

RL Task Objective. We first optimize the latent representations of the source robot to align with the task objective. In particular, we optimize $\tilde{G}^s, F^s$ jointly with the latent policy π^z

using a deterministic policy gradient algorithm, e.g., TD3 [110, 55]. Recall that in deterministic policy gradient algorithms, the objective of a parameterized policy π_θ is to maximize the expected cumulative reward (or to minimize the expected cumulative cost):

$$\mathcal{L}_{RL}(\theta) = -\mathbb{E}_{\pi_\theta}\left[\sum_{k=0}^{\infty} \gamma^k r(\mathbf{s}_k, \mathbf{a}_k)\right]. \tag{4.11}$$

The gradient of the objective is:

$$\nabla_\theta \mathcal{L}_{RL}(\theta) = -\mathbb{E}\left[\nabla_\theta \pi_\theta(\mathbf{s}) \nabla_{\mathbf{a}} Q^{\pi_\theta}(\mathbf{s}, \mathbf{a})|_{\mathbf{a}=\pi_\theta(\mathbf{s})}\right], \tag{4.12}$$

where the action-value (Q) function of π_θ is

$$Q^{\pi_\theta}(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}\sim T_{\pi_\theta}, \mathbf{a}\sim\pi_\theta}\left[\sum_{k=0}^{\infty} \gamma^k r(\mathbf{s}_k, \mathbf{a}_k) | \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}\right],$$

i.e., the expected sum of rewards when choosing action $\mathbf{a}$ in state $\mathbf{s}$ and following π_θ afterwards. In our approach, since the policy is a composition of the encoder F^s, decoder $\tilde{G}^s$ and latent policy π^z, the policy gradient in (4.12) is backpropagated through each component to update their parameters.

Latent Dynamics Loss. The latent space is loosely constrained with a policy representation that is optimized by the task objective only. We use a self-supervision signal based on latent dynamics prediction [73] to learn forward and inverse dynamics models $T^z : \mathcal{S}^z \times \mathcal{A}^z \to \mathcal{S}^z$ and $\tilde{T}^z : \mathcal{S}^z \times \mathcal{S}^z \to \mathcal{A}^z$. The latent dynamics loss is optimized simultaneously within the RL loop and is defined as:

$$\mathcal{L}_{dyn}(T^z, \tilde{T}^z, F^s, G^s) = \mathbb{E}_{(\mathbf{s}_k^s, \mathbf{a}_k^s, \mathbf{s}_{k+1}^s)\sim D^s}\left[\left\|T^z(\mathbf{s}_k^z, \mathbf{a}_k^z) - \mathbf{s}_{k+1}^z\right\|_2^2 + \left\|\tilde{T}^z(\mathbf{s}_k^z, \mathbf{s}_{k+1}^z) - \mathbf{a}_k^z\right\|_2^2\right] \tag{4.13}$$

where $\mathbf{s}_k^z = F^s(\mathbf{s}_k^s)$ and $\mathbf{a}_k^z = G^s(\mathbf{s}_k^s, \mathbf{a}_k^s)$. Note that we use a deterministic latent dynamics model for ease of implementation while a stochastic model can also be considered [102] and is left for future work.

Reconstruction Loss. While it may not be necessary to reconstruct distractions or noise when learning latent representations from visual observations, we consider robot joint configuration as the source domain and it includes crucial information that should be captured in the latent space. Therefore, we also require F_s and $\tilde{F}_s$ (as well as G_s and $\tilde{G}_s$) to be inverse mappings of each other:

$$\mathcal{L}_{rec}(F^s, \tilde{F}^s, G^s, \tilde{G}^s) = \mathbb{E}_{(\mathbf{s}_k^s, \mathbf{a}_k^s)\sim D^s}\left[\left\|\tilde{F}^s(F^s(\mathbf{s}_k^s)) - \mathbf{s}_k^s\right\|_2^2 + \left\|\tilde{G}^s(\mathbf{s}_k^s, G(\mathbf{s}_k^s, \mathbf{a}_k^s)) - \mathbf{a}_k^s\right\|_2^2\right]. \quad (4.14)$$

The pseudo-code for learning source domain policy through latent space is shown in Alg. 5. We first initialize a replay buffer $\mathcal{D}$ containing the source random transitions $\mathcal{D}^s$. In the RL loop, we update the $F^s, \pi^z, \tilde{G}^s$ with policy gradient in (4.12) and use the same minibatch samples to optimize the latent dynamics $T^z, \tilde{T}^z$ and to enforce the encoder-decoder reconstruction loss in (4.14).

Target Domain Alignment

In the previous section, we have learned a projection $(F^s, \tilde{F}^s, G^s, \tilde{G}^s)$ from the source domain to the latent domain and a source policy π^z parameterized through that latent domain. Next, we will consider projecting the target domain to the same latent domain induced by source policy training and construct a target policy from the latent policy. Our objective function contains three types of terms as shown in Fig. 4.8: adversarial losses for matching the translated distribution in the source, target and latent domains; cycle consistency loss such that the state and action mappings can learn from unpaired samples in each domain; and finally latent dynamics consistency where target samples follow the same latent dynamics trained in the Sec. 4.2.2.

Adversarial Loss. In the previous section, we have learned the projection from the source domain to latent via encoders and decoders $F^s, \tilde{F}^s, G^s, \tilde{G}^s$, which fix the latent distribution from $\mathcal{D}^s$. We would like to train the target domain encoders and decoders $F^t, \tilde{F}^t, G^t, \tilde{G}^t$ such that the latent distribution from target domain matches that from the source. We consider an adversarial learning objective where a discriminator $D^z : \mathcal{S}^z \times \mathcal{A}^z \rightarrow [0,1]$ tries to distinguish latent state-action pairs $(F^s(\mathbf{s}^s), G^s(\mathbf{s}^s, \mathbf{a}^s))$ from the source domain and $(F^t(\mathbf{s}^t), G^t(\mathbf{s}^t, \mathbf{a}^t))$ from the target domain. Note that the distribution of $(F^s(\mathbf{s}^s), G^s(\mathbf{s}^s, \mathbf{a}^s))$ is now fixed because F^s, G^s which are trained in Sec.4.2.2 and are frozen during alignment in target domain. The target domain encoders F^t, G^t act as a generator that tries to generate latent state-action $F^t(\mathbf{s}^t), G^t(\mathbf{s}^t, \mathbf{a}^t)$ which are indistinguishable from the source latents. The adversarial objective in the latent space can be expressed as follows:

$$\max_{D^z} \min_{F^t, G^t} \mathcal{L}^z_{GAN}(F^t, G^t, D^z) =$$
$$\mathbb{E}_{(\mathbf{s}^s, \mathbf{a}^s) \sim \mathcal{D}^s} [\log D^z(F^s(\mathbf{s}^s), G^s(\mathbf{s}^s, \mathbf{a}^s))] + \quad \mathbb{E}_{(\mathbf{s}^t, \mathbf{a}^t) \sim \mathcal{D}^t} \left[\log(1 - D^z(F^t(\mathbf{s}^t), G^t(\mathbf{s}^t, \mathbf{a}^t)))\right]. \tag{4.15}$$

Additionally, we can also consider matching the translated distributions in the source and target domains. For example, in the target domain, the translated state-action from source are $\bar{\mathbf{s}}^t = \tilde{F}^t(F^s(\mathbf{s}^s))$ and $\bar{\mathbf{a}}^t = \tilde{G}^t(\bar{\mathbf{s}}^t, G^s(\mathbf{s}^s, \mathbf{a}^s))$. With another discriminator $D^t : \mathcal{S}^t \times \mathcal{A}^t \rightarrow [0,1]$ which distinguishes real target pairs $(\mathbf{s}^t, \mathbf{a}^t)$ from fake ones $(\bar{\mathbf{s}}^t, \bar{\mathbf{a}}^t)$, the adversarial objective in the target space is:

$$\max_{D^t} \min_{\tilde{F}^t, \tilde{G}^t} \mathcal{L}^t_{GAN}(\tilde{F}^t, \tilde{G}^t, D^t) = \mathbb{E}_{(\mathbf{s}^t, \mathbf{a}^t) \sim \mathcal{D}^t} \left[\log D^t(\mathbf{s}^t, \mathbf{a}^t)\right] + \mathbb{E}_{(\mathbf{s}^s, \mathbf{a}^s) \sim \mathcal{D}^s} \left[\log(1 - D^t(\bar{\mathbf{s}}^t, \bar{\mathbf{a}}^t)\right] \tag{4.16}$$

Similarly, we can construct a source domain discriminator D^s which distinguishes the translated target distribution in the source domain:

$$\max_{D^s} \min_{F_t, G_t} \mathcal{L}^s_{GAN}(F_t, G_t, D^s) = \mathbb{E}_{(\mathbf{s}^s, \mathbf{a}^s) \sim \mathcal{D}^s} [\log D^s(\mathbf{s}^s, \mathbf{a}^s)] + \mathbb{E}_{(\mathbf{s}^t, \mathbf{a}^t) \sim \mathcal{D}^t} [\log(1 - D^s(\bar{\mathbf{s}}^s, \bar{\mathbf{a}}^s)]. \tag{4.17}$$

where $\bar{\mathbf{s}}^s = \tilde{F}^s(F^t(\mathbf{s}^t)$ and $\bar{\mathbf{a}}^s = \tilde{G}^s(\bar{\mathbf{s}}^s, G^t(\mathbf{s}^t, \mathbf{a}^t))$. Combining the adversarial objectives in the latent, source and target domains, the full adversarial objective is

$$\max_{D^z, D^s, D^t} \min_{F^t, \tilde{F}^t, G^t, \tilde{G}^t} \mathcal{L}^z_{GAN} + \mathcal{L}^s_{GAN} + \mathcal{L}^t_{GAN}. \tag{4.18}$$

Cycle Consistency Loss. Inspired by CycleGAN [189], we construct a cycle consistency loss such that the state-action translation from one domain to the other and back to its own domain should recover itself. The cycle consistency constraint leverages unpaired samples since it only requires samples from one domain, obviating the need of paired samples from both domains. Specifically, if we have a translated target state $\bar{\mathbf{s}}^t = \tilde{F}^t(F^s(\mathbf{s}^s))$ from source state, the reconstructed source state from it should be close to itself, i.e., $\bar{\bar{\mathbf{s}}}^s = \tilde{F}^s(F^t(\bar{\mathbf{s}}^t)) \approx \mathbf{s}^s$. The cycle consistency objective also applies to translated actions, i.e., $\bar{\bar{\mathbf{a}}}^s = \tilde{G}^s(\tilde{F}^s(F^t(\bar{\mathbf{s}}^t)), G^t(\bar{\mathbf{s}}^t, \bar{\mathbf{a}}^t)) \approx \mathbf{a}^s$. The full cycle consistency objective for both domains is

$$\begin{aligned} &\mathcal{L}_{cyc}(F^t, \tilde{F}^t, G^t, \tilde{G}^t) = \\ &\quad \mathbb{E}_{(\mathbf{s}^s, \mathbf{a}^s)\sim\mathcal{D}^s}\left[\|\bar{\bar{\mathbf{s}}}^s - \mathbf{s}^s\|_1 + \|\bar{\bar{\mathbf{a}}}^s - \mathbf{a}^s\|_1\right] + \mathbb{E}_{(\mathbf{s}^t, \mathbf{a}^t)\sim\mathcal{D}^t}\left[\left\|\bar{\bar{\mathbf{s}}}^t - \mathbf{s}^t\right\|_1 + \left\|\bar{\bar{\mathbf{a}}}^t - \mathbf{a}^t\right\|_1\right]. \end{aligned} \tag{4.19}$$

Dynamics Consistency Loss. Finally, we introduce dynamics consistency constraint in the latent domain to match the joint distribution $P(F_s(\mathbf{s}^s_k), G_s(\mathbf{a}^s_k), F_s(\mathbf{s}^s_{k+1}))$ and $P(F_t(\mathbf{s}^t_k), G_t(\mathbf{a}^t_k), F_s(\mathbf{s}^t_{k+1}))$. Recall that in the previous section, we have trained an RL policy with additional dynamics consistency constraint in the latent space. During target domain alignment, the dynamics consistency can be applied to the target samples as well. We train the target encoders F^t, G^t such that the target latent state-action pairs follow the forward and inverse dynamics functions $T^z, \tilde{T}^z$ trained in Sec. 4.2.2:

$$\mathcal{L}_{dyn,t}(F_t, G_t) = \mathbb{E}_{(\mathbf{s}^t_k, \mathbf{a}^t_k, \mathbf{s}^t_{k+1})\sim D^t}\left[\left\|T^z(\mathbf{s}^z_k, \mathbf{a}^z_k) - \mathbf{s}^z_{k+1}\right\|_2^2 + \left\|\tilde{T}^z(\mathbf{s}^z_k, \mathbf{s}^z_{k+1}) - \mathbf{a}^z_k\right\|_2^2\right], \tag{4.20}$$

Algorithm 5. Source domain policy learning

```
1: Initialize replay buffer $\mathcal{D}$ with source samples $\mathcal{D}^s$.
2: loop
3:     Select action $\mathbf{a} = \tilde{G}_s(\mathbf{s}^s, \pi^z(F^s(\mathbf{s}^s)) + \varepsilon$ with exploration noise $\varepsilon \sim \mathcal{N}(0, \sigma)$
4:     Observe reward $r$, next state $\mathbf{s}'$ and store $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ in $\mathcal{D}$
5:     Sample a minibatch $\{(\mathbf{s}^s_k, \mathbf{a}^s_k, r^s_k, \mathbf{s}^s_{k+1})\}$ from $\mathcal{D}$
6:     Update $\tilde{G}^s, \pi^z, F^s$ with deterministic policy gradient in (4.12)
7:     Update latent dynamics $T^z, \tilde{T}^z$ and encoders $F^s, G^s$ with latent dynamics loss in (4.13)
8:     Update encoders $F^s, G^s$ and decoders $\tilde{F}^s, \tilde{G}^s$ with reconstruction loss in (4.14)
9: Output: Source encoders and decoders $F^s, \tilde{F}^s, G^s, \tilde{G}^s$, latent dynamics $T^z, \tilde{T}^z$, latent policy $\pi^z$.
```

where $\mathbf{s}^z_k = F^t(\mathbf{s}^t_k), \mathbf{a}^z_k = G^t(\mathbf{s}^t_k, \mathbf{a}^t_k)$. Note that the forward and inverse dynamics functions $T^z, \tilde{T}^z$ are fixed during this latent alignment stage since the latent distribution from source $P(F_s(\mathbf{s}^s_k), G_s(\mathbf{a}^s_k), F_s(\mathbf{s}^s_{k+1}))$ is already optimized in Sec. 4.2.2. Here we are only aligning the latent joint distribution $P(F_t(\mathbf{s}^t_k), G_t(\mathbf{a}^t_k), F_s(\mathbf{s}^t_{k+1}))$ from target towards that from source.

After training the target domain encoders and decoders $F^t, \tilde{F}^t, G^t, \tilde{G}^t$ with the above alignment objectives, we have aligned the source and target samples $\mathcal{D}^s, \mathcal{D}^t$ in the common latent space $\mathcal{M}^z$. Finally, we deploy the target policy from the source policy by replacing the corresponding state encoder and action decoder, i.e., $\pi^t(\mathbf{s}^t) = \tilde{G}^t(\mathbf{s}^t, \pi^z(F^t(\mathbf{s}^t)))$.

Algorithm

Alg. 5 summarizes the reinforcement training procedure for learning source encoders and decoders $F^s, \tilde{F}^s, G^s, \tilde{G}^s$. Alg. 6 summarizes the procedure for learning a latent representation between source and target domains with cycle and dynamics consistency. During deployment in the target domain, only the latent policy π^z from Sec. 4.2.2 remains while the state encoder and action decoder are replaced with F^t and $\tilde{G}^t$ trained in Sec. 4.2.2. Our proposed approach does not depend on paired source and target data, nor does it require reward supervision for test time adaptation.

Algorithm 6. Latent representation alignment for target domain deployment

```
1: Freeze learned models $F^s, \tilde{F}^s, G^s, \tilde{G}^s, T^z, \tilde{T}^z, \pi^z$ from Alg. 5
2: for step in $1, \dots,$ target domain alignment steps do
3:     Sample minibatches $\{(\mathbf{s}^s_k, \mathbf{a}^s_k, \mathbf{s}^s_{k+1})\} \sim D^s$ and $\{(\mathbf{s}^t_k, \mathbf{a}^t_k, \mathbf{s}^t_{k+1})\} \sim D^t$
4:     Update discriminators $D^z, D^s, D^t$ by maximizing (4.18)
5:     Update target encoders and decoders $F^t, \tilde{F}^t, G^t, \tilde{G}^t$ by minimizing (4.18), (4.19), (4.20)
6: Output: Target policy: $\pi^t(\mathbf{s}^t) = \tilde{G}^t(\mathbf{s}^t, \pi^z(F^t(\mathbf{s}^t)))$
```

4.2.3 Experiments

Simulation Results

We first verify our approach and conduct ablation studies in the robosuite simulation environment [191]. We consider four tasks: Reach, Lift, PickPlace and Stack. Environment details are further described in Sec. 4.2.4.

Reach Task and Ablation Studies. In the first experiment, we will test the target state alignment by only using the Panda robot in both domains. Specifically, we will only learn the target state encoder-decoder $F^t, \tilde{F}^t$ while setting the other components to be identity, i.e., $F^s, F^s, G^s, \tilde{G}^s, G^t, \tilde{G}^t$ are identity functions. In $\mathcal{M}^s$, a Panda robot tries to reach a target position with end-effector position control given its end-effector position while in $\mathcal{M}^t$, the same Panda robot uses joint positions as state input for end-effector position control. Under such setting, the target state encoder-decoder $F^t, \tilde{F}^t$ should theoretically approximate the forward and inverse kinematics functions between joint angles and end-effector position. For evaluation metrics, we first compute the ℓ_1-distance between the predicted and ground truth end-effector positions of the target robot to assess state alignment. Next, we evaluate the RL performance as another metric where we train a source policy π^s (in this case $\pi^s = \pi^z$ since F^s and $\tilde{G}^s$ are identity) and apply the target policy $\pi^t = \pi^z \circ F^t$ (since $\tilde{G}^t$ is also identity). Since there is no projection from the source domain to the latent, we simply apply TD3 [55] to obtain the source domain policy.

We compare with the following baselines: (i) ILA [180] which performs state alignment in source domain and does not require cycle consistency; (ii) our model without latent dynamics constraint; (iii) a strongly-supervised model that is trained on paired joint to end-effector data; (iii) an oracle which is an RL policy trained in the target domain and provides an upper bound;

Table 4.4. Transfer learning evaluation on Reach task with Panda end-effector space source and Panda joint space target. The results are averaged over 5 runs. Lower is better for the ℓ_1 error (cm), and higher is better for the RL score. Our model performs better than the baseline ILA [180] which does not consider cycle consistency constraint or then baseline without latent dynamics constraint. The strong supervision model performs better than ours but requires paired training data. The oracle is an RL policy trained in the target domain.

	Ours	Ours w/o dyn.	ILA [180]	Strong supervision	Oracle	Random
ℓ_1 error	0.7 ± 0.1	4.7 ± 0.2	3.9 ± 0.3	0.3 ± 0.1	-	-
RL score	163 ± 10	41 ± 27	53 ± 27	166 ± 7	172 ± 11	13 ± 8

(iv) a random policy in the target domain. The results in Table 4.4 show that the cycle consistency and dynamics consistency are important terms that improve the performance of our model over ILA [180]. The strongly supervised model is marginally better ours even though it uses paired source and target data while our model only uses unpaired data.

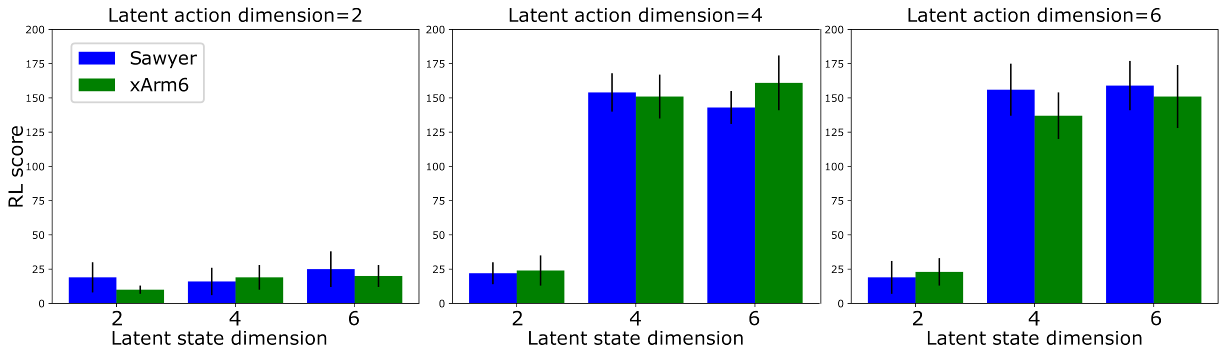


Figure 4.9. Ablation study on latent state and action dimensions for transfer from Panda to Sawyer and xArm6 robots. The lowest state and action dimensions with reasonable performance are 4.

Next, we use joint velocity control for both the source and target domains and ablate the latent state and action dimensions. The Panda and Sawyer have 7 degrees-of-freedom (DoF) arms while the xArm6 is 6 DoF. Hence, the robot state dimensions, represented in sine and cosine functions of the joint angles, are 14 for Panda and Sawyer and 12 for xArm6. The action dimension for each robot is its DoF when we apply joint velocity control. Fig. 4.9 shows the RL performance on the transferred policy from Panda to Sawyer and xArm for the Reach task with different latent state $\mathbf{s}^z$ and action $\mathbf{a}^z$ dimensions. We find that the transfer performance

Table 4.5. RL reward for a source policy trained on Panda and transferred to Sawyer and xArm6 robots. The reward of an oracle policy trained directly on the target robot is shown in parenthesis.

Task	Lift	PickPlace	Stack
Sawyer	122 ± 41 (181 ± 4)	70 ± 35 (86 ± 15)	85 ± 40 (121 ± 18)
xArm6	132 ± 23 (171 ± 7)	74 ± 36 (84 ± 9)	78 ± 49 (116 ± 23)

drops significantly when the state or action dimensions are too small. This makes sense as the end-effector position control takes place in 3-dimensional space and latent trajectories in a lower dimensional space cannot recover the 3D motions for Reach task. In the subsequent experiments, we choose latent state and action dimensions to be both 4 since it is the smallest dimension that still achieves a good performance.

Lift, PickPlace, Stack. In this section, we use joint velocity controller to perform Lift, PickPlace and Stack tasks in robosuite simulation. We train the source domain policy with reinforcement learning on the Panda robot and transfer to Sawyer and xArm6 robots. The state variable includes robot joint angles, gripper width, gripper touch signal and object and target positions. Detailed experiment settings are described in Appendix 4.2.4. Fig. 4.10 shows a Lift task example of the transferred policy from Panda robot to Sawyer and xArm robots. Table 4.5 shows that quantitatively our model can learn a meaningful mapping between robots of different embodiments. However, manipulation tasks require very precise alignment in order to correctly grasp objects and sometimes the transferred policy cannot complete the task successfully.

Real Robot Experiments

In this section, we evaluate our model's capability for sim-to-real skill transfer. The physical setup of the robot and experiment task is shown in Fig. 4.11. We train a source policy with the Panda robot in simulation and instead of transferring to a simulated Sawyer or xArm robot, we directly transfer to a real xArm6 robot. Note that the robot to train a policy in simulation is *not* the same type of robot that is used in real world deployment. The source policy on the simulated Panda is trained with behavioral cloning on 100 human demonstrated trajectories in each task in order to avoid jerky motions when we transfer on the real robot for safety concerns.

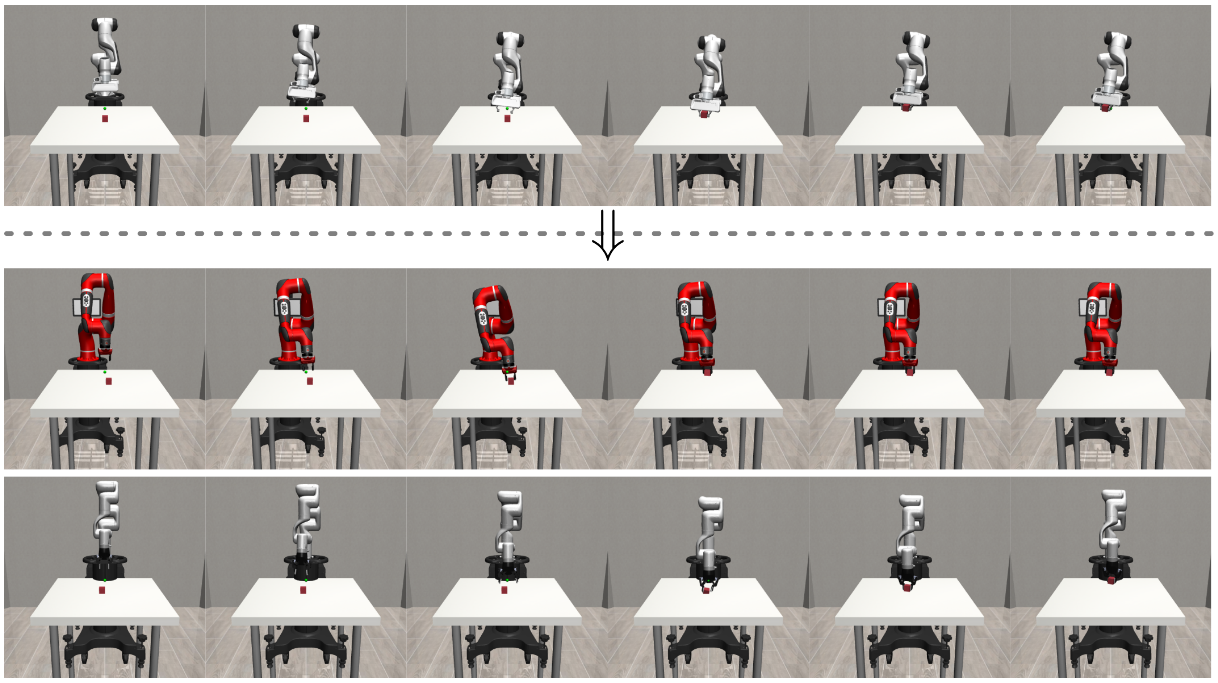


Figure 4.10. Examples of transferring Panda robot policy (top row) to Sawyer robot (middle row) and xArm6 robot (bottom row) for the Lift task in robosuite. We learn encoders to map the source Panda robot states and actions into a latent space and simultaneously a latent policy to solve the task. The latent space can be used to align different types of target robots (Sawyer or xArm6) and successfully transfer the learned policy without finetuning in the target domains using their reward functions or additional expert demonstrations.

Since we do not have ground truth object information in real experiments as we did in the simulated experiments, we use an RGBD camera to estimate object position in real time. Details about object tracking are described in Sec. 4.2.4. For evaluation metrics, we count the success rate for each task over 10 test episodes. The success conditions are defined in Sec. 4.2.4 for each task. While simulation environments are quite tolerant to collisions, it would cause protection responses on the real robot arm and are counted as failures.

We transfer the source policy trained on a simulated Panda robot to a real xArm6 robot

Table 4.6. Transfer results of 10 episodes from simulated Panda to real xArm6 with joint velocity control. Our model can successfully transfer from sim to real. The success rate on the real robot is lower due to dropped objects or collisions that cause a safety stop.

Task	Success	Collision	Drop
Lift	70%	20%	10%
PickPlace	60%	20%	20%

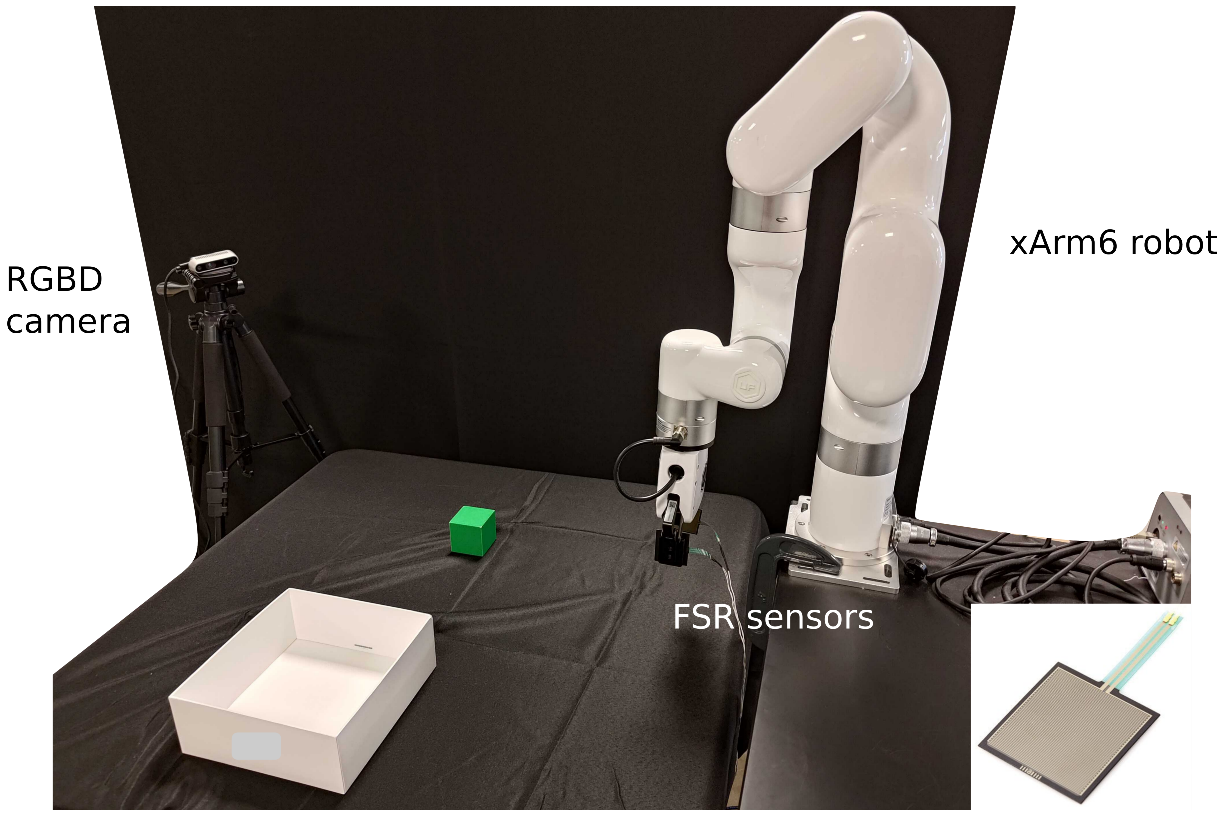


Figure 4.11. Real-world experiment setup. We use the xArm6 robot with an xArm gripper. Force sensing resistors (FSR) are attached on grippers (enlarged on bottom right) to obtain pressure signals when objects are grasped. We use an RGBD camera to estimate object positions.

with reasonable success rates as shown in Table 4.6. Fig. 4.12 shows a successful trajectory of the transferred policy of the PickPlace task while Fig. 4.13 shows some failure modes. When the alignment is not perfect, the gripper would collide with the cube or the table. We also observe that the gripper can sometimes drop the cube prematurely before it reaches the target. The robot is not able to re-grasp the dropped cube if its landing position is outside that of training distribution.

4.2.4 Implementation Details

Experiment Settings

We use Reach, Lift, and PickPlace environments from the robosuite simulator [191] as shown in Fig. 4.14. The custom Reach task contains only the robot manipulator and it tries to reach a target position in 3D space with its end-effector. In the Lift task, the robot tries to lift

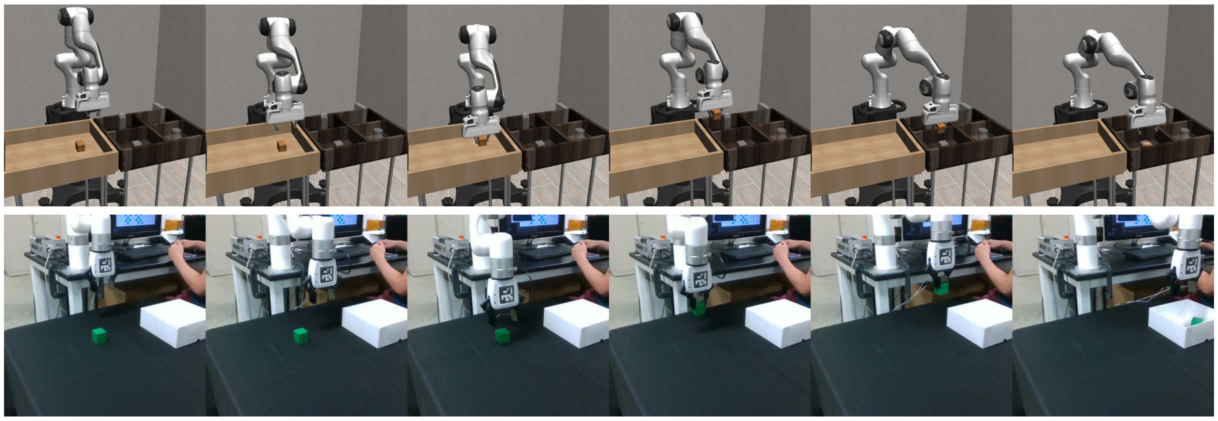


Figure 4.12. Simulation to real transfer for PickPlace task. The source policy is trained with behavior cloning in simulation with Panda robot (top row) and transferred to a real xArm6 robot (bottom row).

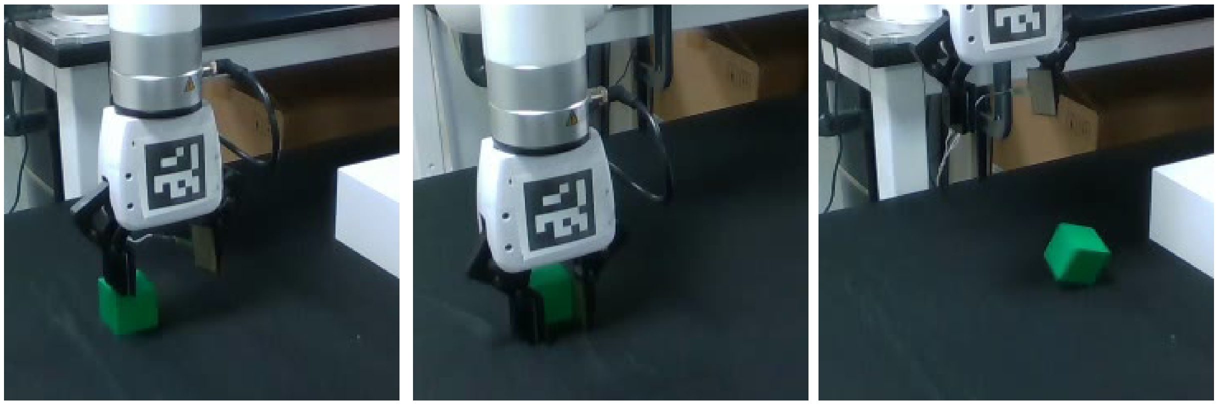


Figure 4.13. Examples of failure modes including cube collisions, table collisions, or permanently dropping the cube.

the cube to a target position above the table center. In the PickPlace task, the robot picks up a bread object from one side and places it in the bin on the other side. In the Stack task, the robot stacks the red cube on top of the green cube. The state space consists of robot related states, including sine and cosine functions of joint angles or the gripper end-effector position and gripper open width, and object related states, including object position and target position. The action space consists of desired joint velocities for joint velocity control or delta position values for end-effector position control, and gripper open or close control. For the Lift, PickPlace and Stack tasks, we also include touch sensors from the robot grippers. In simulation, the touch sensor signal is obtained from checking collisions between gripper finger and objects. In real

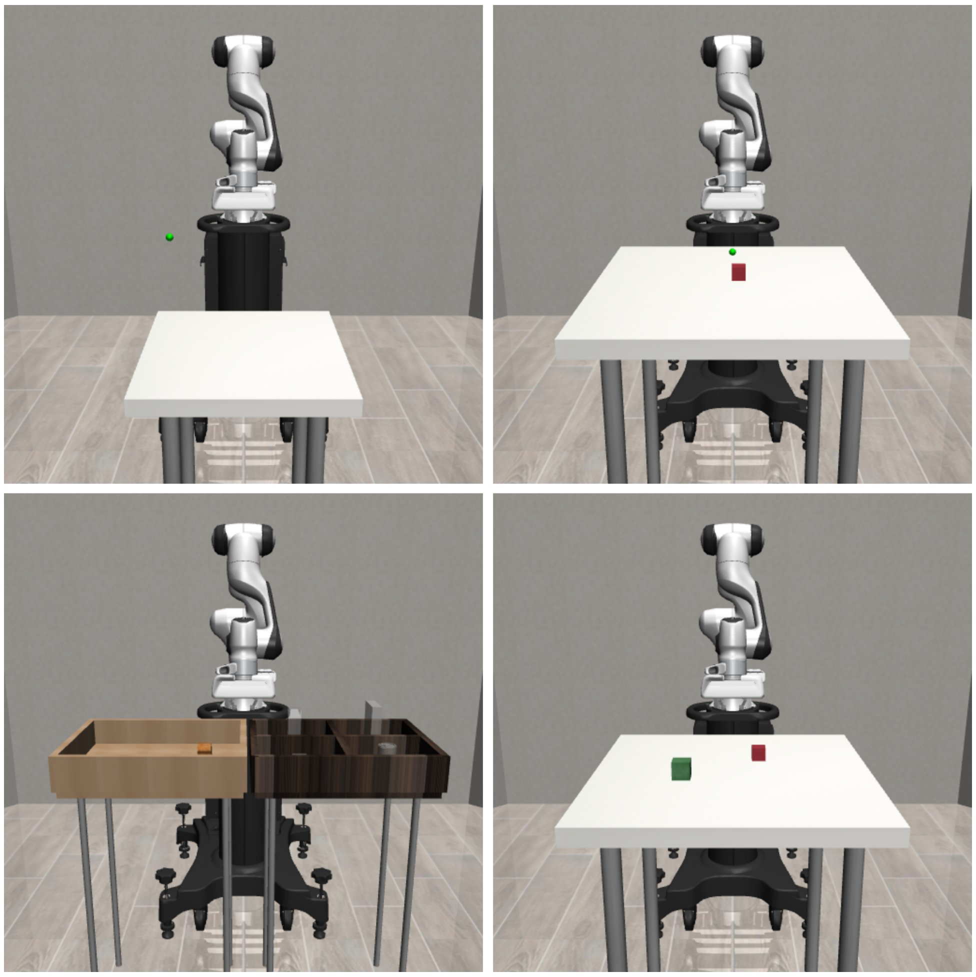


Figure 4.14. Robosuite simulation environment tasks (from top to bottom, left to right): Reach, Lift, PickPlace and Stack.

experiments, this is obtained from force sensing resistors attached on the xArm gripper.

Model Architecture

The state and action encoder and decoder functions $F_{s,t}, F_{s,t}^{-1}, G_{s,t}, G_{s,t}^{-1}$, latent dynamics functions H_{fwd}, H_{inv}, and discriminators $D^{s,t,z}$ are neural network layers with hidden layers of size $[256, 256, 256]$. We use ReLU activations for all hidden layers and hyperbolic tangent function for the final output layer for all models except for discriminators where leaky ReLU activations are used for hidden layers. All models are trained with Adam optimizer using decay rates $\beta_1 = 0.9, \beta_2 = 0.999$.

Object Tracking

Real-time object tracking is performed with an Intel RealSense D435 stereo camera. We use a simple color detector on the RGB image to find the pixel location of the object since the object has a high color contrast from its background. The color range for green cube in HSV space is from $[30, 80, 50]$ to $[90, 255, 255]$. Next, the 3D position of that pixel is obtained from querying the corresponding depth value on the depth image, where post-processing filters including disparity, spatial and temporal, are applied to reduce depth noise. Finally, we get the 3D object position in robot frame from image frame with calibrated camera extrinsics parameters which are obtained from hand-eye calibration using ArUco markers.

Dataset Collection

We collect random trajectories for Panda, Sawyer and xArm6 robots ($\mathcal{D}^s$ and $\mathcal{D}^t$) in the robosuite simulation for the state-action alignment. For each robot type, the robot base is placed such that its gripper initial position is at $[-0.2, 0, 1.05]$. In each episode, the robot gripper moves in straight line to a randomly sampled target position in a 3D rectangular region bounded by $[-0.2, 0.2]$ in x, $[-0.25, 0.25]$ in y and $[-0.8, 1.2]$ in z. It continues to move to a newly sampled target without resetting at the end of the episode unless the gripper goes out of bounds. We collect 10000 episodes of length 200 for each robot. We find that this sampling strategy covers the robot workspace better than randomly sampling actions at each step. With random actions at each step, the robot trajectory often result in either self collision or purposeless directions above or even behind the base.

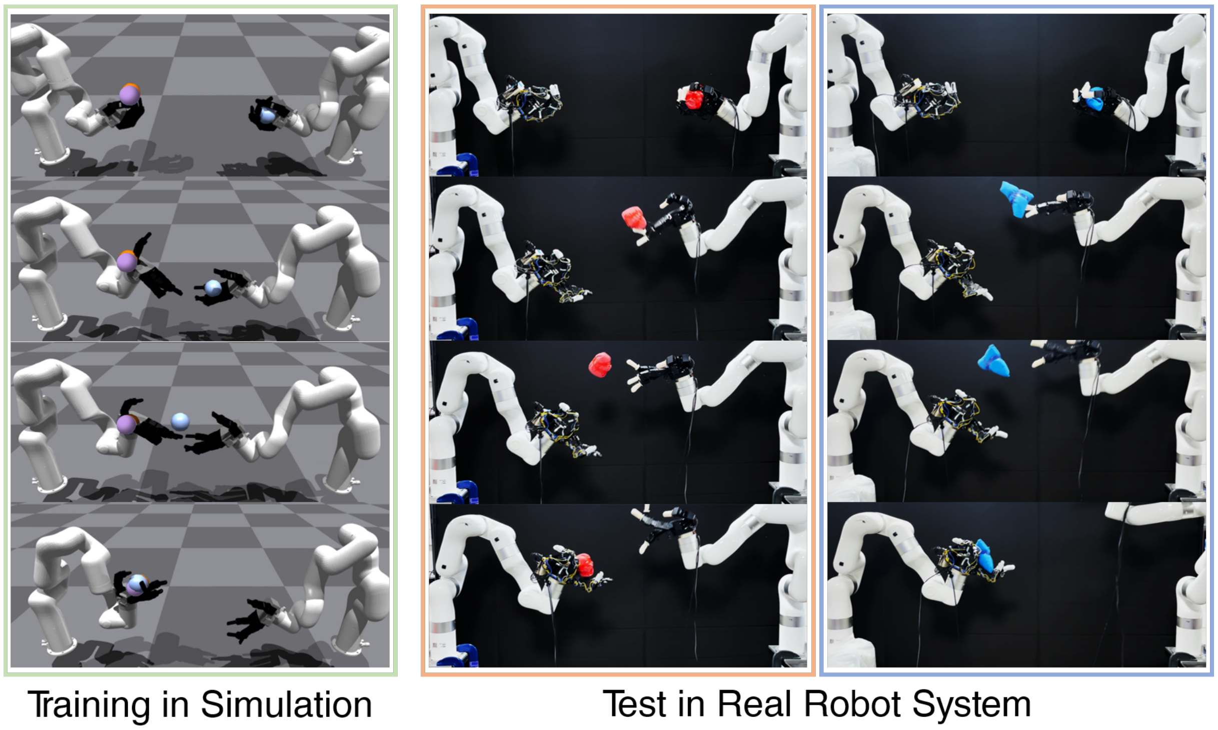


Figure 4.15. We propose **Dynamic Handover**, a new bimanual dexterous hands system designed for throwing and catching tasks. The system consists of two Allegro Hands, each individually attached to a separate XArm robot, arranged in a facing configuration. Using multi-agent reinforcement learning, we train policies in a simulation environment and subsequently transfer them to the real world.

4.3 Dynamic Handover: Throw and Catch with Bimanual Hands

4.3.1 System Setup

Task Description. We focus on the bimanual Catching and Throwing task with two dexterous robot hand. This task involves two robot agents: (i) a thrower robot agent (Figure 4.16 right) that needs to execute swift movements to toss the grasped object towards the other side, and (ii) a catcher agent (Figure 4.16 left) that needs to react dynamically to catch the airborne object. This task is important because it enables the catcher robot to access objects beyond its kinematic range by leveraging the object's momentum imparted by the thrower. It also serves as a good test-bed for evaluating the coordination and performance of bimanual systems in high-speed, real-time scenarios.

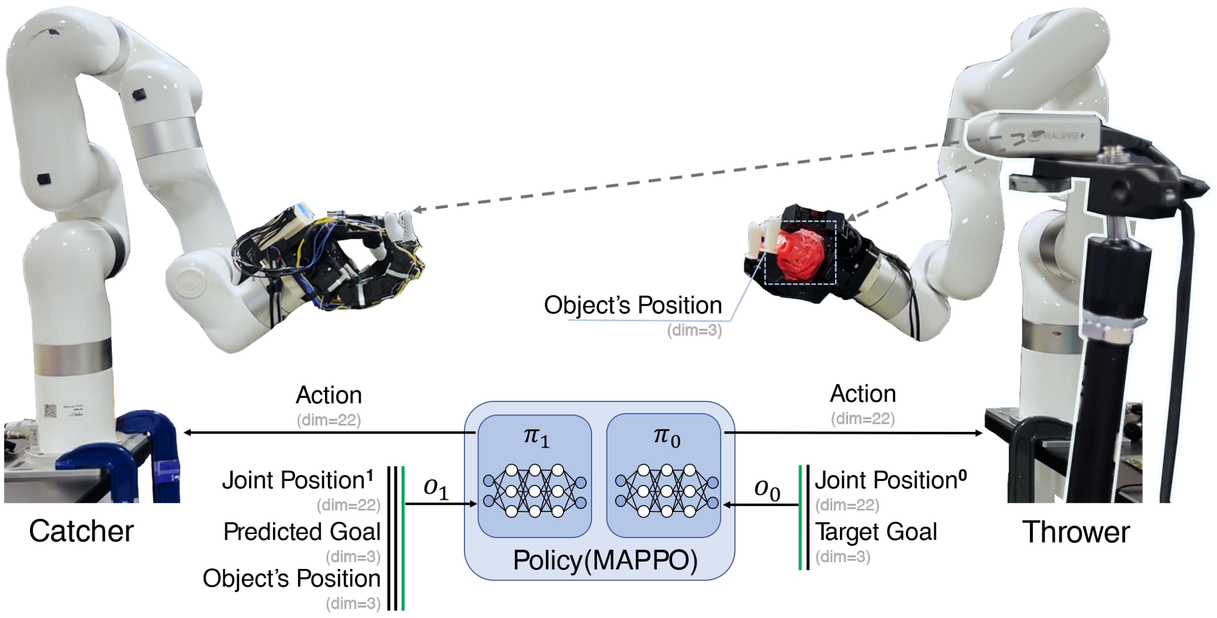


Figure 4.16. Real Robot System: We employ two Allegro Hands, each individually mounted on separate XArm-6 robots, arranged in a face-to-face configuration. We incorporate a RealSense D435 camera for real-time object position tracking, which is oriented towards the working space. We use k prior states in observation.

Real World Setup. We construct a bimanual system for executing our throwing and catching task, as depicted in Figure 4.16. The system includes two arm-hand subsystems and a RealSense D435 camera. Each arm-hand subsystem features a 6-DoF XArm-6 robot arm paired with a 16-DoF Allegro Hand, culminating in a 44-DoF system. To create a closed-loop policy, we use a RealSense camera to capture the real-time position of objects within the catcher robot's frame of reference. For observation, distinct feedback mechanisms are provided for each agent, as depicted at the bottom of Figure 4.16. The thrower (Figure 4.16: right) depends exclusively on its own proprioceptive data, while the catcher (Figure 4.16: left) obtains feedback from not only its own proprioception but also the object's real-time positions estimated by the camera. In other words, the thrower operates based on its current state, whereas the catcher necessitates both proprioceptive and visual input to dynamically and interactively perform catching actions. Further information regarding this system's implementation can be found in the Appendix.

Simulation Setup. In this work, we use the IsaacGym physical simulator [113] for training our throwing and catching task. The simulation setup is shown on the left side of

Figure 4.15. The simulation frequency is set at 120Hz while the control frequency is 20Hz. We train the end-to-end reinforcement learning policy in the simulated environment and then transfer the policy to the real world. Further information regarding the details about the simulation setup (e.g. policy architecture) can be found in the Appendix.

Action Space. The policy outputs a 22-dimensional PD control target, with the first six dimensions corresponding to XArm-6 and the remaining 16 dimensions corresponding to Allegro hand. For the XArm-6, we employ delta joint positions as the control target, while for the Allegro hand, we utilize absolute joint positions as the control target. This design choice is made to avoid jerky motion of robot arm for safety reasons, while still allowing the hand to swiftly react and release its grasp to throw the object. In our experiments, we find that controlling only the second and third joints of the robot arm and keeping the other four joints fixed results in a more effective and safer policy. Therefore, the action space consists of an 18-dimensional target for the thrower and a 22-dimensional target for the catcher.

4.3.2 Learning Bimanual Dexterous Hands Policy

Catching an object in mid-air poses significant difficulties due to the high-speed requirement. First, object's real-time velocity and anticipated trajectory must be taken into account in order for the catcher to determine its movement. Second, even though the thrower can consistently toss the object toward the pre-defined target goal in simulated environment, the policy transfer to the real world is imperfect due to the substantial dynamics gap between simulation and real physical. Consequently, there is a discrepancy between the pre-defined throwing goal and the object's actual destination. In light of these two challenges, a goal estimator becomes crucial for predicting the thrower's actual destination instead of the predetermined target goal. This allows the catcher to move based on the forecasted object destination and successfully catch it.

To achieve this, we introduce a novel three-stage training pipeline for learning bimanual throwing and catching. (i) In the first stage, we train a base policy using Multi-Agent RL to

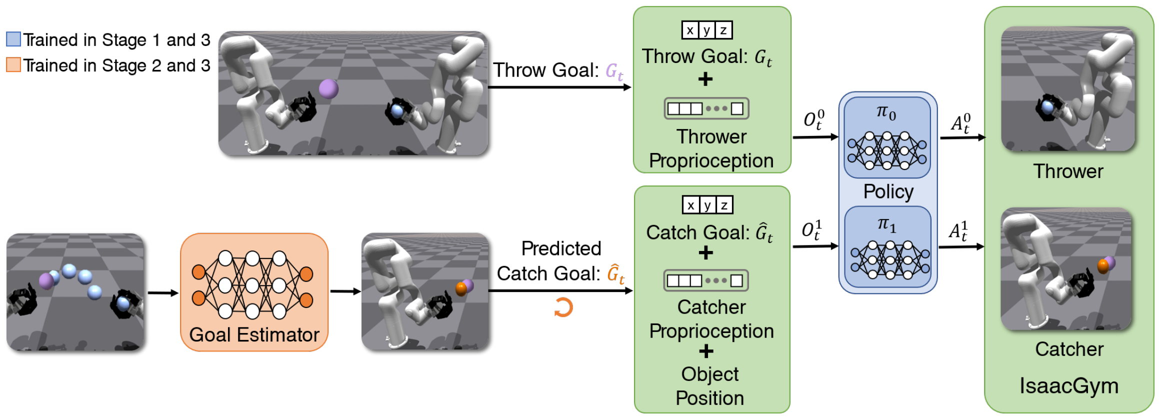


Figure 4.17. Joint End2End Learning: The two agents receive input from both their own observations and the catcher agent additionally receives the predicted catching position. The goal estimator takes past 20 frames of the object's positions as input and predicts the catch goal for each time step. We use a violet ball to represent the pre-defined goal for the throwing. The orange ball represents the predicted goal for the catcher to catch during the throwing task. The blue ball represents the object that is currently been manipulated.

tackle the task, with the catcher observing the pre-defined throwing goal. The policy trained during this stage is expected to perform well within the simulator but may hardly transfer to the real world. (ii) Next, we freeze the base policy and train a goal estimator through supervised learning, using the rollout trajectory of the base policy as training data. (iii) Finally, we replace the pre-defined throwing goal in the catcher's observation with the estimated goal and unfreeze the policy for fine-tuning both the base policy and the goal estimator in an end-to-end fashion. The refined policy is expected to bridge the dynamics gap between simulation and reality with the predicted object's future trajectory.

Stage 1: Multi-Agent Reinforcement Learning

We employ the Multi-Agent Proximal Policy Optimization (MAPPO) [181] in a non-parameter sharing way to train the thrower and the catcher to obtain basic policies in the first stage. MAPPO is an application of the PPO algorithm to multi-agent settings. It leverages centralized training with decentralized execution, allowing each robot agents to efficiently accomplish the cooperative task using partial observations.

As shown at the bottom of Figure 4.16, the observations for the thrower and catcher in

MAPPO training is not identical. The thrower's policy, denoted as π_0, receives its proprioception and a pre-defined target position for throwing. In contrast, the catcher's policy, denoted as π_1, takes as input its proprioception, the pre-defined goal position and the current position of the object for catching. To satisfy MAPPO's requirement for equal input dimensions across agents, we pad zeros to the thrower's input for dimension alignment. Besides, we include observations from past k frames as input for both policies to provide temporal information. In our implementation, we set $k = 2$.

Given the object position $\boldsymbol{p}_t$, target goal position $\mathcal{G}_t$, the velocity of the object $\boldsymbol{v}$, the unit direction vector from thrower to catcher $\hat{\boldsymbol{u}}$, and robot joint torque $\boldsymbol{\tau}$, we design the reward function using three components: (i) distance between object and throwing goal; (ii) object velocity projected in the direction from thrower to catcher; (iii) robot joint torque. The final reward r can be computed as $r = r_{dis} + r_{linvel} + r_{torque}$, where $r_{dis} = exp(-20 * (\boldsymbol{p}_t - \mathcal{G}_t))$ represents the distance, $r_{linvel} = clamp(\boldsymbol{v} \cdot \hat{\boldsymbol{u}}, -0.1, 0.1)$ denotes the object's velocity towards the catcher, and $r_{torque} = -0.003 * \|\boldsymbol{\tau}\|_2^2$ corresponds to the torque penalty.

Stage 2: Goal Estimator Learning

After training the basic policies, the next step involves freezing the basic policies and training a goal estimator (Orange Block in Figure 4.17) to predict the goal for the catcher based on the trajectory of the object. This is a crucial step due to the sim2real gap, which implies that although the thrower may consistently hit the goal in simulation, it is unlikely to achieve the same level of accuracy in the real world. Therefore, predicting the actual goal based on the object's trajectory becomes essential for improving sim-to-real transfer. In this stage, we utilize the historical positions of the object over a span of k frames as input to the goal estimator. The output of the goal estimator is the predicted 3D position of the goal, which provides crucial information for the catcher to anticipate the intended catching point and enhance the coordination between the two hands. We use Adam to optimize the L2 distance between the position of the predicted goal and the thrower's goal until convergence:

$$\mathcal{L}(\omega) = \|\omega(\mathbf{p}^1_{t-k:t}) - \mathcal{G}^0_t\|^2 \tag{4.21}$$

where $\mathbf{p}^1_{t-k:t}$ is the position of the object from $t-k$ to t frames, $\mathcal{G}^0_t$ is the thrower's goal, and the 0 and 1 represent the thrower and catcher respectively. It is important to highlight that in the previous training stage, the object's landing point was primarily influenced by the thrower's goal. This is because the thrower, operating in the simulation environment, had the ability to consistently hit the specified goal in the simulation.

Stage 3: End2End Joint Learning

In stage 1, the catcher's base policy uses a pre-defined throw goal in its observation. In this stage, we replace the pre-defined goal with the predicted one from the goal estimator. However, the distribution shift brought by this replacement can result in compounding errors. To address this issue, we jointly fine-tune the goal estimator and the policy network in this stage, as visualized in Figure 4.17, allowing the catcher to adapt to the goal estimator. For example, when the goal estimator is inaccurate, the policy will not depend solely on it for decision-making. This joint training approach helps reduce compounding errors when integrating the goal estimator with the policy.

4.3.3 Experiments

Evaluation Criterion. To evaluate the performance of the trained policies for throwing and catching, we consider several metrics as follow:

(i) Success Rate(SR): It is calculated as the ratio of successful throws and catches to the total attempts. A better policy will lead to a higher Success Rate.

(ii) Hit Rate(HR): This metric is defined as the proportion of objects that successfully hit the hand palm of catcher. A better policy and goal estimator will lead to a higher Hit Rate.

Training and Dataset. During the training process, we utlize three different objects: a ball, a cube, and a rod, which are three typical geometries for robot manipulation. At the begin-

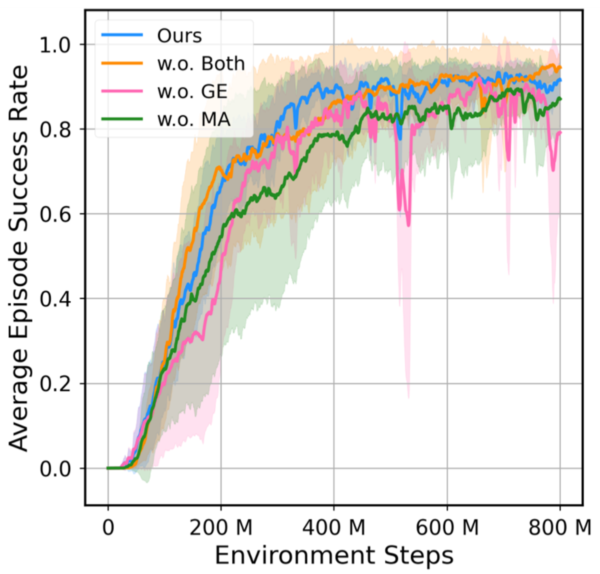


Figure 4.18. Training Curves. The plot shows multi-object training curves of our method and 3 baselines.

ning of each episode, we randomly select one object for training. For simulation experiments, we expand the object set to include additional objects to evaluate the generalizability of our policy to novel objects. The objects used in the simulation experiments are depicted in Figure 4.19(a) and Figure 4.19(b). We chose objects that have similar sizes to the training objects but differ in shape and size to assess the model's generalization capabilities. In the real-world experiments, we employ sandbags in three different shapes for throwing, as shown in Figure 4.19(c): a ball, a cylinder, and a triangle prism.

Baselines. In this section, we compare our method with the following baselines:

(i) Open-Loop Policy: We employ a pre-defined trajectory for the bimanual hand-arm system to execute the throwing and catching task. The trajectories are collected on the real robot using kinesthetic teaching and replayed later without considering feedback or adjustment during execution.

(ii) Without Multi-Agent: We train our policy using PPO instead of using MAPPO. However, we still incorporate goal estimation during the learning process. Under this setup, both agents share the same observation.

(iii) Without Goal Estimation: We restrict our learning process to the first stage in

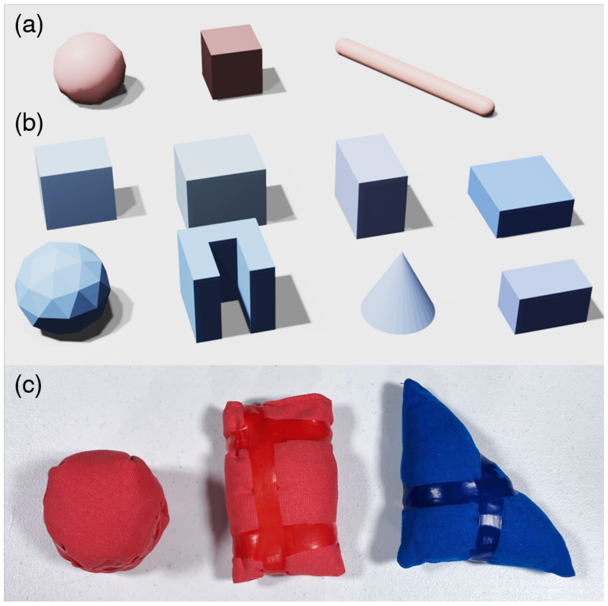


Figure 4.19. Objects Sets. (a) Training objects. (b) Additional objects in evaluation. (c) Real-world objects.

Section 4.3.2 with MAPPO algorithm and evaluate the policies without goal estimation.

(iv) Without Both: We restrict our learning process to the first stage in Section 4.3.2 with PPO algorithm and evaluate the policies without goal estimation.

Table 4.7. Ablation Study in Simulation: Success Rate of throwing and catching task on different objects in simulation. We use **11** trained objects and **14** novel objects. The results are averaged on 5 seeds, each seed has 100 trails.

Settings	**Known Obj.**	**Novel Obj.**
w/o Multi-Agent	0.89 ± 0.07	0.24 ± 0.05
w/o Goal Est.	0.88 ± 0.04	0.22 ± 0.04
w/o Both	0.93 ± 0.07	0.12 ± 0.06
Ours	**0.95 ± 0.07**	**0.37 ± 0.04**

Results in Simulation

We conduct an analysis of our method with three baselines in the simulation environment. Figure 4.18 shows the training curve of four methods. Table 4.7 presents the success rates for two categories: Known Objects and Novel Objects. Our findings can be summarized as follows.

First, for the test on known objects experiment, we observe that the baseline (Without Both) outperforms the other three methods, including ours. One possible reason for this is that the

Table 4.8. Comparison for Pre-throw Conditions: We calculate the standard deviation of landing points on the table in the x and y directions, based on 10 runs for each pose. The units are in meters. Pose A: simply placing the object. Pose B: gripping the object with the robot hand. Pose C: firmly grasping the object.

Settings	**Pose A**	**Pose B**	**Pose C**
Std(x)	0.051	0.072	**0.024**
Std(y)	0.087	0.048	**0.043**

policy without multi-agent coordination has access to full observations of both hands and the arm system, as well as the ground-truth object position. As a result, the policy can easily overfit to a specific point and successfully solve the task. However, it is important to note that this baseline policy may have lower generalization capabilities when it comes to novel objects or uncertain parameters. In the real world, we encounter noise and uncertainties, and obtaining the ground-truth object position is not feasible. Therefore, the policy with full observations demonstrates lower transferability from simulation to the real world, as we validate in Section 4.3.3.

Secondly, in the novel objects experiment, although all methods show a significant drop in performance, our method outperforms the baselines. The utilization of multi-agent reinforcement learning and goal estimation proves beneficial for accomplishing the throwing and catching task. Both the thrower and catcher agents receive observations from their respective perspectives, enabling them to perform their tasks cohesively. Furthermore, goal estimation assists the catcher in predicting the landing point of the objects based on historical positions. This feature helps mitigate the impact of unpredictable parameters during the manipulation process, such as friction, unexpected collisions, and other dynamic factors.

Results in Real World

We perform sim-to-real experiments to assess the performance of our method and two baselines on a real robot platform. As depicted in Figure 4.16, we deploy multi-agent reinforcement learning policies on the real robot agents, with both agents controlled by the same host. The task execution sequence is visualized on the second and third columns in Figure 4.15. Further implementation details and communication methods can be found in the Sec. 4.3.4.

The results of our real world evaluation are presented in Table 4.9. We successfully transfer our multi-agent reinforcement policy to the real robot system with a reasonable success rate after performing system identification to align the PD controllers between the simulation and the real robot. Our method outperforms the baseline methods, indicating the effectiveness of multi-agent reinforcement learning (MARL) and goal estimation in real robot experiments. These components provide benefits in dealing with various unpredictable factors encountered in the real-world setting, leading to improved performance and robustness. We also notice that the success rates achieved in real world experiments are lower than the hit rate. This is primarily attributed to occasional challenges encountered during the grasping phase of the catcher. In some cases, the catcher may fail to firmly grasp the object before it bounces back, leading to the object slipping off the robot's hand.

How Pre-throw Conditions Impact Throwing Performance?

In this experiment, we examine the repeatability of the thrower. Our intuition is that a thrower policy capable of generating consistent object trajectories often results in a higher success rate for the catcher. We find that the robots' and objects' initial positions have a significant impact on the stability of the throwing motion. To address this, we investigate three different initial conditions (depicted in Figure 4.20) in real-world. For each initial position, we train a MARL policy and conduct 10 trials with the thrower robot. We compute the variance of the landing point on the table to evaluate the repeatability under different initial conditions. A smaller variance indicates better repeatability and stability. The results are summarized in Table 4.8. We observe that condition (c) demonstrates the smallest variance and enables more stable throws towards the target compared to conditions (a) and (b). This suggests that an initial firm grasp is advantageous for subsequent throwing behavior.

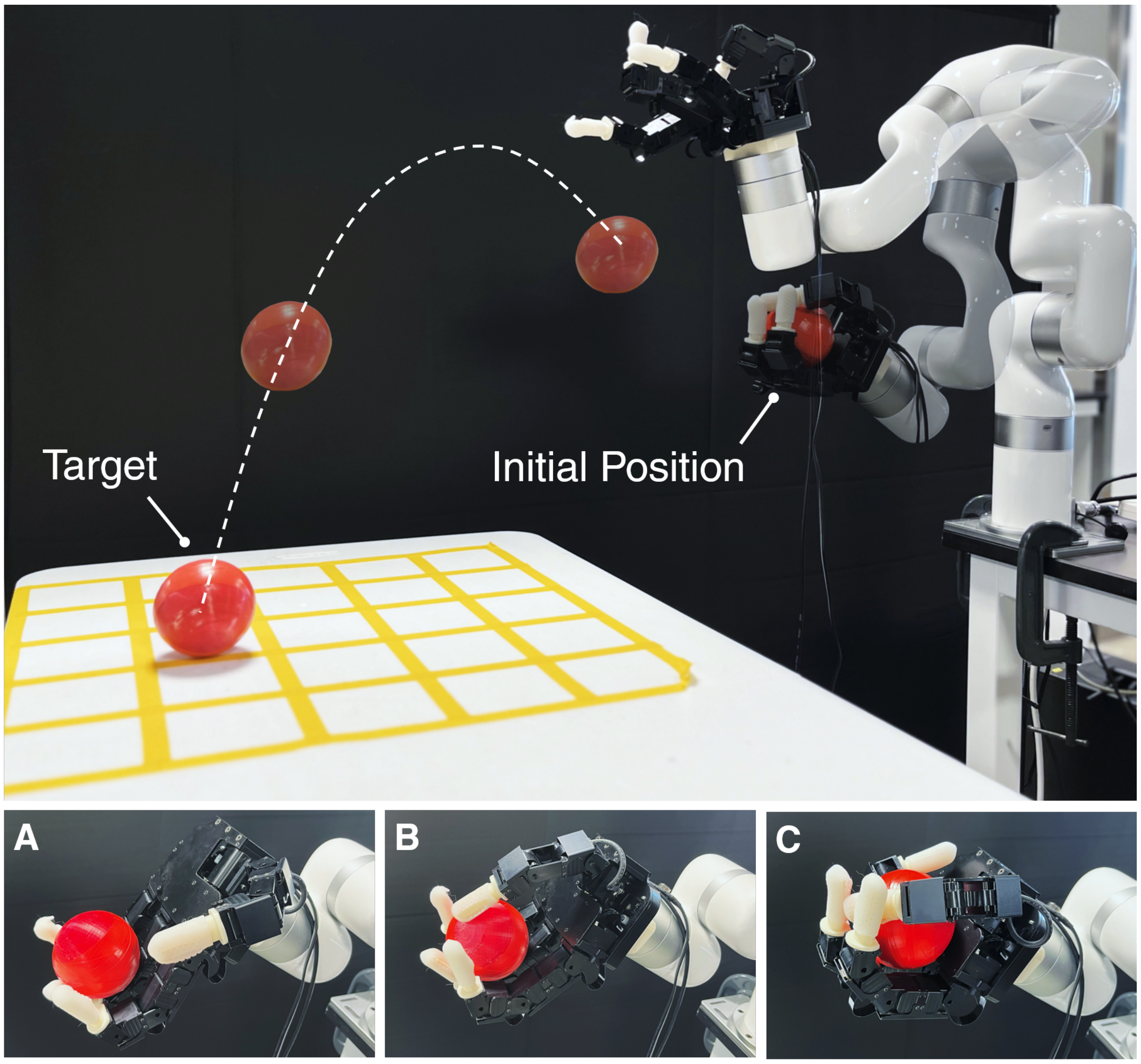


Figure 4.20. Throwing Stability Test of different initial settings: (a) simply placing the object on an open robot hand, (b) gripping the object with the robot hand, resembling a parallel gripper, and (c) firmly grasping the object with the robot hand.

Table 4.9. Ablation Study in Real World: Performance of throwing and catching task on 3 different unknown objects in real robot platform. Objects are made of sandbags with the same mass but different shapes. The results are averaged on 3 seeds with 5 trails for each. The two terms stand for:(i) Hit Rate(HR): This metric is defined as the proportion of objects that successfully hit the hand palm of catcher. A better policy and goal estimator will lead to a higher Hit Rate. (ii) Success Rate(SR): It is calculated as the ratio of successful throws and catches to the total attempts. A better policy will lead to a higher Success Rate.

Settings	Ball		Cylinder		Triangle	
	HR	SR	HR	SR	HR	SR
Open-Loop	0.60 ± 0.12	0.13 ± 0.12	0.47 ± 0.12	0.13 ± 0.12	0.27 ± 0.12	0.07 ± 0.12
w/o Multi-Agent	0.73 ± 0.31	0.40 ± 0.20	0.53 ± 0.31	0.20 ± 0.00	0.47 ± 0.12	0.20 ± 0.20
w/o Goal Estimation	0.60 ± 0.20	0.33 ± 0.23	0.67 ± 0.31	0.40 ± 0.34	0.40 ± 0.20	0.13 ± 0.23
w/o Both	0.47 ± 0.13	0.12 ± 0.12	0.40 ± 0.20	0.07 ± 0.12	0.20 ± 0.20	0.00 ± 0.00
Ours	$\mathbf{0.93 \pm 0.12}$	$\mathbf{0.60 \pm 0.20}$	$\mathbf{0.80 \pm 0.20}$	$\mathbf{0.53 \pm 0.12}$	$\mathbf{0.86 \pm 0.12}$	$\mathbf{0.33 \pm 0.12}$

4.3.4 Implementation Details

Detailed Implementation of Real Robot System

Bimanual Hands System. For our system, we have developed a ROS-based pipeline that operates at a control frequency of 20Hz. This pipeline serves as the foundation for controlling our setup, enabling efficient communication and coordination between the different components. In our configuration, the Arm-Hand subsystems are controlled by a single policy utilizing multiple agents. This unified policy governs the actions of both subsystems, promoting synchronized and collaborative behavior in our setup. To achieve this, we control the motion of the robotic arms through Modbus TCP (Transmission Control Protocol) using an AC/DC Control Box. The control boxes of the two robotic arms are connected to a router via Ethernet cables, and the router is then connected to the host computer. Additionally, the two robot hands are directly connected to the same computer using RS-485 serial communication.

Object Tracking. Real-time object tracking is performed with an Intel RealSense D435 stereo camera. Since the object has a high color contrast from its background, we first use a simple color detector on the RGB image to find the pixel location of the object. The color range for detecting a blue object is constrained between $[80, 200, 0]$ and $[120, 255, 0]$ in HSV color space. Next, the 3D position of that pixel is obtained from querying the corresponding depth value on the depth image, where post-processing filters including disparity, spatial and temporal, are applied to reduce depth noise. Finally, we get the 3D object position in robot frame from image frame with calibrated camera extrinsics parameters.

Sim2Real Transfer

System Identification. To achieve a successful sim-to-real transfer, we utilize system identification techniques to align the behavior of the PD (Proportional-Derivative) controller of the arm and hand in simulation with that in the real world. This involves tuning the PD coefficients of the controllers to ensure that their responses to impulse and sinusoidal inputs are aligned. This step is crucial in ensuring that the control actions generated in simulation can be effectively applied to the real robot setup, enabling a reliable sim-to-real transfer of our system.

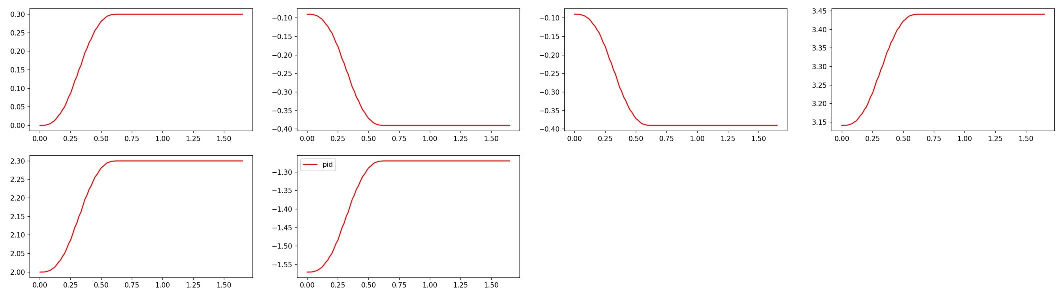


Figure 4.21. Process Reaction Curve of Arm: We set the same delta joint angles for each joint and execute. During this execution, we record the process reaction curve.

Domain Randomization. Isaac Gym offers several domain randomization functions for reinforcement learning training. We apply randomization to the task, as indicated in Table. 4.10 for each environment. We generate new randomizations every 1000 simulation steps.

Table 4.10. Domain randomization parameters.

Parameter	Type	Distribution	Initial Range
Robot			
Mass	Scaling	uniform	[0.5, 1.5]
Friction	Scaling	uniform	[0.7, 1.3]
Joint Lower Limit	Scaling	loguniform	[0.0, 0.01]
Joint Upper Limit	Scaling	loguniform	[0.0, 0.01]
Joint Stiffness	Scaling	loguniform	[0.0, 0.01]
Joint Damping	Scaling	loguniform	[0.0, 0.01]
Object			
Mass	Scaling	uniform	[0.5, 1.5]
Friction	Scaling	uniform	[0.5, 1.5]
Scale	Scaling	uniform	[0.95, 1.05]
Observation			
Obs Correlated. Noise	Additive	Gaussian	[0.0, 0.001]
Obs Uncorrelated. Noise	Additive	Gaussian	[0.0, 0.002]
Action			
Action Correlated Noise	Additive	Gaussian	[0.0, 0.015]
Action Uncorrelated Noise	Additive	Gaussian	[0.0, 0.05]
Environment			
Gravity	Additive	normal	[0, 0.4]

Hyperparameters of the RL Algorithms

The hyperparameters of the RL algorithms used are shown in Table 4.11 and 4.12.

Table 4.11. Hyperparameters of MAPPO.

Hyperparameters	Throw and Catch
Num mini-batches	1
Num opt-epochs	5
Num episode-length	8
Hidden size	[1024, 1024, 512]
Use popart	True
Use value norm	True
Use proper time limits	False
Use Huber loss	True
Huber delta	10
Clip range	0.2
Max grad norm	10
Learning rate	5.e-4
Opt-eps	5.e-4
Discount (γ)	0.96
GAE lambda (λ)	0.95
Std x coef	1
Std y coef	0.5
Ent-coef	0

Table 4.12. Hyperparameters of PPO.

Hyperparameters	Throw and Catch
Num mini-batches	4
Num opt-epochs	5
Num episode-length	8
Hidden size	[1024, 1024, 512]
Clip range	0.2
Max grad norm	1
Learning rate	3.e-4
Discount (γ)	0.96
GAE lambda (λ)	0.95
Init noise std	0.8
Desired kl	0.016
Ent-coef	0

Reward Design

The reward of our system r can be computed as $r = r_{dis} + r_{linvel} + r_{torque}$. In the design of our reward, r_{dis} is the reward that mainly responds to throwing objects to the target position. r_{linvel} is a reward that encourages throwers to release the ball from hand. r_{torque} is a penalty item for robots that torque is too big. In our reward function, if r_{dis} is missing, the object will not be thrown to the exact position, but will only be thrown forward vigorously. Without r_{linvel}, it would often fall into a sub-optimal where the thrower holds the ball in its hand and doesn't release. r_{torque} is a common reward term that allows robots to avoid jitter and large dangerous movements.

4.4 Summary

In this chapter, we first introduce LAPAL, an approach that learns a latent action space to imitate expert behaviors efficiently in robotic locomotion and manipulation tasks. Our experiments show that LAPAL converges faster and yield significant improvements over a standard adversarial imitate learning baseline, especially in high-dimensional complex environments. We also introduce cross embodiment policy transfer where different robots can be aligned in a common latent space representation to perform a task. The latent space alignment is learned via adversarial training for distribution matching and cycle consistency which leverages unpaired data. It enables policy transfer between different types of robot arms either in simulation or in real world. Finally, we introduce Dynamic Handover, a system capable of throwing and catching with bimanual hands. Through the use of multi-agent reinforcement learning and goal estimation, our system demonstrates the ability to achieve successful throw and catch in both simulation and real world environments. We find that the goal estimation aids in mitigating the effects of unpredictable parameters and enhances the overall stability to bridge the large dynamics gap between sim and real.

4.5 Acknowledgements

Chapter 4.1, in part, is currently being prepared for submission for publication of the material as it may appear in T. Wang, N. Karnwal and N. Atanasov, “Latent Policies for Adversarial Imitation Learning.” The dissertation author is the primary author of this paper.

Chapter 4.2, in part, is currently being prepared for submission for publication of the material as it may appear in T. Wang, D. Bhatt, X. Wang and N. Atanasov, “Cross Embodiment Robot Manipulation Skill Transfer from Cycle Consistency.” The dissertation author is the primary author of this paper.

Chapter 4.3, in part, is a reprint of the material as it appears in B. Huang, Y. Chen, T. Wang, Y. Qin, Y. Yang, N. Atanasov, X. Wang, “Dynamic Handover: Throw and Catch with Bimanual Hands, ” Conference on Robot Learning, 2023. The dissertation author is a co-author of this paper.

Chapter 5

Conclusions and Future Work

It is crucial for autonomous robots to adapt quickly to environment changes and generalize prior knowledge to tackle unseen scenarios. In this dissertation, we present methods to learn appropriate representation of the environment and task and obtain generalizable policies which enable robots to imitate effectively from demonstrations and adapt efficiently to domain mismatch.

In Chapter 3, we develop novel models to learn semantic understanding and logic structure from demonstrations. We introduce novel deep learning-based cost function representation which is optimized by differentiable motion planning algorithms. We demonstrate that the models can enable long-horizon planning and generalize to unseen autonomous driving scenarios with dynamic obstacles.

In Chapter 4, we investigate the generalization capabilities of robot manipulation skills across different physically embodied robots and from simulation to real. We introduce latent invariant feature space to align different robot embodiments and learn generalizable latent policies. We also demonstrate that domain randomization and adaptation techniques can be applied to a challenging real world bimanual robot manipulation problem.

To improve generalization capability for autonomous robots, we propose the following directions for future work:

- **Large-scale suboptimal and heterogeneous demonstrations.** In this dissertation we have applied IRL algorithms where expert demonstrations are assumed to be optimal. However, it is often expensive to collect perfect demonstrations at large scale. It remains an open challenge to effectively learn policies from large scale offline data of different levels of suboptimality, in terms of reward function, user preference and ranking [106]. In Chapter 4, we have also discussed learning invariant feature space for heterogeneous robot embodiments in manipulation tasks. Large-scale offline dataset like RT-X [124] contains examples of different robot embodiments across a diverse set of skills and tasks. This suggests that we can learn robot-specific foundation models for better adaptability and generalization capability.

- **Language semantic understanding for planning.** In Chapter 3, we have discussed how to discover and exploit semantics and logic from observations to shape a cost function for motion planning. With the recent advance in large language models (LLMs), we have seen their applications as pretrained models for learning language-conditioned policy in robotics [20, 157]. Understanding language instructions can provide high-level task planning when we can convert them into meaningful semantics, e.g. from natural language to temporal logic.

- **Safety quantification.** LLMs often hallucinate by producing seemingly plausible but factually incorrect, logically inconsistent, or physically infeasible solutions [188]. The applications of LLMs can be limited in safety-critical robotic applications. To enable autonomous robots to survive in the wild or co-exist in human-centered environments, it is crucial to provide safety guarantees, quantify prediction uncertainty and account for distributional shifts.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, page 1, 2004.

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

[3] Fabio Amadio, Adrià Colomé, and Carme Torras. Exploiting symmetries in reinforcement learning of bimanual robotic tasks. *IEEE Robotics and Automation Letters*, 4(2):1838–1845, 2019.

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

[5] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[6] Denis Arrivault, Dominique Benielli, François Denis, and Rémi Eyraud. Scikit-splearn: a toolbox for the spectral learning of weighted automata compatible with scikit-learn. In *Conférence francophone sur l'Apprentissage Aurtomatique*, 2017.

[7] Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *International Conference on Machine Learning*, volume 97, pages 12–20, 1997.

[8] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI Conference on Artificial Intelligence*, 2017.

[9] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[10] Yunfei Bai, Wenhao Yu, and C Karen Liu. Dexterous manipulation of cloth. In *Computer Graphics Forum*, volume 35, pages 523–532. Wiley Online Library, 2016.

[11] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT press, 2008.

[12] Andrea Bajcsy, Dylan P Losey, Marcia K O'malley, and Anca D Dragan. Learning robot objectives from physical human interaction. In *Conference on Robot Learning*, 2017.

[13] Borja Balle and Mehryar Mohri. Spectral learning of general weighted automata via constrained matrix completion. In *Advances in Neural Information Processing Systems*, 2012.

[14] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[15] Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*, volume 12. Springer-Verlag, 1988.

[16] Dimitri Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[17] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi. Sampling-based motion planning with temporal goals. In *IEEE International Conference on Robotics and Automation*, 2010.

[18] Lionel Blondé and Alexandros Kalousis. Sample-efficient imitation learning via generative adversarial nets. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3138–3148. PMLR, 2019.

[19] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on robotics*, 30(2):289–309, 2013.

[20] Rogerio Bonatti, Sai Vemprala, Shuang Ma, Felipe Frujeri, Shuhang Chen, and Ashish Kapoor. Pact: Perception-action causal transformer for autoregressive robotics pre-training. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3621–3627. IEEE, 2023.

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

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

[23] Alejandro M Castro, Frank N Permenter, and Xuchen Han. An unconstrained convex formulation of compliant contact. *IEEE Transactions on Robotics*, 2022.

[24] Letian Chen, Rohan Paleja, and Matthew Gombolay. Learning from suboptimal demonstration via self-supervised reward regression. In *Conference on robot learning*. PMLR, 2021.

[25] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision*, pages 801–818, 2018.

[26] Tao Chen, Megha Tippur, Siyang Wu, Vikash Kumar, Edward Adelson, and Pulkit Agrawal. Visual dexterity: In-hand dexterous manipulation from depth. *arXiv preprint arXiv:2211.11744*, 2022.

[27] Yuanpei Chen, Yaodong Yang, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuan Jiang, Zongqing Lu, Stephen Marcus McAleer, Hao Dong, and Song-Chun Zhu. Towards human-level bimanual dexterous manipulation with reinforcement learning. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[28] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

[29] Cheng Chi, Benjamin Burchfiel, Eric Cousineau, Siyuan Feng, and Shuran Song. Iterative residual policy: for goal-conditioned dynamic manipulation of deformable objects. *arXiv preprint arXiv:2203.00663*, 2022.

[30] Rohan Chitnis, Shubham Tulsiani, Saurabh Gupta, and Abhinav Gupta. Efficient bimanual manipulation using learned task schemas. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1149–1155. IEEE, 2020.

[31] Glen Chou, Necmiye Ozay, and Dmitry Berenson. Explaining multi-stage tasks by learning temporal logic formulas from suboptimal demonstrations. In *Robotics: Science and Systems (RSS)*, 2020.

[32] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D spatio-temporal convnets: Minkowski convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[33] Matei Ciocarlie, Corey Goldfeder, and Peter Allen. Dimensionality reduction for hand-independent dexterous robotic grasping. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3270–3275. IEEE, 2007.

[34] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[35] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International Conference on Machine Learning*, pages 2990–2999, 2016.

[36] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. Salsanext: Fast, uncertainty-aware semantic segmentation of lidar point clouds for autonomous driving. *arXiv preprint arXiv:2003.03653*, 2020.

[37] Neha Das, Sarah Bechtle, Todor Davchev, Dinesh Jayaraman, Akshara Rai, and Franziska Meier. Model-based inverse reinforcement learning from visual demonstrations. In *Conference on Robot Learning*, pages 1930–1942. PMLR, 2021.

[38] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[39] Xu Chu Ding, Stephen L Smith, Calin Belta, and Daniela Rus. Mdp optimal control under temporal logic constraints. In *IEEE Conference on Decision and Control and European Control Conference*, 2011.

[40] David Dohan, Brian Matejek, and Thomas Funkhouser. Learning hierarchical semantic segmentations of lidar data. In *International Conference on 3D Vision*, pages 273–281, 2015.

[41] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[42] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. In *International Colloquium on Automata, Languages, and Programming*, pages 513–525. Springer, 2005.

[43] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer Science & Business Media, 2009.

[44] Georgios E Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[45] Georgios E Fainekos, Hadas Kress-Gazit, and George J Pappas. Hybrid controllers for path planning: A temporal logic approach. In *IEEE Conference on Decision and Control*, 2005.

[46] Marie Farrell, Matt Luckcuck, and Michael Fisher. Robotics and integrated formal methods: Necessity meets opportunity. In *International Conference on Integrated Formal Methods*, 2018.

[47] Norman Ferns and Doina Precup. Bisimulation metrics are optimal value functions. In *UAI*, pages 210–219, 2014.

[48] Fanny Ficuciello, Alessandro Migliozzi, Eulalie Coevoet, Antoine Petit, and Christian Duriez. Fem-based deformation control for dexterous manipulation of 3d soft objects. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4007–4013. IEEE, 2018.

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

[50] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[51] Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

[52] Jie Fu, Nikolay Atanasov, Ufuk Topcu, and George J Pappas. Optimal temporal logic planning in probabilistic semantic maps. In *IEEE International Conference on Robotics and Automation*, 2016.

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

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

[55] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.

[56] Yi Gai, Yukinori Kobayashi, Yohei Hoshino, and Takanori Emaru. Motion control of a ball throwing robot with a flexible robotic arm. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 7:937–945, 2013.

[57] L. Gan, R. Zhang, J. W. Grizzle, R. M. Eustice, and M. Ghaffari. Bayesian spatial kernel smoothing for scalable dense semantic mapping. *IEEE Robotics and Automation Letters*, 5(2):790–797, 2020.

[58] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of machine learning research*, 17(59):1–35, 2016.

[59] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. Deep predictive policy training using reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2351–2358. IEEE, 2017.

[60] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pages 1259–1277, 2020.

[61] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pages 1259–1277. PMLR, 2020.

[62] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[63] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 2020.

[64] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[65] Zhaohan Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-Bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. In *International Conference on Machine Learning*, pages 3875–3886. PMLR, 2020.

[66] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *International Conference on Learning Representations*, 2017.

[67] Abhishek Gupta, Clemens Eppner, Sergey Levine, and Pieter Abbeel. Learning dexterous manipulation for a soft robotic hand from human demonstrations. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3786–3793. IEEE, 2016.

[68] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[69] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361, 2017.

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

[71] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. 2019.

[72] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, Yashraj Narang, Jean-Francois Lafleche, Dieter Fox, and Gavriel State. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. *arXiv*, 2022.

[73] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[74] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposium Series*, 2015.

[75] Donald Hejna, Lerrel Pinto, and Pieter Abbeel. Hierarchically decoupled imitation for morphological transfer. In *International Conference on Machine Learning*, pages 4159–4171. PMLR, 2020.

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

[77] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[78] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[79] Yew Cheong Hou, Khairul Salleh Mohamed Sahari, and Dickson Neoh Tze How. A review on modeling of flexible deformable object for dexterous robotic manipulation. *International Journal of Advanced Robotic Systems*, 16(3):1729881419848894, 2019.

[80] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, volume 37, pages 448–456, 2015.

[81] Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. Learning preferences for manipulation tasks from online coactive feedback. *The International Journal of Robotics Research*, 2015.

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

[83] Hong Jun Jeon, Smitha Milli, and Anca Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning. *Advances in Neural Information Processing Systems*, 2020.

[84] Hong Jin Kang and David Lo. Adversarial specification mining. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2021.

[85] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[86] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[87] Satoshi Kataoka, Seyed Kamyar Seyed Ghasemipour, Daniel Freeman, and Igor Mordatch. Bi-manual manipulation and attachment via sim-to-real reinforcement learning. *arXiv preprint arXiv:2203.08277*, 2022.

[88] Liyiming Ke, Sanjiban Choudhury, Matt Barnes, Wen Sun, Gilwoo Lee, and Siddhartha Srinivasa. Imitation learning as f-divergence minimization. In *International Workshop on the Algorithmic Foundations of Robotics*, 2020.

[89] A. Khan, C. Zhang, N. Atanasov, K. Karydis, V. Kumar, and D. D. Lee. Memory augmented control networks. In *International Conference on Learning Representations*, 2018.

[90] Kuno Kim, Yihong Gu, Jiaming Song, Shengjia Zhao, and Stefano Ermon. Domain adaptive imitation learning. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2020.

[91] Diederik P Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

[92] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. CompILE: Compositional imitation learning and execution. In *International Conference on Machine Learning*, 2019.

[93] Jens Kober, Matthew Glisson, and Michael Mistry. Playing catch and juggling with a humanoid robot. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 875–881. IEEE, 2012.

[94] Jens Kober, Erhan Öztop, and Jan Peters. Reinforcement learning to adjust robot movements to new situations. In *International Joint Conference on Artificial Intelligence*, 2010.

[95] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *arXiv preprint arXiv:1809.02925*, 2018.

[96] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *International Conference on Learning Representations*, 2019.

[97] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Where's waldo? sensor-based temporal logic motion planning. In *IEEE International Conference on Robotics and Automation*, 2007.

[98] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[99] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 2016.

[100] Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. *Advances in Neural Information Processing Systems*, 31, 2018.

[101] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. Tr 98-11, Comp. Sci. Dept., Iowa State University, 1998.

[102] Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, 33:741–752, 2020.

[103] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General ltl specification mining (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015.

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

[105] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[106] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[107] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011.

[108] Sizhe Li, Zhiao Huang, Tao Chen, Tao Du, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Dexdeform: Dexterous deformable object manipulation with human demonstrations and differentiable physics. *arXiv preprint arXiv:2304.03223*, 2023.

[109] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, page 767–774, 2004.

[110] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[111] C. Lu, M. J. G. van de Molengraft, and G. Dubbelman. Monocular semantic occupancy grid mapping with convolutional variational encoder-decoder networks. *IEEE Robotics and Automation Letters*, 4(2):445–452, 2019.

[112] Matt Luckcuck, Marie Farrell, Louise Dennis, Clare Dixon, and Michael Fisher. Formal specification and verification of autonomous robotic systems. *ACM Computing Surveys (CSUR)*, 2019.

[113] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[114] Matthew T. Mason and Kevin Lynch. Dynamic manipulation. In *Proceedings of (IROS) IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 152 – 159, July 1993.

[115] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. Rangenet++: Fast and accurate lidar semantic segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4213–4220, 2019.

[116] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *International Conference on Learning Representations*, 2017.

[117] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[118] Suraj Nair, Eric Mitchell, Kevin Chen, Silvio Savarese, Chelsea Finn, et al. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, pages 1303–1315. PMLR, 2022.

[119] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *Conference on Robot Learning*, 2022.

[120] Gergely Neu and Csaba Szepesvári. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Conference on Uncertainty in Artificial Intelligence*, pages 295–302, 2007.

[121] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, pages 663–670, 2000.

[122] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto. Voxblox: Incremental 3D Euclidean signed distance fields for on-board MAV planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1366–1373, 2017.

[123] Manu Orsini, Anton Raichuk, Léonard Hussenot, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. What matters for adversarial imitation learning? *Advances in Neural Information Processing Systems*, 34, 2021.

[124] Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

[125] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A Theodorou, and Byron Boots. Imitation learning for agile autonomous driving. *The International Journal of Robotics Research*, 39(2-3):286–302, 2020.

[126] George Papandreou, Liang-Chieh Chen, Kevin P Murphy, and Alan L Yuille. Weakly- and semi-supervised learning of a deep convolutional network for semantic image segmentation. In *IEEE International Conference on Computer Vision*, pages 1742–1750, 2015.

[127] Jyothish Pari, Nur Muhammad Shafiullah, Sridhar Pandian Arunachalam, and Lerrel Pinto. The surprising effectiveness of representation learning for visual imitation. *Proceedings of Robotics: Science and Systems (RSS)*, 2021.

[128] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *IEEE International Conference on Robotics and Automation*, pages 763–768, 2009.

[129] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8026–8037, 2019.

[130] Aleksei Petrenko, Arthur Allshire, Gavriel State, Ankur Handa, and Viktor Makoviychuk. Dexpbt: Scaling up dexterous manipulation for hand-arm systems with population based training. In *Robotics: Science and Systems*, 2023.

[131] Erion Plaku and Sertac Karaman. Motion planning with temporal-logic specifications: Progress and challenges. *AI communications*, pages 151–162, 2016.

[132] Haozhi Qi, Ashish Kumar, Roberto Calandra, Yi Ma, and Jitendra Malik. In-Hand Object Rotation via Rapid Motor Adaptation. In *Conference on Robot Learning (CoRL)*, 2022.

[133] Yuzhe Qin, Binghao Huang, Zhao-Heng Yin, Hao Su, and Xiaolong Wang. Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation. In *Conference on Robot Learning*, pages 594–605. PMLR, 2023.

[134] Yuzhe Qin, Hao Su, and Xiaolong Wang. From one hand to multiple hands: Imitation learning for dexterous manipulation from single-camera teleoperation. *IEEE Robotics and Automation Letters*, 7(4):10873–10881, 2022.

[135] Guillaume Rabusseau, Tianyu Li, and Doina Precup. Connecting weighted automata and recurrent neural networks through spectral learning. In *International Conference on Artificial Intelligence and Statistics*, 2019.

[136] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3, 2019.

[137] Aravind Rajeswaran*, Vikash Kumar*, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[138] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *International Joint Conferences on Artificial Intelligence Organization*, volume 7, pages 2586–2591, 2007.

[139] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *International Conference on Machine Learning*, pages 729–736, 2006.

[140] Matthew Riemer, Miao Liu, and Gerald Tesauro. Learning abstract options. 2018.

[141] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, pages 661–668, 2010.

[142] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

[143] Arto Salomaa and Matti Soittola. *Automata-theoretic Aspects of Formal Power Series*. Springer Science & Business Media, 2012.

[144] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in Neural Information Processing Systems*, 28, 2015.

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

[146] Sunando Sengupta, Paul Sturgess, L'ubor Ladickỳ, and Philip HS Torr. Automatic dense visual semantic mapping from street-level imagery. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 857–862, 2012.

[147] Taku Senoo, Akio Namiki, and Masatoshi Ishikawa. High-speed throwing motion based on kinetic chain approach. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3206–3211, 2008.

[148] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1134–1141. IEEE, 2018.

[149] Ankit Jayesh Shah, Pritish Kamath, Shen Li, and Julie A Shah. Bayesian inference of temporal task specifications from demonstrations. 2018.

[150] Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*, volume 3. Springer Science & Business Media, 2012.

[151] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[152] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[153] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[154] Yeeho Song. Inverse reinforcement learning for autonomous ground navigation using aerial and satellite observation data. Master's thesis, Carnegie Mellon University, 2019.

[155] Bradly C Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *International Conference on Learning Representations*, 2017.

[156] L. Sun, Z. Yan, A. Zaganidis, C. Zhao, and T. Duckett. Recurrent-octomap: Learning state-based map refinement for long-term semantic mapping with 3D-lidar data. *IEEE Robotics and Automation Letters*, 3(4):3749–3756, 2018.

[157] Yanchao Sun, Shuang Ma, Ratnesh Madaan, Rogerio Bonatti, Furong Huang, and Ashish Kapoor. Smart: Self-supervised multi-task pretraining with control transformers. *arXiv preprint arXiv:2301.09816*, 2023.

[158] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[159] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[160] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[161] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[162] Parker Allen Tew. *An investigation of sparse tensor formats for tensor libraries*. PhD thesis, Massachusetts Institute of Technology, 2016.

[163] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[164] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[165] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[166] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE international conference on computer vision*, pages 4068–4076, 2015.

[167] Nikolaus Vahrenkamp, Markus Przybylski, Tamim Asfour, and Rüdiger Dillmann. Bimanual grasp planning. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 493–499. IEEE, 2011.

[168] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[169] Angelina Wang, Thanard Kurutach, Kara Liu, Pieter Abbeel, and Aviv Tamar. Learning robotic manipulation through visual planning and acting. *Proceedings of Robotics: Science and Systems (RSS)*, 2019.

[170] Steven Wang, Sam Toyer, Adam Gleave, and Scott Emmons. The `imitation` library for imitation learning and inverse reinforcement learning. https://github.com/HumanCompatibleAI/imitation, 2020.

[171] Tianyu Wang, Vikas Dhiman, and Nikolay Atanasov. Learning navigation costs from demonstration in partially observable environments. In *IEEE International Conference on Robotics and Automation*, 2020.

[172] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3D lidar point cloud. In *International Conference on Robotics and Automation*, pages 1887–1893, 2018.

[173] Markus Wulfmeier, Ingmar Posner, and Pieter Abbeel. Mutual alignment transfer learning. In *Conference on Robot Learning*, pages 281–290. PMLR, 2017.

[174] Markus Wulfmeier, Dominic Zeng Wang, and Ingmar Posner. Watch this: Scalable cost-function learning for path planning in urban environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2089–2095, 2016.

[175] Ning Xi, Tzyh-Jong Tarn, and Antal K Bejczy. Intelligent planning and control for multirobot coordination: An event-based approach. *IEEE transactions on robotics and automation*, 12(3):439–452, 1996.

[176] Fan Xie, Alexander Chowdhury, M Kaluza, Linfeng Zhao, Lawson LS Wong, and Rose Yu. Deep imitation learning for bimanual robotic manipulation. In *Advances in Neural Information Processing Systems*, 2020.

[177] Jianglong Ye, Jiashun Wang, Binghao Huang, Yuzhe Qin, and Xiaolong Wang. Learning continuous grasping function with a dexterous hand from human demonstrations. *IEEE Robotics and Automation Letters*, 2023.

[178] Zhao-Heng Yin, Binghao Huang, Yuzhe Qin, Qifeng Chen, and Xiaolong Wang. Rotating without seeing: Towards in-hand dexterity through touch. *arXiv preprint arXiv:2303.10880*, 2023.

[179] Zhao-Heng Yin, Lingfeng Sun, Hengbo Ma, Masayoshi Tomizuka, and Wu-Jun Li. Cross domain robot imitation with invariant representation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 455–461. IEEE, 2022.

[180] Takuma Yoneda, Ge Yang, Matthew R. Walter, and Bradly Stadie. Invariance through latent alignment. *Proceedings of Robotics: Science and Systems (RSS)*, 2022.

[181] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

[182] Kevin Zakka, Laura Smith, Nimrod Gileadi, Taylor Howell, Xue Bin Peng, Sumeet Singh, Yuval Tassa, Pete Florence, Andy Zeng, and Pieter Abbeel. RoboPianist: A Benchmark for High-Dimensional Robot Control, 2023.

[183] Kevin Zakka, Andy Zeng, Pete Florence, Jonathan Tompson, Jeannette Bohg, and Debidatta Dwibedi. Xirl: Cross-embodiment inverse reinforcement learning. In *Conference on Robot Learning*, pages 537–546. PMLR, 2022.

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

[185] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

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

[187] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[188] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren's song in the ai ocean: A survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*, 2023.

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

[190] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. Reinforcement and imitation learning for diverse visuomotor skills. In *Robotics: Science and Systems*, 2018.

[191] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[192] Brian D. Ziebart, Andrew Maas, J.Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, pages 1433–1438, 2008.

[193] R Zollner, Tamim Asfour, and Rüdiger Dillmann. Programming by demonstration: dual-arm manipulation tasks for humanoid robots. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 1, pages 479–484. IEEE, 2004.