LEARNING TO ADAPT FOR INTELLIGENT ROBOT BEHAVIOR

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

Mengxi Li
March 2023

Abstract

The field of robotics has been rapidly evolving in recent years, and robots are being used in an ever-increasing number of applications, from manufacturing to healthcare to household chores. One of the key challenges in robotics is enabling robots to perform complex manipulation tasks in unstructured and dynamic environments. While there have been significant advances in robot learning and control, many existing approaches are limited by their reliance on pre-defined motion primitives or generic models that do not account for the specific characteristics of individual users, other cooperative agents or the interacting objects. In order to be effective in these various settings, robots need to be able to adapt to different tasks and environments, and to interact with different types of agents, such as humans and other robots. This thesis investigates learning approaches for enabling robots to adapt their behavior in order to achieve intelligent robot behavior.

In the first part of this thesis, we focus on enabling robots to better adapt to humans. We start by exploring how to leverage different sources of data to achieve personalization for human users. Firstly, we investigate how humans prefer to teleoperate assistive robot arms using low-dimensional controllers, such as joysticks. We present an algorithm that can efficiently develop personalized control for assistive robots. Here the data is obtained by initially demonstrating the behavior of the robot and then query the user to collect their corresponding preferred teleoperation control input from the joysticks. Subsequently, we delve into the exploration of leveraging weaker signals to infer information from agents, such as physical corrections. Experiment results indicate that human corrections are correlated and reasoning over these corrections together achieves improved accuracy. Finally, instead of only adapting to a single human user, we investigate how robots can more efficiently cooperate with and influence human teams by reasoning and exploiting the team structure. We apply our framework to two types of group dynamics, leading-following and predator-prey, and demonstrate that robots can first develop a group representation and utilize this representation to successfully influence a group to achieve various goals.

In the second part of this thesis, we extend our investigation from human users to robot agents. We tackle the problem of how decentralized robot teams can adapt to each other by observing only the actions of other agents. We identify the problem of an infinite reasoning loop within the team and propose a solution by assigning different roles, such as "speaker" and "listener," to the robot

agents. This approach enables us to treat observed actions as a communication channel, thereby achieving effective collaboration within the decentralized team.

Moving on to the third part of this thesis, we explore the topic of adapting to different tasks by developing customized tools. We emphasize the critical role of tools in determining how a robot interacts with objects, making them important in customizing robots for specific tasks. To address this, we present an end-to-end framework to automatically learn tool morphology for contact-rich manipulation tasks by leveraging differentiable physics simulators.

Finally, we conclude the thesis by summarizing our efforts and discussing future directions.

Acknowledgments

I would like to express my deepest gratitude to my thesis advisors, Prof. Dorsa Sadigh and Prof. Jeannette Bohg, for their unwavering support, guidance, and encouragement throughout my research journey. Before starting my Ph.D., I did not have any research experiences on robotics. Looking back, I feel extremely fortunate to have the chance to rotate with them and end up getting co-advised. Throughout the five years, their invaluable insights, patience, and expertise were instrumental in shaping my research and helping me overcome the challenges that I faced along the way. Notably, in addition to their academic expertise, they are also supportive mentors and compassionate friends who have made a significant impact on my personal and professional development. Their influence also extends beyond the research, and I have learned a great deal from them about life and leadership. In my future career path, I will continue to emulate these outstanding qualities. And hopefully, when I eventually assume the role of a mentor, I can create a positive impact on my mentees' personal and professional development, just as my advisors have done for me.

I am also grateful to other members of my thesis committee, Prof. Chelsea Finn, Prof. Mykel Kochenderfer and Prof. Karen Liu for their insightful and constructive feedback, which helped me refine my work and improve the quality of my thesis. I would also like to thank Prof. Raquel Urtasun, Prof. Renjie Liao, Prof. Jun Zhu and Prof. Yujin Zhang, for their mentorship and advising during my undergraduate years. They led me into the amazing world of research and without them I would not have pursued my Ph.D in the first place.

Most of my works are done in close collaboration with other incredible minds. I would like to thank all my co-authors including Rika Antonova, Dylan Losey, Minae Kwon, Zhangjie Cao, Alper Canberk, Alexandre Bucquet and Yucen Luo. I would also like to thank all my colleagues and labmates including Erdem Biyik, Lin Shao, Mengyuan Yan, Sidd Karamcheti, Shushman Choudhury, Michelle Lee, Yuchen Cui, Negin Heravi, Andy Shih, Wenzhen Yuan, Suneel Belkhale, Claire Chen, Megha Srivastava, Mike Salvato, Priya Sundaresan, Toki Migimatsu, Jennifer Grannen, Krishnan Srinivasan, Joey Hejna, Suvir Mirchandani, Woodrow Z. Wang, Zihan Wang, Andrey Kurenkov, Yilin Wu, Ruinan Wang, Chris Agia and Marion Lepert for insightful discussion and fostering a pleasant and supportive atmosphere in the lab. During my Ph.D., I have also spent two wonderful summers doing industrial internship. I would like to thank my internship advisors Kavya Srinet,

Arthur Szlam, Ajay Mandlekar, Animesh Garg, Fabio Ramos and Prof. Dieter Fox for kindly hosting and advising me.

Throughout the years, I've always felt loved and supported by my amazing friends. First I would like to thank my longtime friends, including Yupei Jian, Yunzhu Hu, Minghan Haung, Zhiyue Song and Shirui He. We've known each other for more than ten years and your friendship continues to nourish me even if we reside in different places now. I also want to thank my college and graduate school friends including Haotian Du, Xiaomeng Jin, Mingyi Lu, Zhiren Bao, Kaichun Mo, Kaidi Cao, Hongyu Ren, Ying Sheng, Hongwei Wang, Xiaoye Yuan, Meishen Liu, Jia Zhuang, Chenzhuo Zhu, Han Xu, Qiaoyi Liu, Shujia Liang, Yiyun Liang, Xueyao Zhang, Yuanlei Niu, Ruochen Yang, Jingxuan Hou, Yang Wang, Yeming Wen, Jingbo Yang, Tianyuan Huang, Yuan Zhou etc.

Finally, I am deeply indebted to my family, including my parents Jianzhong Li, Nina Wang and my fiancé Zheng Wu for their support in all aspects. Throughout my academic journey, their constant encouragement have been my greatest source of inspiration.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

As the field of robotics has been rapidly evolving in recent years, robots are being used in an ever-increasing number of applications, from manufacturing to healthcare to household chores. With robots becoming more capable of performing complex tasks and interacting with humans in a variety of scenarios, it is crucial that they can adapt to changing circumstances, whether it be a change in the environment, the task, or the users they are interacting with. We refer robot adaptation to the ability of robots to modify their behavior in response to changes in their environment, task or users. This can involve adjusting their behavior according to the specific environment, changing the way they communicate and coordinate with other agents, or reasoning over the intent of specific human users to better meet their needs. Adaptation is essential for robots to operate effectively in the real world, where the environment can be unpredictable, tasks can be dynamic, and users can have varying abilities and preferences.

One of the main reasons why robot adaptation is necessary is that robots are often deployed in situations where the task or environment is not well-defined or predictable. For example, in disaster response scenarios, robots may need to navigate through partially collapsed buildings or hazardous environments to locate and rescue survivors. In these situations, the robot needs to be able to adapt its behavior to efficiently communicate and collaborate with other human rescue teams and robots, and navigate the environment safely and effectively. Similarly, in healthcare settings, robots may need to adapt their behavior to accommodate the needs and preferences of different patients, who may have varying levels of physical or cognitive abilities.

However, despite the importance of robot adaptation, it is a challenging problem. First, while significant advancements have been made in automation in recent years, the development of robots with full autonomy remains a challenge. This limitation is due to the fact that robots lack the ability to handle all possible scenarios and respond appropriately to unexpected events. Second, even when

the robot is fully capable of executing current tasks, there can be various human preferences over how to complete them, especially in complex manipulation scenarios. Different humans might assume different preferences, and even the same human users can change their intentions over time. Dealing with the variations and the non-stationarity of human intentions requires the robot to have a better understanding of humans. Finally, the range of tasks that a robot may need to perform can be quite broad, ranging from manipulation tasks that require precise control to social interactions that require nuanced communication and understanding of human behavior.

To this end, the goal of this dissertation is to address these challenges and develop methods for enabling robots to adapt to changing circumstances, including robots adapting to humans, robots adapting to robots, and robots adapting to tasks. A detailed outline of the thesis is provided in Sec. 1.2. By addressing these different aspects of robot adaptation, we aim to develop a more comprehensive understanding of the challenges involved and facilitate more intelligent robot behavior.

1.2 Thesis Outline

This dissertation contains three parts. Part I describes how we leverage different sources of data to enable robots to adapt to humans, including individual human users and human teams. Part II further extends the adaptation from humans to robot agents and introduces how we enable decentralized robot teams to learn from their partners' actions. In Part III, we focus on adapting the robot for specific tasks by designing customized tools. We provide more detailed descriptions of these components as follows.

Part I: Adaptation to Humans. In this part, we first focus on achieving personalization for individual human users to improve user experiences and execute tasks better in Chapter 2 and Chapter 3, and further investigate how to more efficiently cooperate and influence human teams in Chapter 4.

- In Chapter 2, we present a general framework to enable personalized control for teleoperating assistive robots. When humans control drones, cars, and robots, we often have some preconceived notion of how our inputs should make the system behave. Existing approaches to teleoperation typically assume a one-size-fits-all approach, where the designers pre-define a mapping between human inputs and robot actions, and every user must adapt to this mapping over repeated interactions. Instead, we propose a personalized method for learning the human's preferred or preconceived mapping from a few robot queries. Given a robot controller, we identify an alignment model that transforms the human's inputs so that the controller's output matches their expectations. We make this approach data-efficient by recognizing that human mappings have strong priors: we expect the input space to be proportional, reversable, and consistent. Incorporating these priors ensures that the robot learns an intuitive mapping from few examples. We test our learning approach in robot manipulation tasks inspired by

assistive settings, where each user has different personal preferences and physical capabilities for teleoperating the robot arm. Our simulated and experimental results suggest that learning the mapping between inputs and robot actions improves objective and subjective performance when compared to manually defined alignments or learned alignments without intuitive priors. The content of this chapter is based primarily on [108].

- In Chapter 3, we explore leveraging weaker signals, i.e. physical corrections to reason over human preferred objectives. When assistive, and interactive robots make mistakes, humans naturally and intuitively correct those mistakes through physical interaction. In simple situations, one correction is sufficient to convey what the human wants. But when humans are working with multiple robots or the robot is performing an intricate task often the human must make several corrections to fix the robot's behavior. Prior research assumes each of these physical corrections are *independent* events, and learns from them one-at-a-time. However, this misses out on crucial information: each of these interactions are interconnected, and may only make sense if viewed together. Alternatively, other work reasons over the *final* trajectory produced by all of the human's corrections. But this method must wait until the end of the task to learn from corrections, as opposed to inferring from the corrections in an online fashion. In this chapter, we formalize an approach for learning from sequences of physical corrections during the current task. To do this we introduce an auxiliary reward that captures the human's trade-off between making corrections which improve the robot's immediate reward and long-term performance. We evaluate the resulting algorithm in remote and in-person human-robot experiments, and compare to both *independent* and *final* baselines. Our results indicate that users are best able to convey their objective when the robot reasons over their sequence of corrections. The content of this chapter is based primarily on [106].

- In Chapter 4, we shift our focus from adapting to individual humans to adapting to human teams. In human groups, roles such as leading and following can emerge naturally. However, in human-robot teams, such roles are often predefined due to the difficulty of scalably learning and adapting to them. In this chapter, we enable a robot to efficiently learn how group dynamics emerge and evolve in human teams and we leverage this understanding to plan for influencing actions for autonomous robots that guide the team toward achieving a common goal. We first develop an effective and concise representation of group dynamics, such as leading and following, by enforcing a graph structure while learning the weights of the edges corresponding to one-to-one relationships between the agents. We then develop an optimization-based robot policy that leverages this graph representation to attain an objective by influencing a human team. We apply our framework to two types of group dynamics, leading-following and predator-prey, and show that our structured representation is scalable with different human team sizes and also generalizable across different tasks. We also show that robots that utilize this representation are able to successfully influence a group to achieve

various goals compared to robots that do not have access to these graph representations. The content of this chapter is based primarily on [107].

Part II: Adaptation to Robots. In this part, we extend from adapting to humans to adapting to robot agents.

- In Chapter 5, we explore how decentralized robot teams can adapt to each other by only observing the other agents' actions. Similar to Chapter 3, we rely on weak signals obtained by observing the actions of the other agents, enabling us to reason about their intents. Communication is often necessary, when teams of robots collaborate to complete a task. Like humans, robot teammates should implicitly communicate through their actions: but interpreting our partner's actions is typically difficult, since a given action may have many different underlying reasons. Here we propose an alternate approach: instead of not being able to infer whether an action is due to exploration, exploitation, or communication, we define separate roles for each agent. Because each role defines a distinct reason for acting (e.g., only exploit, only communicate), teammates now correctly interpret the meaning behind their partner's actions. Our results suggest that leveraging and alternating roles leads to performance comparable to teams that explicitly exchange messages. The content of this chapter is based primarily on [116].

Part III: Adaptation to Tasks. In this part, we delve into enabling robot adaptation for specific tasks by designing customized tools.

- In Chapter 6, we investigate customized tool design for specific contact-rich manipulation task. When humans perform contact-rich manipulation tasks, customized tools are often necessary to simplify the task. For instance, we use various utensils for handling food, such as knives, forks and spoons. Similarly, robots may benefit from specialized tools that enable them to more easily complete a variety of tasks. We present an end-to-end framework to automatically learn tool morphology for contact-rich manipulation tasks by leveraging differentiable physics simulators. Previous work relied on manually constructed priors requiring detailed specification of a 3D object model, grasp pose and task description to facilitate the search or optimization process. Our approach only requires defining the objective with respect to task performance and enables learning a robust morphology through randomizing variations of the task. We make this optimization tractable by casting it as a continual learning problem. We demonstrate the effectiveness of our method for designing new tools in several scenarios, such as winding ropes, flipping a box and pushing peas onto a scoop in simulation. Additionally, experiments with real robots show that the tool shapes discovered by our method help them succeed in these scenarios. The content of this chapter is based primarily on [105].

Lastly, we summarize the contributions of this thesis and discuss promising directions for future research in Chapter 7.

Part I

Adaptation to Humans

Chapter 2

Learning User-Preferred Mappings for Intuitive Robot Control

2.1 Introduction

Humans are extremely good at developing tools and systems. Each of us interact with these systems everyday—from driving your car to work to playing video games at home. Unfortunately, we are not always good at these interactions. Using a joystick to play a kart racing computer game seems relatively easy, and we can do this without thinking much about it. But actually learning how to drive a real-life car does require practice. Even more challenging is flying a helicopter—this is highly demanding, and requires professional training. In these systems the human must adapt to the robot's controller across multiple rounds of interaction. Rather than forcing the human to adapt to the robot, we wonder if instead the robot could intelligently adapt to our preferences, making control more easy and intuitive?

Alongside rapid advancements in the field of robotics, intuitive control is increasingly in demand, particularly since a wide spectrum of users are operating robots beyond just engineers [9]. User-friendliness and comfort are crucial in many application domains including surgical and assistive robots [10, 158], mobile robots [128] and even more abstract smart home systems [81]. At the heart of these systems, there is seamless *teleoperation*. Typically—for teleoperation—there are two main categories of control methods. One is to use control interfaces such as joysticks or sip-and-puff devices [26, 88]. These controllers are light-weight but low-dimensional, which makes control of robots with many *degrees of freedom* (DoF) challenging. To make up for this limitation when teleoperating robot arms, [77] propose to that the controller should change modes between different DoFs. Other work focuses on capturing high-DoF human body movement with wearable devices, cameras, or sensors [153], and then maps them to robot actions. While these methods provide more

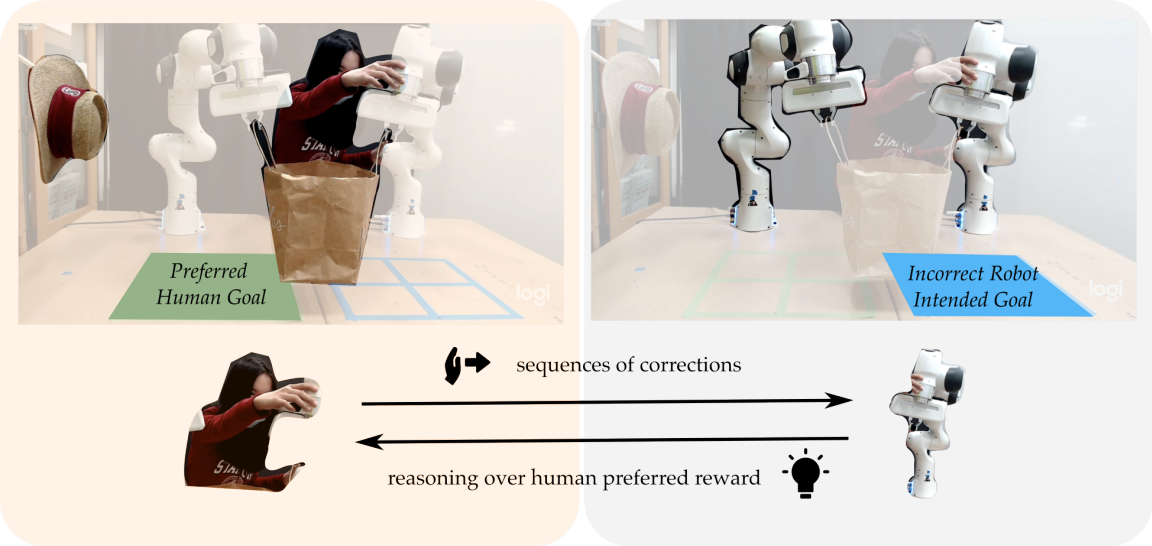


Figure 2.1: The human has in mind a preferred mapping between their joystick inputs and the robot's actions. The robot learns this mapping (i.e., θ) offline from labelled data and intuitive priors, so that the robot's online actions are correctly aligned with the human's intentions.

accurate measurements and natural control interfaces, they are typically larger and more expensive systems that might not be available to everyday users.

Within this work, we are interested in the first category of teleoperation methods. We present a human-centered, adaptive system for robotic manipulation so that human users can smoothly and intuitively teleoperate high-DoF robots with simple, low-DoF controllers. We envision a model that provides a general framework for learning intuitive input mappings that are agnostic to both the controller and the dynamics of the underlying robotic system (see Fig. 2.1). Based on a general architecture of feedforward neural networks, the presented framework can be applied to different robots and controllers for various tasks with only a few demonstrations needed from the human users. Our key insight is:

By incorporating the intuitive priors that humans expect, including proportionality, reversability and consistency, we can quickly learn how humans prefer to control robots.

Our approach first asks the human user to label a set of robot actions with their preferred inputs. For example, the robot moves towards the cup, and then asks what joystick direction should be associated with this action. Based on a small number of these labeled state-action pairs, we learn an alignment model that maps from human inputs to robot inputs such that the resulting robot actions match the human's preferences. Ensuring a *small number of questions* is critical for real-world implementation: we cannot ask the human to label hundreds of examples actions! Our insight—incorporating intuitive priors—enables the robot to generalize across the workspace, leading to offline learning from a limited amount of labeled data.

More precisely, we make the following contributions:

Formalizing Priors for Human Control. We formalize the properties that humans expect to have when controlling robotic systems, including proportionality, reversability, and consistency. These priors are inspired by [93], and we here analyse their relative importance when learning the human's preferred input mapping.

Generalizable Data-Efficient Learning. We build a general framework for learning human control preferences. We develop this data-efficient method by incorporating the intuitive priors that humans expect, so that users only need to provide a few examples of their desired mappings. Our proposed method develops a light-weight solution that can generalize across different controllers, dynamics, and tasks.

Evaluating Learned Human Alignment. We implement our approach both in simulation and on robot manipulation tasks, and get subjective feedback from users. Our tasks are inspired by assisitve robotics, where users living with physical disabilities have different capabilities and preferences. Results in simulation as well as in user studies suggest that our algorithm successfully learned the human-preferred alignment between the low-DoF control interface and high-DoF robot actions. In practice, this learned alignment led to improved control and more seamless interaction.

2.2 Related Work

Intuitive Control. When controllers are intuitive, the effects of user inputs match with that user's expectations. Such a property is very important for robotic control, especially in human-robot teleoperation. Many research efforts are devoted to tackle this problem [52, 68, 38]. Some use wearable devices to capture human motion patterns [156], some leverage augmented reality and use a gesture-based system to enable intuitive human robot teaming [67], and others rely on various sensors to measure and understand humans' haptic feedback [82]. These previous works require additional devices to infer the human intent behind their actions, and then provide a universal solution for intuitive control. Instead of directly enforcing a *one-size-fits-all* control strategy to human users, we separately query each user for their *individual* preferences. With no additional hardware requirements, we come up with a light-weight solution for intuitive control that could generalize across different platforms and tasks.

Assistive Robots. Intelligent assistive robot systems (e.g., wheelchair-mounted robot arms) are one important setting in which intuitive control is essential for widespread use [195]. Because users are constrained by person-specific capabilities, one-size-fits-all controllers are insufficient. Moreover, the problem of "dimensionality mismatch" arises when humans try to teleoperate these high-DoF robots by manipulating a low-DoF controller, e.g., a 2-axis joystick. Prior works tackle this with a mode switching mechanism [141, 185]: however, this forces users to constantly switch modes when performing complex tasks. Other works mitigate mode-switching with model-based optimization [77]

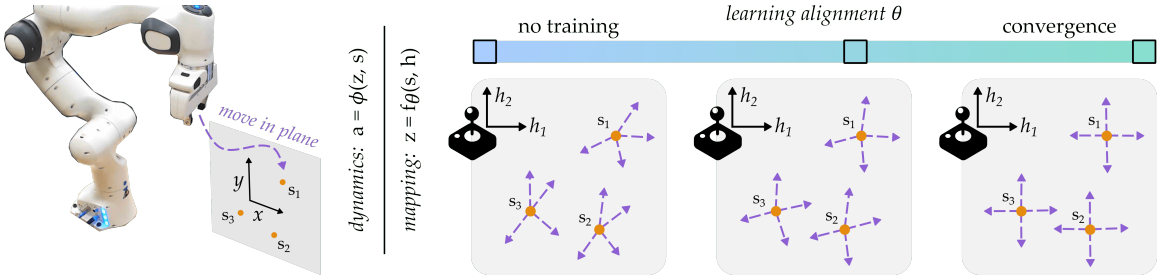


Figure 2.2: Visualization for the training process of the alignment model $z = f_\theta(h, s)$ parametrized by θ. Here the example task is for the robot to move in a plane, and the human would like the robot's end-effector motion to align with their joystick inputs (h_1 and h_2). We take snapshots at three different points during training, and plot how the robot actually moves when the human presses up, down, left, and right. Note that this alignment is state dependent. As training progresses, the robot learns the alignment θ, and the robot's motions are gradually and consistently pushed to match with the human's preferences.

or in a data-driven manner [86], but the underlying mode switching still remains discontinuous. An alternative was recently proposed by [120], in which the robot learns a continuous mapping between a low-DoF latent action space and the high-DoF robot action space using autoencoders. While this approach enables continuous control of the high-DoF robot, it still fails to provide an intuitive control mapping to the user.

In this chapter, we use *assistive robot manipulation* as the test domain for our framework. We consider the teleoperation problem from the user's perspective, and attempt to learn a mapping between their inputs and the assistive robot's actions which caters to the user's preferences.

2.3 Formalism for Modeling Preference Alignment

We formulate a robot manipulation task as a discrete-time Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0)$. Here $\mathcal{S} \subseteq \mathbb{R}^n$ is the state space, $\mathcal{A} \subseteq \mathbb{R}^m$ is the robot action space, $\mathcal{T}(s, a)$ is the transition function, R is the reward, γ is the discount factor, and ρ_0 is the initial distribution.

2.3.1 Problem Statement.

Assume humans control a robot using the teleoperation interface $a_t = \phi(z_t, s_t)$, where t denotes timestep, $s_t \in \mathcal{S}$ is the system state, $z_t \in \mathbb{R}^d$ is the d-dimensional system input, and $a_t \in \mathcal{A}$ is the action executed by the robot. We use $h_t \in \mathbb{R}^d$ to denote the *human's input*. For instance, when the human is using a joystick to teleoperate the robot arm, their two-DoF joystick input corresponds to $h \in \mathbb{R}^2$.

Traditionally, the human's input h is directly used as the system input z, so that $z_t = h_t$, and the robot takes action $a_t = \phi(h_t, s_t)$. However — for complex or non-intuitive systems — it may be hard for human users to directly interact with the controller ϕ. For instance, using a low-DoF joystick

to control a high-DoF assistive robot can be difficult, since we do not know how to coordinate the robot's joints. Moreover, different users also have different preferences for controlling their robot—which may or may not match with the system controller. Put another way, the pre-defined mapping of the controller $a_t = \phi(h_t, s_t)$ is often quite different from what users want!

Our goal is to make the robotic system easier to control: instead of forcing the human to adapt to ϕ, we want the robot to adapt to the user's preferences (without fundamentally changing the controller). We therefore propose to learn an *alignment function* $z_t = f_\theta(h_t, s_t)$ parameterized by θ, which maps the human input $h_t \in \mathbb{R}^d$ and the robot state $s_t \in \mathcal{S}$ to the controller input $z_t \in \mathbb{R}^d$. In this way, we construct a new, two-step mapping between the action of the robotic system a_t and the human's input h_t:

$$a_t = \phi(z_t, s_t) = \phi(f_\theta(h_t, s_t), s_t) \tag{2.1}$$

State Conditioning. Consider the person in Fig. 2.1, who is using a 2-axis joystick to control a high-DoF assistive robot arm to reach and pour a cup. The user's preferred way to control the robot is unclear: what does the user mean if they push the joystick forward? When the robot is far from the cup, the user might intend to move the robot towards the cup—but when the robot is holding the cup, pressing forward now indicates that the robot should rotate, and pour the cup into a bowl! This mapping from the user input to intended action is not only person dependent, but it is also *state dependent*. In practice, this state dependency prevents us from learning a single transformation to uniformly apply across the robot's workspace—we need an intelligent strategy for understanding the human's preferences in different contexts.

2.3.2 Background: Latent Action Embeddings

We will leverage the assistive robot controller proposed in [120] as the main test domain for our alignment framework. Here a conditional autoencoder is trained from demonstrations of related tasks, learning the control function ϕ in Eqn. (2.1). More formally, $\phi : \mathcal{Z} \times \mathcal{S} \mapsto \mathcal{A}$ is a decoder that recovers a high-DoF robot action $a_t \in \mathcal{A}$ given the system input $z_t \in \mathcal{Z}$ and current context $s_t \in \mathcal{S}$. Overall, ϕ is a suitable test domain for our work since it is not immediately clear what the system inputs map to, i.e., there is no prior over how z_t is aligned with a_t at different states. We also point out that this controller captures the state conditioning described above. Since ϕ depends on s, the robot's action a changes based on the current context: the same input z moves the robot towards the cup when the gripper is empty, and then rotates to pour when holding the cup. This enables easy switching between tasks, like reaching and pouring.

2.4 Approach

2.4.1 Constructing Alignment Model

Human Alignment. As described in Sec. 2.3, we seek to obtain a mapping $z_t = f_\theta(h_t, s_t)$ that converts the human input h_t to the controller input z_t at timestep t. Using this transformed system input, the controller will then execute action a_t on the robot, as in Eqn. (2.1).

Model. We leverage a function $f_\theta(h_t, s_t)$ with parameters θ to capture this alignment. Here s_t — the current state of the robot — is also passed as part of the input since the human's alignment could be state-dependent. In this work, we will utilize a general Multi-Layer Perceptron (MLP) to represent f_θ, where θ becomes the weights of the network.

Objective Function. Given the alignment model f, we will apply supervised learning to match the output of this model to the true human preferences (as visualized in Fig. 2.2).

Formally, with the action a_t computed by Eqn. (2.1), the robot state s_{t+1} at next timestep is given by the transition model $s_{t+1} = \mathcal{T}(s_t, a_t)$. We therefore denote the overall mapping between the current human input h_t at state s_t and the next state s_{t+1} using a function T with parameters θ:

$$s_{t+1} = T_\theta(h_t, s_t) = \mathcal{T}(s_t, \phi(f_\theta(h_t, s_t), s_t)) \tag{2.2}$$

Because we are ultimately interested in how well the robot's action a_t matches the human's preference, we define our objective function as the distance d between the robot's actual next state s_{t+1} and the ground truth state s^*_{t+1}, which is where the human really *intended* to go[1]. Taking the expectation, we get the supervised loss L_{sup}:

$$L_{sup}(\theta) = \mathbb{E}[d(s^*_{t+1}, s_{t+1})] = \mathbb{E}[d(s^*_{t+1}, T_\theta(h_t, s_t))] \tag{2.3}$$

We leverage Euclidean Distance as our distance metric between states, although other metrics are also possible: $d(s_1, s_2) = ||s_1 - s_2||_2$. To approximate the integral in Eqn. (2.3), we implement Monte Carlo Sampling by stochastically picking N data points from the state distribution:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} d(s^{*i}_{t+1}, T_\theta(h_t, s^i_t)) \tag{2.4}$$

2.4.2 Ensuring Data-Efficiency with Intuitive Priors

Recall that we are learning the human's preferred mapping with a function approximator. In general, training these models requires a large number of labeled data samples, particularly in complex scenarios where the mapping changes in different states. However, since we need annotations from humans to identify their intended states, it is impractical for the robot to collect a large

[1] We query the user for this intended state, as explained in Section 2.4.3

labeled dataset. Accordingly—to tackle the challenge of insufficient labeled data—we employ a *semi-supervised* learning method [35]. Our contribution here is to formulate the intuitive priors that humans have over their control mappings as *loss terms*, which can then be leveraged within this semi-supervised learning.

Formally, for a given robotic arm, we let s be the robot's joint position, and we denote the forward kinematics as Ψ. For a human control input h at state s, the next state is given by $T_\theta(h, s)$ as in Eqn. (2.2). The corresponding end-effector pose change is $\Delta x(s) = \Psi(T_\theta(h, s)) - \Psi(s)$. With these definitions in mind, we argue an intuitive control mechanism should satisfy the following properties:

1. *Proportionality* – The amount of change in the 3D pose of the robot's end effector should be proportional to the scale of the human input, i.e.,

$$\alpha \cdot |\Psi(T_\theta(h, s)) - \Psi(s)| = |\Psi(T_\theta(\alpha \cdot h, s)) - \Psi(s)|$$

   where $\alpha \in \mathbb{R}$ is the scaling factor. Proportionality is quite common and intuitive in system design: it indicates that humans expect the system to be linearly interpolatable. We define the proportionality loss L_{prop} as:

$$L_{prop} = \left[\Psi(T_\theta(\alpha \cdot h, s)) - (\Psi(s) + \alpha \Delta x(s))\right]^2$$

   where α is independently sampled from a uniform distribution $\alpha \sim U(-1, 1)$, since the control input is bounded.

2. *Reversability* – If an action h makes the robot transit from state s_1 to s_2, the opposite action, denoted by the negation of h, i.e., $-h$, should be able to make the robot transit from s_2 back to s_1.

$$s_2 = T_\theta(h, s_1) \rightarrow s_1 = T_\theta(-h, s_2)$$

   This property ensures recoverability when users mistakenly operate the system, so that people can undo their mistakes and return to the original state. We define the reversability loss as:

$$L_{reverse} = \left[\Psi(s) - \Psi(T_\theta(-h, T_\theta(h, s)))\right]^2$$

   where $\Psi(s)$ is the current 3D pose of the robot end effector, while $\Psi(T_\theta(-h, T_\theta(h, s)))$ is the 3D pose of the end effector after executing human input h followed by executing the opposite input $-h$.

3. *Consistency* – The same action taken at nearby states should lead to similar amounts of change

in the 3D pose of the robot's end effector, i.e.,

$$\Delta x_1 = |\Psi(T_\theta(h, s_1)) - \Psi(s_1)|$$
$$\Delta x_2 = |\Psi(T_\theta(h, s_2)) - \Psi(s_2)|$$
$$\forall \epsilon > 0, \exists \delta > 0, s.t. |s_1 - s_2| < \delta \rightarrow |\Delta x_1 - \Delta x_2| < \epsilon.$$

Consistency encourages the control to be smooth, so that the mapping does not discontinuously change alignment. We define the consistency loss as:

$$L_{con} = \exp\left(-\gamma||s_1 - s_2||\right) \cdot (\Delta x(s_1) - \Delta x(s_2))^2$$

We use the weight term $\exp\left(-\gamma||s_1 - s_2||\right)$ to gauge the similarity between states s_1 and s_2, where $\gamma > 0$ is a hyperparameter controlling the temperature. A large weight is assigned to the state pair (s_1, s_2) if the difference between them is small.

During the training process, we minimize the supervised loss for *labeled* data combined with the semi-supervised loss for *unlabeled* data using intuitive priors:

$$L = L_{sup} + \lambda_1 L_{prop} + \lambda_2 L_{reverse} + \lambda_3 L_{con} \tag{2.5}$$

Here L_{sup} is supervised loss term as shown in Eq. (2.3), and $\lambda_1, \lambda_2, \lambda_3$ are constant coefficients. Importantly, incorporating these different loss terms — which are inspired by human priors over controllable spaces [93] — enables the robot to generalize the labeled human data (which it performs supervised learning on) to unlabeled states (which it can now perform semi-supervised learning on)!

2.4.3 Data Collection by Querying Users

To enable the training of our algorithm, we need to collect labeled data tuples (s_t, h_t, s^*_{t+1}) as well as unlabeled data tuples (s_t, s^*_{t+1}), where s_t is the robot's current state, h_t is the human's low-dimensional input, and s^*_{t+1} is the intended next robot state corresponding to h_t. We emphasize that the robot never needs to detect the human's intended state — rather, the human labels robot actions with their preferred inputs. Our data collection procedure is as follows:

- **Step 0**: Implement the controller $a = \phi(s, z)$. This step is optional depending on the type of the controller; we want to emphasize that the controller will remain fixed, and will not be altered by our alignment model.
- **Step 1**: For the task of interest, we sample a valid state s_t from the state distribution and randomly sample a valid controller input z_t at state s_t.

- **Step 2**: At the sampled state s_t, we apply system input z_t to the controller ϕ in order to get the robot action $a_t = \phi(s_t, z_t)$.
- **Step 3**: We record the subsequent robot state s_{t+1} after executing the action a_t at the state s_t.
- **Step 4**: Steps 1, 2, 3 are repeated for N iterations to get the unlabeled dataset $\{(s_t^i, s_{t+1}^{*i})_{i=1}^N\}$, consisting of N unlabeled (state, next-state) tuples.
- **Step 5**: We randomly sample (s_t, s_{t+1}^*) from the unlabeled dataset collected in Step 4, and then we query the humans for the corresponding labels h_t. Here the user provides examples to the robot of what controls they would find intuitive to move from s_t to s_{t+1}^*.
- **Step 6**: Step 5 is repeated for K iterations to get the labeled dataset $\{(s_t^i, h_t^i, s_{t+1}^{*i})_{i=1}^K\}$, consisting of $K < N$ labeled samples.

For supervised learning approaches, typically K needs to be fairly large so that the state space is well covered. Here we learn the model with only a small number of labeled samples, and then generalize to other unlabeled states by exploiting intuitive control properties. However, we do point out that the number of queries needed could vary according to the task complexity. In addition, for query selection, more intelligent methods such as *active learning* could be integrated to further improve training. Within this work, we will employ a simple strategy of *passively* sampling from the state distribution. Even with this simple approach, we are able to learn the user's preferred mappings from a limited number of queries in both our simulations and the robot manipulation tasks, as we will show in the following sections.

2.5 Experiments

2.5.1 Simulation Experiments

To test our proposed alignment algorithm, we conducted simulations on a Franka Emika Panda robot arm for three manipulation tasks of increasing complexity. We will first discuss the details that are consistent across all experiments, and then elaborate on each task respectively.

Setup. We used a simulated Panda robot arm. The state $s \in S$ denotes the robot's joint position, and the action $a \in A$ refers to the robot's joint velocity. The system transition function is $s' = s + a \cdot dt$, where dt is the step size. We trained a conditional autoencoder [120] for each task by collecting trajectory demonstrations on the simulated robot. The learned decoder maps the low-dimensional system input $z \in \mathbb{R}^2$ to the high-dimensional robot action $a \in \mathbb{R}^7$ — this decoder is used as the controller ϕ in our experiments. The alignment model $z = f_\theta(h, s)$ aims to identify the correspondence between control inputs z and human inputs h at each state s. Applying a simulated

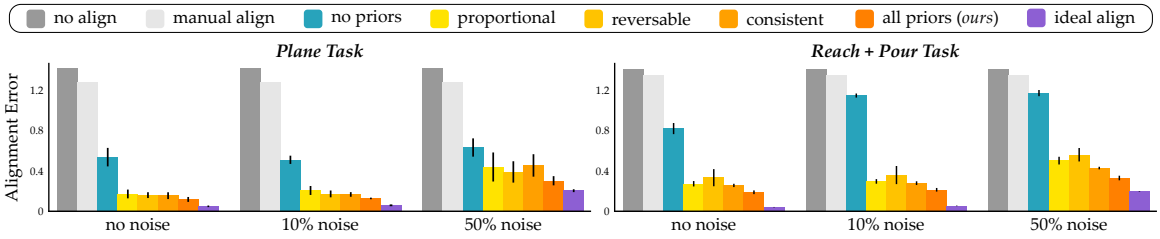


Figure 2.3: Quantitative results from the simulation experiments. We tested three different tasks with increasing complexity, and here we display the results of the easiest (*Plane*) and the hardest (*Reach + Pour*). *Alignment Error* refers to a weighted sum of the relative positional and rotational error in end-effector space. To explore the robustness of our method, we additionally varied how noisy the human was when providing labels. Across different tasks and levels of human noise, including all priors consistently outperformed the other methods, and almost matched an ideal alignment learned from abundant data.

virtual human H_{sim} that is separately defined for each task, we generated data for training and testing using the procedure described in Section 2.4.3.

Tasks. We considered three tasks with increasing levels of complexity; these tasks roughly correspond to our user study tasks shown in Fig. 2.4. Across all tasks the simulated user was given a 2-DoF joystick, such that $h \in \mathbb{R}^2$.

1. *Plane*: The simulated robot arm moves its end-effector in a 2-dimensional horizontal plane. In this task, the simulated human's preference is to use one dimension of the joystick for controlling the movement along the x axis and the other dimension for controlling the movement along the y axis (also see Fig. 2.2).

2. *Pour*: The simulated robot arm moves and rotates its end-effector along the vertical axis. Here the simulated human's preference is to use one dimension of the joystick for controlling the position and to use the other dimension for controlling the rotation of the end-effector.

3. *Reach & Pour*: The simulated robot arm reaches for a bottle and then lifts and rotates the bottle to pour it into a bowl. In this task, the simulated human's preference is divided into two parts, and is based on the previous tasks. For reaching the bottle in the 2D plane, the preference is defined as in the task *Plane*. For pouring, the preference for joystick control is the same as in the task *Pour*.

Data Efficiency. The simulated human provides 10 labeled data samples $\{(s_t^i, h_t^i, s_{t+1}^{*i})_{i=1}^{10}\}$, and the robot collects 1000 unlabeled data samples $\{(s_t^j, s_{t+1}^{*j})_{j=1}^{1000}\}$ for the tasks *Plane* and *Pour*. For the sequential task *Reach & Pour*, we have 20 labeled samples and 2000 unlabeled samples.

Model Details. The teleoperation controller in our experiments follows the structure in [120], and the human alignment model is constructed as in Section 2.4.1. Our human alignment model is a Multi-Layer Perceptron (MLP) network with 2 hidden layers. For supervised training, the loss

function we adopt is given by $L = L_{trans} + \lambda L_{rot}$. Let L_{trans} be the L_2 loss between predicted position and ground truth position, and let L_{rot} be the L_2 loss defined on the quaternion representation between predicted and actual rotation. In the *Plane* task, $\lambda = 0$ because rotation is involved: but for the other two tasks, $\lambda = 1$. During semi-supervised training, the loss function is a combination of supervised loss terms and semi-supervised loss terms as described in Eq. (2.5).

Independent Variables. Within each simulated task, we varied (a) the type of alignment model and (b) the noise level of the simulated human oracle H_{sim}. We compared against two baselines that did not learn an alignment: *no align*, where $z = h$, and *manual align*, where we applied an affine transformation that best matched the data across all states. To understand which priors are useful, we also performed an ablation study where the robot learned the mapping with one prior at a time. Finally, we included a *no prior* baseline, which only leveraged supervised training. To test the model robustness to noisy human labels—where the human incorrectly matches h to a—we set the coefficient of variance $\frac{\sigma}{\mu} \in \{0, 0.1, 0.5\}$ for the simulated human.

Dependent Measures. In both our approach and baselines, we measured position and rotation errors. For positions, we computed the *relative* distance error $E_d = \frac{\|x^*_{t+1} - x_{t+1}\|}{\|x^*_{t+1} - x_t\|}$, where x_t is the current end-effector position before movement, $x^*_{t+1} = \Psi(s^*_{t+1})$ is the desired end-effector position corresponding to intended next state s^*_{t+1}, and x_{t+1} is the actual end-effector position the robot reaches after executing action $a_t = \phi(s_t, f_\theta(h_t, s_t))$. We also measured the distance between rotations $E_r = 2\arccos(|\langle q_{t+1}, q^*_{t+1}\rangle|)$, where q_{t+1} and q^*_{t+1} are the quaternion representation of the predicted and ground truth orientation, respectively. For each experiment setting, we reported mean and standard deviation of the performance metrics over 10 total runs.

Hypotheses. We have the following three hypotheses:

H 1. *With abundant labeled data, the alignment model will accurately learn the human's preferences.*

H 2. *Compared to the fully-supervised baseline, our semi-supervised alignment models that leverage intuitive priors will achieve similar performance with far less human data.*

H 3. *Semi-supervised training with proportional, reversible, and consistent priors will outperform alignment models trained with only one of these priors.*

Results & Analysis. The simulations demonstrate that our alignment method successfully learned the human's preference with a limited amount of labeled data. As shown in Fig. 2.3, each of the proposed alignment models significantly outperformed the one-size-fits-all baselines. For models that do not leverage data or learning — i.e., *no align* and *manual align* — the error is significantly higher than learning alternatives. With abundant data and noise-free human annotations, *ideal align* provided the best-case performance: indicating that our parametrization of the alignment model is capable of capturing the human's control preferences.

Of course, in practice the amount of human feedback is limited. We therefore focus on models that learned from only a small number of labeled datapoints ($10 - 20$ examples). Here our proposed

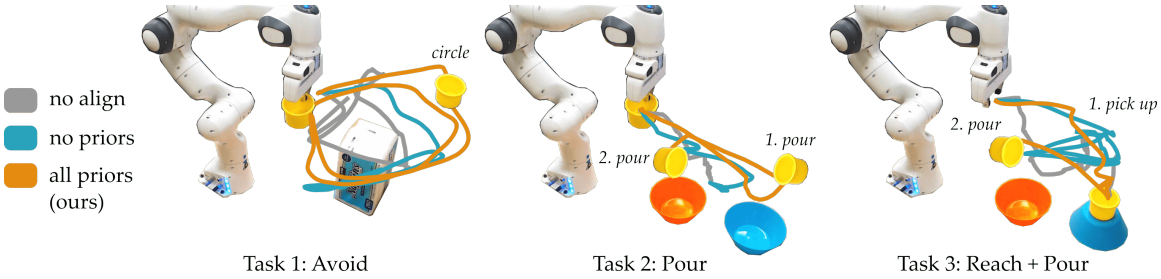


Figure 2.4: Example end-effector trajectories for the *Avoid*, *Pour* and *Reach + Pour* tasks during our user studies. Participants teleoperated the 7-DoF Panda robot arm without any alignment model (*no align*), with an alignment model trained only on their supervised feedback (*no priors*), and with our proposed method, where the robot generalizes the human's feedback using intuitive priors (*all priors*). For both baselines, we can see examples of the human getting confused, counteracting themselves, or failing to complete the task.

priors were critical: semi-supervised models that included at least one intuitive prior performed twice as well as *no priors*, the supervised baseline.

Across different noise levels, our model that leverages *all priors* consistently demonstrated the lowest mean error and standard deviation. This was particularly noticeable when the human oracle is noisy, suggesting that the three priors are indeed *complementary*, and including each of them together brings a performance boost!

Comparing the easier *Plane* task to the more complex *Reach & Pour* task, we also saw that using priors became more important as the task got harder. This suggests that — in complex scenarios — simply relying on a few labeled examples may lead to severe overfitting. On the other hand, our intuitive priors could effectively mitigate this problem.

Summary. Viewed together, the results of our simulations strongly support the hypotheses **H1**, **H2**, and **H3**. Our proposed alignment model successfully learned the mapping between human actions and the system input space (**H1**). In settings with limited labels, our proposed alignment model with intuitive control priors reached results that almost match supervised training with abundant data (**H2**). Finally, in ablation studies, we showed how combining all three proposed priors leads to superior performance and greater training stability than training with a single prior (**H3**).

2.5.2 User Studies

Within this section we show the results of a user study that evaluates our framework across three robot manipulation tasks based on assistive settings. Participants teleoperated a 7-DoF robotic arm (Panda, Franka Emika) through a handheld 2-axis joystick controller. As in our simulations, we use [120] to learn a decoder that enables low-DoF control over the high-DoF robot arm. The robot learned the user's individual preferences for how this controller should behave by asking for a limited number of examples and then generalizing with our intuitive priors.

Tasks. Similar to our simulation experiments, we considered three different tasks. These tasks are visualized in Fig. 2.4.

1. *Avoid*: The Panda robot arm moves its end-effector within a 2-dimensional horizontal plane. Users are asked to guide the robot around an obstacle without colliding with it. The task ended once users completed one clockwise rotation followed by one counter-clockwise rotation.

2. *Pour*: The Panda robot arm is holding a cup, and users want to pour this cup into two bowls. Users are asked to first pour into the farther bowl, before moving the cup back to the start and pouring into the closer bowl.

3. *Reach & Pour*: This is the most complex task. Users start by guiding the robot towards a cup and then pick it up. Once the users reach and grasp the cup, they are asked to take the cup to a target bowl, and finally pour into it.

Independent and Dependent Variables. For each of the task described above, we compared three different alignment models: making no adjustments to the original controller (*no align*), an alignment model trained just using the human's labeled data (*no priors*), and an alignment model trained using our proposed semi-supervised approach (*all priors*). Our semi-supervised learning model should generalize the human's preferences by enforcing intuitive priors such as proportionality, reversibility, and consistency.

To evaluate the effectiveness of these different alignment strategies, we recorded quantitative measures including task completion time and trajectory length. We also calculated the percentage of the time that users *undo* their actions by significantly changing the joystick direction — undoing suggests that the alignment is not quite right, and the human is still adapting to the robot's control strategy. Besides these objective measures, we also collected subjective feedback from the participants through 7-point Likert scale surveys.

Hypothesis. An alignment model learned from user-specific feedback and generalized through intuitive priors makes it easier for humans to control the robot and perform assistive manipulation tasks.

Experimental Procedures. We recruited 10 volunteers that provided informed written consent (3 female, ages 23.7 ± 1.5). Participants used a 2-axis joystick to teleoperate the 7-DoF robot arm, and completed three manipulation tasks inspired by assistive settings. At the start of each task, we showed the user a set of robot movements and ask them to provide their preferred input on the joystick — i.e., "if you wanted the robot to perform the movement you just saw, what joystick input would you provide?" Users answered 7 queries for task *Avoid*, 10 queries for task *Pour* and 30 queries for task *Reach & Pour*. After the queries finished, the users started performing tasks sequentially using each of the alignment strategies. The order of alignment strategies was counterbalanced.

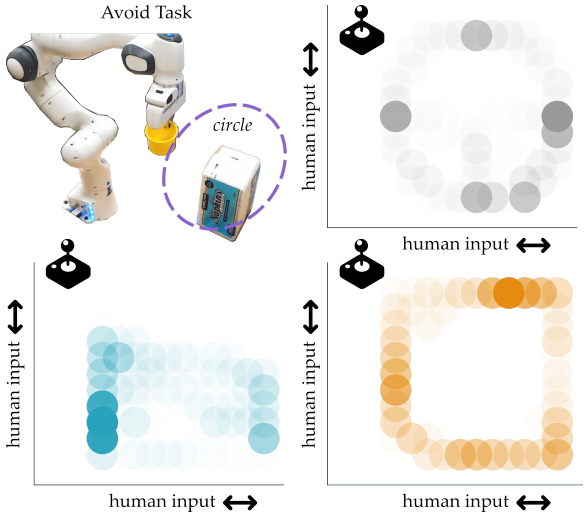


Figure 2.5: Heatmaps of the participants' joystick inputs during task *Avoid*. For *no align* in the upper right, people primarily used the cardinal directions. For *no priors* in the bottom left, the joystick inputs were not clearly separated, and no clear pattern was established. For our *all priors* model on the bottom right, however, we observed that the human inputs were *evenly distributed*. This indicates that the users smoothly completed the task by continuously manipulating the joystick in the range $[-1, +1]$ along both axes.

Results & Analysis. The objective results of our user study are summarized in Fig. 2.6. Across tasks and metrics, our model with *all priors* outperforms the two baselines. In addition, our model not only has the best average performance, but it also demonstrates the least variance. Similar to our results in simulation, when the task is difficult (i.e., *Reach & Pour*), the performance of the align model trained with only supervised loss and *no priors* drops significantly compared to simpler tasks, reinforcing the importance of priors in learning complex alignment models!

We also illustrate our survey responses in Fig. 2.7. Across the board, we found that users exhibited a clear preference for our proposed method. Specifically, they perceived *all priors* as resulting in better alignment, more natural, accurate, and effortless control, and would elect to use it again. These subjective results highlight the importance of personalization when controlling high-DoF systems — we contrast these results to [120], where participants perceived the unaligned controller as confusing and unintuitive.

To better visualize the user experiences, we also display example robot end-effector trajectories from one of the participants in Fig. 2.4. Here we observe that the trajectories of our model (in orange) are smooth and do not detour during the tasks, while the trajectories for *no align* (in grey) and *no priors* (in blue) have many movements that counteract themselves, indicating that this user was struggling to understand and align with the control strategy. In the worst case, participants were unable to complete the task with the *no priors* model (see the *Avoid* task in Fig. 2.4) because no joystick inputs mapped to their intended direction, effectively causing them to get stuck at

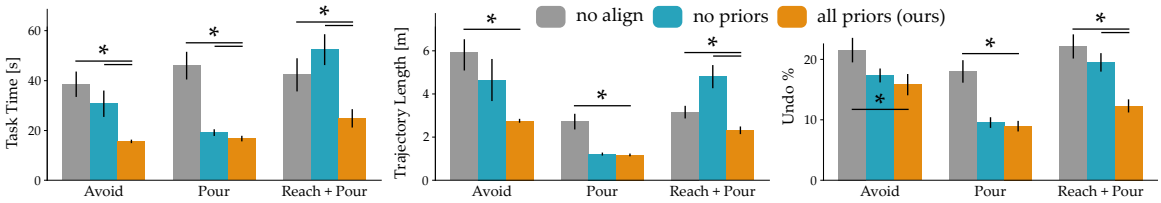


Figure 2.6: Objective results from our user study. Left: Average time taken to complete each task. Middle: Average trajectory length as measured in end-effector space. Right: Percentage of the time people spend undoing their actions. Error bars show the standard deviation across the 10 participants, and colors match Fig. 2.4. Asterisks denote statistically significant pairwise comparisons between the two marked strategies ($p < .05$).

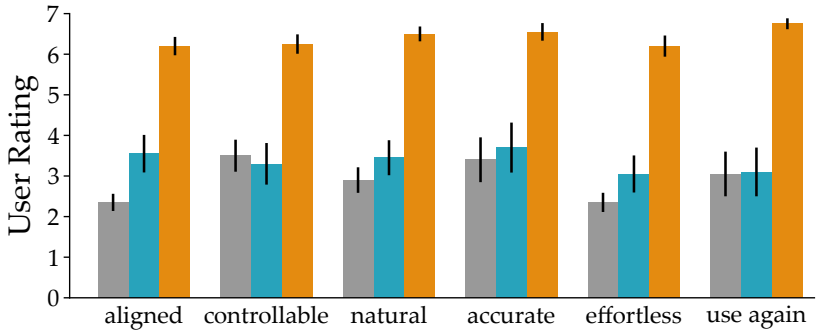


Figure 2.7: Results from our 7-point Likert-scale survey. Higher ratings indicate agreement. Users thought that our learned model with intuitive priors aligned with their preferences, was easy to control, and improved efficiency — plus they would choose to use it again! Pairwise comparisons between our approach and the baselines are statistically significant across the board.

undesirable states.

To further validate that our model is learning the human's preferences, we illustrate heatmaps over user inputs for the *Avoid* task in Fig. 2.5. Recall that this task requires moving the robot around an obstacle. Without the correct alignment, users default to the four cardinal directions (*no align*), or warped circle-like motions (*no priors*). By contrast, under *all priors* the users smoothly moved the joystick around an even distribution, taking full advantage of the joystick's $[-1, +1]$ workspace along both axes.

Summary. Taken together, these objective measurements as well as subjective results empirically support our hypothesis. Using the alignment model learned with our intuitive priors, users are able to complete the manipulation tasks more efficiently, and in a way that matches their personal preferences.

2.6 Conclusion and Discussion

Summary. We developed a framework for learning personalized mappings between human inputs and robot actions. Since no two humans are the same, we doubt that any one-size-fits-all approach

will work well with everyday users. But asking humans about their preferences — and getting their feedback in every situation — is prohibitively time consuming. We therefore proposed a semi-supervised approach, where the robot has access to a few examples of the human's desired mapping, and must generalize that mapping across unlabeled data. We achieve this generalization by recognizing that humans have strong *priors* over how controllers should operate: we expect the input space to be proportional, reversible, and consistent. By incorporating these priors, our proposed approach learned the human's preferences from a few queries. Importantly, this proposed method does not affect the system controller itself, is lightweight and agnostic to the underlying robot dynamics, and does not require any additional hardware or software for intent recognition.

Limitations and Future Work. The robot currently queries the human at random states. Future work will incorporate active learning, so that the robot intelligently selects informative states to ask for human preferences.

Chapter 3

Learning Human Objectives from Sequences of Physical Corrections

3.1 Introduction

Imagine that you just got back from the store, and a two-armed personal robot is helping you unpack a bag of groceries (see Fig. 3.1). You don't want this robot to bump the bag into any cabinets or the hat on the left, but you also don't want the robot to stretch and rip the bag, or squeeze it and crush your groceries. Out of the corner of your eye, you notice that the robot is heading towards a side of the table that is wet (the blue region in the figure). So you *physically correct* it — pushing, pulling, or twisting the robot's arms to guide it away from that region and move it toward the green region on the left. Importantly, you can only physically interact with one arm at a time, since it's difficult to pay attention to how to correct both arms simultaneously: in the process of guiding this arm away from the obstacle, you might inadvertently move both arms closer together, squeezing the bag. Teaching this robot about all your preferences requires more than just one physical correction. You must provide a *sequence*: alternating between fixing the position of the arm closest to the obstacle, and adjusting the other arm so that the bag is held correctly.

State-of-the-art methods *learn* from physical corrections by treating each interaction as an *independent* event [16, 15, 118, 27]. These works assume that the human makes corrections based only on their objective, without considering the other corrections they have already made or are planning to provide. But if we view human corrections as isolated events, we can misinterpret what they convey: for instance, when the human moves one arm away from the hat and closer to the other arm, this robot will mistakenly learn that the human wants to squeeze the bag.

At the other end of the spectrum, robots can learn by looking at the *final* trajectory collectively produced by all of the human's corrections [4, 11, 144, 152, 85]. There are two issues with this: i)

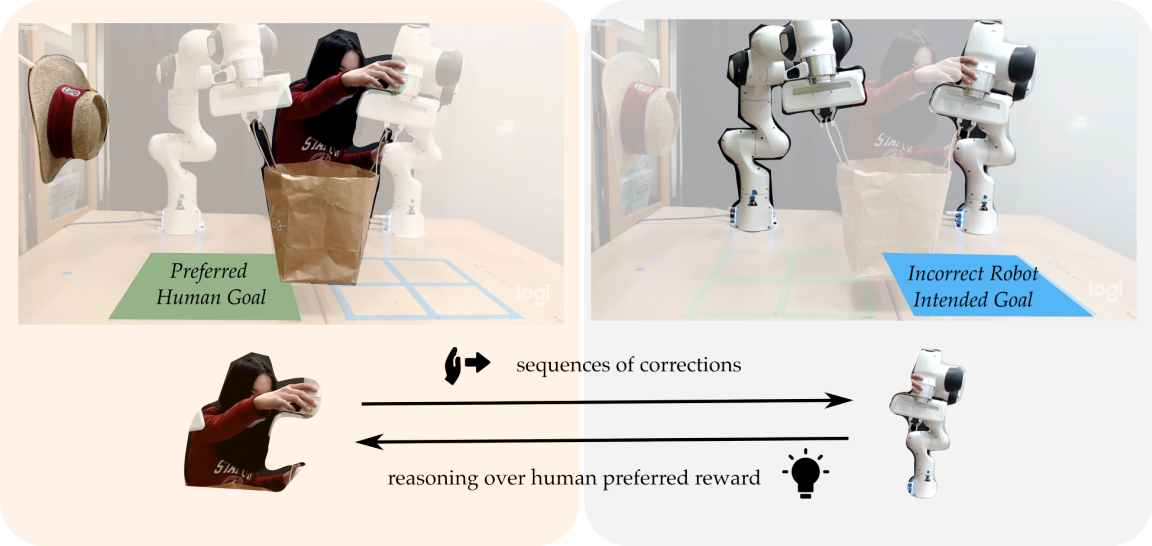


Figure 3.1: Learning from physical sequences. Two robot arms are carrying a grocery bag toward an undesirable wet region, blue region on the right. The human provides a sequence of physical corrections to guide the robot toward their preferred objective, i.e., placing the bag on the green region while also avoiding the obstacles on the left, and holding the bag upright without squeezing or stretching it.

the robot does not learn or update its behavior until after the entire task is over, and ii) even the final trajectory may not capture what the human really wants. Returning to our example: because the user can only interact with one arm at a time, the final trajectory has some parts where the distance to the obstacle is right, and other sections where the bag is held correctly — but the final trajectory fails to capture both throughout.

At their core, prior works miss out on part of the process that humans use to correct the robot's behavior. Not everything can be fixed at once, or even fixed perfectly:

Humans corrections are not independent events — we often use multiple correlated interactions to correct the robot.

We leverage this insight to learn the human's reward function online from sequences of physical corrections, *without assuming* that human corrections are conditionally independent. Let's jump back to our example: a robot that reasons over the sequence recognizes that *collectively* the human corrections keep the robot away from the obstacle while holding the bag, even though the corrections *individually* fail to convey this objective, and can even be counter-productive.

Overall, we make the following contributions:

Capturing Conditional Dependence. We enable robots to learn from sequences of corrections by introducing an auxiliary reward function. This reward captures the human's trade-off between making corrections that increase the short-term reward (i.e., avoiding the obstacle) and reaching their long-term objective (i.e., carrying groceries).

Learning from Sequences. We introduce a tractable method to learn from a sequence of physical corrections by i) using the Laplace approximation to estimate the partition function and ii) solving a mixed-integer optimization problem.

Conducting Online and In-Person User Studies. Participants interacted with robots in both single- and multi-agent environments. We recorded user's corrections, and compared our approach to both *independent* and *final* baselines. Our proposed method outperforms both baselines, demonstrating the effectiveness of reasoning over sequences.

3.2 Related Work

Physical Human-Robot Interaction. When humans and robots share a workspace, physical interaction is *inevitable*. Prior work studies how robots can safely respond to physical interactions [72, 43, 83]. This includes impedance control [78] and other reactive strategies [71]. Most relevant to our setting is *shared control* [110, 117, 2, 130], where the human and robot arbitrate between leader and follower roles. Although shared control enables the user to temporarily correct the robot's motion, it does not alter the robot's long term behavior: the robot does not *learn* from physical corrections.

Learning from Corrections (Online). Recent research recognizes that physical human corrections are often *intentional*, and therefore informative [16, 15, 118, 27]. These works learn about the human's underlying objective in real-time by comparing the current correction to the robot's previous behavior. Importantly, each correction is treated as an *independent* event. Outside of physical human-robot interaction, shared autonomy follows a similar learning scheme — the robot uses human inputs to update its understanding of the human's goal online, but does not reason about the connections between multiple interactions [88, 51, 87, 89]. We build upon these prior works by learning from sequences of corrections.

Learning from Corrections (Offline). By contrast, other works learn from the *final* trajectory produced after all the corrections are complete. This research is closely related to learning from demonstrations [11]. For example, in [4] the human corrects keyframes along the robot's trajectory, so that the next time the robot encounters the same task it moves through the corrected keyframes. Most similar to our setting are [85] and [152], where the robot iteratively updates its understanding of the human's objective *after* the human corrects the robot's entire trajectory. Although these works take multiple corrections into account, they do so offline, and are not helpful during the current task.

3.3 Formalizing Sequences of Physical Corrections

In this section we formalize a physical human-robot interaction setting where one or more robots are performing a task incorrectly. The human expert knows how these robots should behave, and physically corrects the robots to convey the true objective. But the human doesn't interact just once: the human may need to interact *multiple times* in order to correct the robots. Our goal is for these robots to learn the human's objective from this sequence of physical corrections.

3.3.1 Task Formulation

We formulate our problem as a discrete-time Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma, \rho_0)$. Here $\mathcal{S} \subseteq \mathbb{R}^n$ is the robot state space, $\mathcal{A} \subseteq \mathbb{R}^m$ is the robot action space, $\mathcal{T}(s, a)$ is the transition probability function, r is the reward, γ is the discount factor, and ρ_0 is the initial distribution.

Reward. Let the robot start from a state s^0 at time $t = 0$. As the robot completes the task it follows a trajectory of states: $\xi = \{s^0, s^1, \ldots, s^T\} \in \Xi$. The human has in mind a trajectory that they prefer for the robot to follow. Recall our motivating example — here the human wants the robot arms to follow a trajectory that avoids the cabinets without squashing the bag. Similar to prior work [200, 1, 144, 90], we capture this objective through our reward function: $R(\xi; \theta) = \theta \cdot \Phi(\xi)$. Here Φ denotes the feature counts over trajectory ξ, and θ captures how important each feature is to the human. We let ξ_R^t denote the robot's trajectory at timestep t, and we let θ^t denote the robot's current reward weights.

Suboptimal Initial Trajectory. The system of one or more robots starts off with an initial reward function $R(\xi; \theta^0)$, and optimizes this reward function to produce its initial trajectory.

$$\xi_R^0 = \arg\min_{\xi \in \Xi} \theta^0 \cdot \Phi(\xi)$$

But this initial trajectory ξ_R^0 misses out on what the human really wants — going back to our example, the robot does not realize that the blue region is wet and it needs to place the bag on the green region. More formally, the robot's estimated reward function (which is parameterized by θ^0) does not match the human's preferred reward function (parameterized by the true weights θ^*).

Human Corrections. The robot learns about the human's reward — i.e., the true reward weights — from physical corrections. Intuitively, these corrections are applied forces and torques which push, twist, and guide the robots. To formulate these interactions we must revise our problem definition: let a_R be the robot's action and let a_H be the human's *correction*. In practice, both a_R and a_H could be applied joint torques [16, 15]. Now the overall system transitions based on both human and robot actions: $s^{t+1} = \mathcal{T}(s^t, a_R + a_H)$. We use $A_H = \{(t_i, a_H^i), i = 1, \ldots, K\}$ to denote a *sequence* of K ordered human corrections a_H^i at time step t_i, where i keeps track of order of the corrections. Our goal is to learn the human's true reward weights from the sequence of corrections A_H.

3.3.2 Physical Corrections as Observations

When robots are performing a task suboptimally, the human expert intervenes to correct those robots towards the right behavior. Going back to our example from Fig. 3.1. The user sees that the robot is making a mistake (moving towards the wet blue region), and physically intervenes. In the process of fixing this first issue, the human is forced to create another problem: by moving the first robot arm away from the blue region, they also move both arms closer together, and start to squash the bag. We note two important characteristics of these corrections: i) each human correction is intentional, and conveys information about the human's objective, but ii) the corrections viewed together may provide more information than isolating each interaction.

Leveraging these corrections, our goal is to find a better estimate of the reward parameters $P(\theta \mid A_H, \xi_R^0)$. We start by applying Bayes' rule:

$$P(\theta \mid A_H, \xi_R^0) \propto P(\theta) P(A_H \mid \xi_R^0, \theta) \tag{3.1}$$

In line with prior work [16, 15, 118, 27], we will model $P(A_H|\xi_R^0, \theta)$ by mapping each human correction to a *preferred trajectory*. Given the human's correction (t_i, a_H^i), we deform the robot's trajectory to reach ξ_H^i.

One simple example of this is to let the robot execute $a_R^{t_i} + a_H^{t_i}$ at this time step t_i, and stick to its original action plan a_R^t for future time steps $t > t_i$. More generally, we propagate the human's applied correction along the robot's current trajectory [119]:

$$\begin{aligned} \xi_H^1 &= \xi_R^0 + \mu A^{-1} a_H^1 \\ \xi_H^i &= \xi_H^{i-1} + \mu A^{-1} a_H^i, \quad i \in \{2, \dots, K\} \end{aligned} \tag{3.2}$$

Consistent with [119] and [49], μ and A are hyperparameters that determine the deformation shape and size. We emphasize that here the robot is not yet learning — instead, it is locally modifying its trajectory in the direction of the applied correction. Within our motivating example, let the human apply a force pushing the first robot arm away from the blue region. Equation (3.2) maps this correction to ξ_H, a trajectory that moves the robot arm farther from the blue region than ξ_R. In Fig. 3.2, we demonstrate how a sequence of corrections lead to a sequence of trajectories that enable the robot to correct its path and reach the preferred goals.

Now that we have this tool for mapping corrections to preferred trajectories, we can rewrite Equation (3.1):

$$\begin{aligned} P(\theta \mid A_H, \xi_R^0) &\propto P(\theta) P(A_H \mid \xi_R^0, \theta) \\ &= P(\theta) P\Big((t_1, a_H^1), \dots, (t_K, a_H^K) \mid \xi_R^0, \theta\Big) \\ &\approx P(\theta) P(\xi_H^1, \dots, \xi_H^K \mid \xi_R^0, \theta) \end{aligned} \tag{3.3}$$

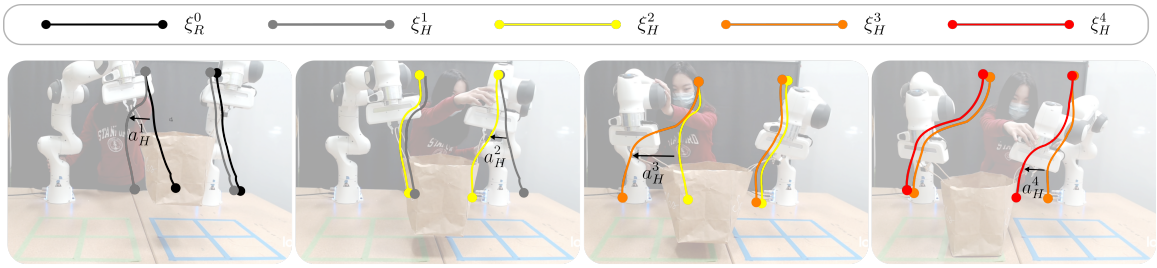


Figure 3.2: An example of a sequence of human corrections along with her corresponding correction trajectories $\xi_H^1, \xi_H^2, \xi_H^3, \xi_H^4$ to guide the robot to place the grocery bag on the green region while avoiding any stretching or squeezing of the bag.

Here $P(\theta)$ is the robot's prior over the human's objective, and $P(\xi_H^1, \ldots, \xi_H^K \mid \xi_R^0, \theta)$ is the likelihood that the human provides a specific *sequence* of preferred trajectories given the robot's initial behavior ξ_R^0 and the reward weights θ.

3.4 Learning from Sequences of Physical Corrections

In the previous section we outlined how robots can learn from physical corrections using Equation (3.3). However, we still do not know how to evaluate $P(\xi_H^1, \ldots, \xi_H^K \mid \xi_R^0, \theta)$, which captures the relationship between a sequence of human corrections and the human's underlying objective. Prior work [16, 15, 118, 27] has avoided this problem by assuming that the human's corrections are *conditionally independent*:

$$P(\xi_H^1, \ldots, \xi_H^K \mid \xi_R^0, \theta) = \prod_{t=1}^{K} P(\xi_H^t \mid \xi_R^t, \theta) \tag{3.4}$$

Intuitively, this assumption means that there is no relationship between the human's previous corrections and their current correction. But we know this is not always the case — think about our motivating example, where the human's corrections are intricately coupled! Accordingly, here we propose a method for evaluating $P(\xi_H^1, \ldots, \xi_H^K \mid \xi_R^0, \theta)$ *without* assuming conditional independence.

3.4.1 Reasoning over Sequences of Physical Corrections

To learn from sequences of human corrections online, we introduce an auxiliary reward function. In this section we also describe the modeling assumptions made by this auxiliary reward function, as well as an algorithm for leveraging this function for real-time inference.

Accumulated Evidence. We start by introducing the auxiliary reward function: $D(\xi_H^1, \ldots, \xi_H^K, \theta)$. Let's refer to D as the *accumulated evidence* of a sequence of preferred trajectories $\xi_H^1, \ldots, \xi_H^K$ under reward parameter θ. We hypothesize that the accumulated evidence should **(a)** reward not only the

behavior of the final trajectory after all corrections, but also the intermediate trajectories the robot follows during the sequence of corrections. This recognizes that — when users make corrections — they don't sacrifice long-term reward for short-term failure. Consider our motivating example: the human is not willing to correct the robot into crushing the bag, even if that will reduce the overall number of corrections needed to avoid the obstacle. Of course, **(b)** humans also try to minimize their overall effort when making corrections — and any rewards for which the human's corrections are redundant or unnecessary are therefore not likely to be the human's true reward. Combining these two terms:

$$D\big(\xi_H^1, \dots, \xi_H^k, \theta\big) = \sum_{t=1}^{K} \alpha^{K-t} R(\xi_H^t, \theta) - \gamma\Big(\sum_{t=1}^{K} \|a_H^t\|^2\Big) \tag{3.5}$$

α and γ are two hyperparameters decaying the importance of previous corrections, and determining the relative trade-off between intermediate reward and human effort respectively.

Learning Rule. Similar to prior work in inverse reinforcement learning [200, 151, 144, 90], we model humans as noisily rational agents whose corrections maximize accumulated evidence for their preferred θ:

$$P(\xi_H^1, \dots, \xi_H^K \mid \xi_R^0, \theta) \propto \exp\Big(D\big(\xi_H^1, \dots, \xi_H^k, \theta\big)\Big) \tag{3.6}$$

Equation (3.6) expresses our likelihood function for learning from human corrections. Using Laplace's method (as applied in [51]), we can approximate the normalization factor:

$$\begin{aligned} P(\xi_H^1, \dots, \xi_H^K \mid \theta) &= \frac{\exp\Big(D\big(\xi_H^1, \dots, \xi_H^k, \theta\big)\Big)}{\int_{\xi_H^1, \dots, \xi_H^K} \exp\Big(D\big(\xi_H^1, \dots, \xi_H^k, \theta\big)\Big) d\xi_H^1 \dots d\xi_H^K} \\ &\approx \frac{\exp\Big(D\big(\xi_H^1, \dots, \xi_H^k, \theta\big)\Big)}{\exp\Big(\max_{\xi_H^1, \dots, \xi_H^K} D\big(\xi_H^1, \dots, \xi_H^k, \theta\big)\Big)}. \end{aligned} \tag{3.7}$$

Monte-Carlo Mixed-Integer Optimization. Inspecting the denominator of Equation (3.7), we see that — in order to evaluate the likelihood of a sequence of corrections — we need to find the *highest possible* accumulated evidence the human *could* have achieved given that their objective is θ. Put another way, we need to search for the sequence of K corrections that maximize Equation (3.5) under θ:

$$\begin{aligned} D_K^*(\theta) &= \max_{\xi_H^1, \dots, \xi_H^K} D\big(\xi_H^1, \dots, \xi_H^k, \theta\big) \\ &= \max_{(t_1, a_H^1), \dots, (t_K, a_H^K)} D\big(\xi_H^1, \dots, \xi_H^k, \theta\big) \end{aligned} \tag{3.8}$$

Unfortunately, solving Equation (3.8) is hard because we need to figure out both *when* to make each correction $(t_1, \dots, t_K)$, which is a discrete decision, as well as *what* to correct during each interaction

Algorithm 1: Monte-Carlo Mixed-Integer Program

Output: Maximum accumulated evidence $D_K^*(\theta)$, and K optimized human corrections $(t_1, a_H^1), \ldots, (t_K, a_H^K)$

Input: The suboptimal initial robot plan ξ_R^0, reward parameter θ, and hyperparmeters in Eqn. (3.5).
$G(t_1, \ldots, t_K)$ is a continuous optimizer.

Initialize maximum accumulated evidence and correction times: $D_K^* = -\infty$, $T = \emptyset$.
for $i \leftarrow 0$ **to** T_{max} **do**
  while $(t_1, \ldots, t_K) \in T$ **do**
    Randomly sample $(t_1, \ldots, t_K)$
  end
  $T = T \cup \{(t_1, \ldots, t_K)\}$
  $D_i\ ,\ (a_1, \ldots, a_K) = G(t_1, \ldots, t_K, \xi_R^0, \theta)$
  if $D_i > D_K^*$ **then**
    $D_K^* = D_i$
    $T_H^* = (t_1, \ldots, t_K)$
    $A_H^* = (a_1, \ldots, a_K)$
  end
end
return D_K^*, T_H^*, A_H^*

$(a_H^1, \ldots, a_H^K)$, which is a continuous action.

To tackle this mixed-integer optimization problem, we develop a Monte-Carlo mixed-integer optimization method, where we first randomly sample discrete $(t_1, \ldots, t_K)$, and then solve $(a_H^1, \ldots, a_H^K)$ with gradient-based continuous optimization methods. The full pipeline for the optimization is summarized in Algorithm 1.

Importantly, the inputs to this optimization are only the robot's initial trajectory ξ_R^0, the reward parameter θ, and our model hyperparameters. Hence, we can conduct *offline* optimization to solve Equation (3.8) for several sampled values of θ. We then use the resulting library of stored D^* values to learn online, during physical interaction.

Online Inference. We have a way for solving for the denominator in Equation (3.7) offline. What remains to be evaluated is the numerator $\exp\Big(D\big(\xi_H^1, \ldots, \xi_H^k, \theta\big)\Big)$. This can be easily computed online when the human is physically correcting the robot. Thus, we can now evaluate $P(\xi_H^1, \ldots, \xi_H^K \mid \xi_R^0, \theta)$ while accounting for the relationships between corrections.

Our overall pipeline is as follows. Offline, we compute $D_K^*(\theta)$ with Alg. 1. Online, the robot starts by following the optimal trajectory ξ_R^0 with respect to its prior over θ. However, this initial reward might not capture the human reward and thus the human physically corrects the robot. The robot would then perform inference and update its belief over the reward when it receives new human corrections. At time t, the human has provided a total of K corrections $(t_1, a_H^1), \ldots, (t_K, a_H^K)$, where $t_1 < t_2 < \cdots < t_K \leq t$. We propagate these human corrections to get the deformed trajectories $\xi_H^1, \ldots, \xi_H^K$ with Equation (3.2). We then perform Bayesian inference by solving Equation (3.7) and

plugging the likelihood function into Equation (3.3). Finally, the robot uses its new understanding of θ to solve for an updated optimal trajectory ξ_R^t.

3.4.2 Relation to Prior Works: Independent & Final Baselines

To evaluate the effectiveness of our proposed method that reasons over correction sequences, we compare against two baselines: *Independent* and *Final*.

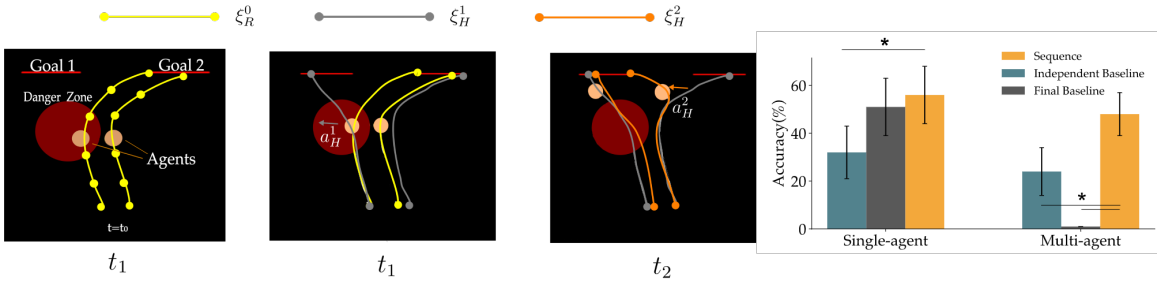


Figure 3.3: Navigation Simulation. We show the sequence of corrections in the multi-agent task, and the accuracy results of the single- and multi-agent tasks.

Independent Baseline (Online). This baseline follows the same formalism as our approach, but assumes each human correction is conditionally independent. Hence, here we use Equation (3.4) to learn from each correction separately. The likelihood of observing an individual correction is related to the reward and effort associated with that correction [16]:

$$P(\xi_H^i \mid \xi_R^i, \theta) \propto \exp\left(R(\xi_H^i, \theta) - \gamma \|\xi_H^i - \xi_R^i\|^2\right) \tag{3.9}$$

Final Baseline (Offline). At the other end of the spectrum, we can always look at the final trajectory when all corrections are finished [85]. In this case, the robot learns by comparing the final trajectory, ξ_H^K, to the initial trajectory, ξ_R^0:

$$P(\xi_H^K \mid \xi_R^0, \theta) \propto \exp\left(R(\xi_H^K, \theta) - \gamma \|\xi_H^K - \xi_R^0\|^2\right) \tag{3.10}$$

To summarize, both our method and these baselines use a Bayesian inference approach. The difference is the likelihood function: *Independent* assumes conditional independence, while *Final* only considers the initial and final trajectory.

3.5 Experiments

To test our proposed algorithm, we conduct experiments with human users in a simulated navigation task and a robot manipulation task. We will discuss details that are consistent in both tasks and then elaborate on each one respectively.

Tasks. Our two tasks are: a web-based online simulated navigation task and an in-person robot manipulation task.

1. *Navigation Simulation*: In this task, a team of robots are navigating together through a specified region as in Fig. 3.3. The robot's objective function considered four features related to: reaching the goal, keeping formation, avoiding the danger zone, and minimum travel distance. We test our algorithm and baselines in different scenarios by varying robot team size, robot initial policy, and specifying different human preference reward values.

2. *Robot Manipulation*: In this task, two robot arms are carrying a full grocery bag to place it on the table. There are four features concerned in the reward: reaching the goal basket (blue or green regions as shown in Fig. 3.1), keeping the groceries inside of the bag while avoiding squeezing or stretching the bag, avoid touching nearby housewares such as cabinets or the hat shown in Fig. 3.1, and minimum trajectory length for efficiency. The robot starts off with the assumption of reaching an incorrect goal while also not realizing the bag is full, and should not be stretched or squeezed. Users apply forces to guide the two robot arms toward the correct goal region, while trying to keep the groceries inside of the bag.

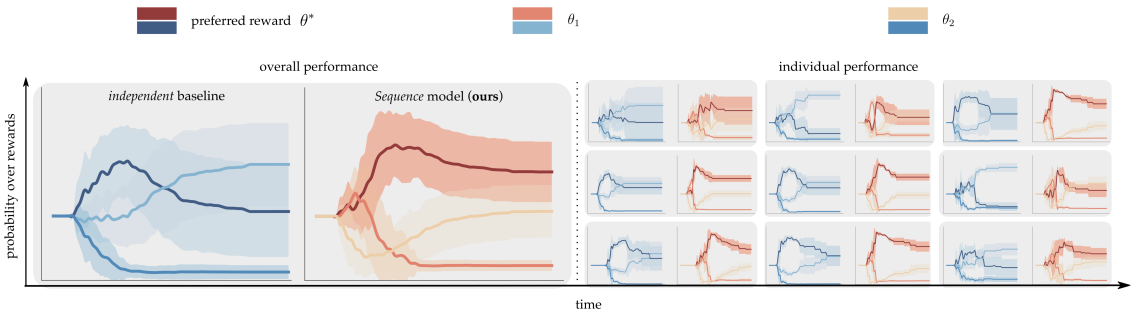


Figure 3.4: Likelihood of three different reward parameters $(\theta^*, \theta_1, \theta_2)$ as more corrections are received over time. The preferred reward θ^* is shown with the darker red or blue. Each pair of plots demonstrate the *Sequence* model compared with the *Independent* model. We demonstrate the aggregate results over all users on the left, and the individual performance on the right. As shown *Sequence* outperforms *Independent* in identifying the preferred reward θ^*.

Independent Variables. We compared three different inference models: reasoning over corrections independently (*Independent*), performing inference only based on the final correction (*Final*), and our model that reasons over the sequence of corrections (*Sequence*). *Independent* and our

Sequence model can perform online inference, while *Final* will conduct offline inference after all corrections are provided.

Dependent Measures. We conduct experiments with human users and evaluate the effectiveness of the models by measuring the inference accuracy. Since we are unable to measure users' internal reward, we specify users' preferred reward function out of a predefined finite set of candidate reward parameters. We convey the preferred reward by explaining the priorities of the features to the user and demonstrating a desired robot trajectory using the preferred reward. Users are instructed to correct the robot to behave as optimally as possible, while minimizing their physical correction effort.

Hypotheses.

H 4. *Compared to the online* Independent *baseline, reasoning over* Sequences *of corrections leads to higher accuracy and faster convergence to the preferred reward.*

H 5. *Compared to the offline* Final *baseline, in addition to the advantages of online inference, our* Sequence *model achieves higher accuracy in challenging tasks – specifically tasks where fully correcting the robots is infeasible.*

3.5.1 Navigation Simulation

Experimental Details. We recruited 15 participants for two simulated navigation tasks. Participants interact with point-mass robots using a web browser, where they can observe the robots' trajectories, select a robot to correct, and provide corrections using the arrow keys. We collected data from humans for 5 episodes in two different scenarios: a simple single-agent scenario with only one robot, and a more complex scenario containing a two-robot team. In both settings, participants are only able to correct one robot at a time.

In both scenarios, robots' initial trajectory goes to an incorrect goal region as shown in Fig. 3.3. In the human's preferred reward function, not only is going to the correct goal encouraged, but other features are also encoded, including avoiding a danger zone and keeping the formation (equal distance between the robots throughout the trajectories).

Results & Analysis. We compute the average inference accuracy for the two baseline methods and our method across all users in both scenarios. The results are shown in Fig. 3.3.

Across both scenarios, our *Sequence* model demonstrates leading performance compared to both the *Independent* and the *Final* baselines. One thing to note is that in the single-agent scenario, the *Final* method has a comparable accuracy to our method. This indicates that in this simple task, the user can correct the robot to behave almost optimally so that simply looking at the final trajectory conveys sufficient information about the preferred reward. While in a more complex multi-agent scenario, the performance of the *Final* method drops significantly . This is because in this complex scenario, even if the user acts optimally, the final trajectory will still not have sufficient information

to identify the preferred reward. In this example reasoning over sequences is dominant over only considering the final correction.

3.5.2 Robot Manipulation

Experimental Details. We recruited 9 participants for the in-person robot manipulation task (Fig. 3.1). Here, the two Franka Emika Panda robot arms are carrying a grocery bag to the table. The participants are asked to physically push or pull the robot to correct its behavior. Similar to the navigation task, the user can only correct one robot at a time, and we collected 5 episodes of corrections from each participant.

As shown in Fig. 3.1, there are two containers on the table for the grocery bag (the blue region and the green region). The robots' initial plan is to carry the bag to the right toward the blue region. However, the human wants to put the grocery to the left container. Meanwhile, since the grocery bag is almost full, in order to keep the groceries from falling out, the participants are also instructed not to squeeze or stretch the bag. In this setting there are three possible reward parameter θ, and the robot tries to infer the correct reward parameter θ^* from the human corrections.

Results & Analysis. We calculate the inference accuracy for all the three models (*Independent*, *Final*, and *Sequence*), and summarize the results in Table 3.1. Our method demonstrates superior performance compared to the baselines.

Table 3.1: Inference accuracy over 9 participants for robot manipulation.

	Sequence (**ours**)	*Independent*	*Final*
accuracy (%)	$\mathbf{82.22 \pm 21.99}$	31.11 ± 26.99	53.33 ± 13.30

In addition, we illustrate the probability distribution over θ with time in Fig. 3.4. Since the *Final* baseline performs the inference offline, we only visualize our *Sequence* model and the *Independent* baseline. As can be seen, across all of the participants, the probability for the preferred reward consistently dominates the other candidate rewards. However, for the *Independent* baseline, even if the probability for the preferred reward sometimes starts off high, it ends up not being the most likely reward as we receive more corrections. This is because in this two-robot task, redirecting the system to the correct goal while simultaneously maintaining the shape of the bag (formation) is not possible. The best possible corrections are: push one arm towards the goal, while stretching or squeezing the bag by a small amount, and then push the other arm in the same direction so that the bag shape is recovered. However, if we reason over these corrections independently, the robot will think that the first push indicates that preserving the bag shape is not important, and this leads to an incorrect inference.

3.5.3 Summary

Our results empirically support both of our hypotheses **H4** and **H5**. Our *Sequence* model not only conducts an online inference, but also demonstrates superior performance specially in complex multi-agent tasks.

3.6 Conclusion and Discussion

Summary. We developed a framework for learning from sequences of user corrections during physical human-robot interaction. We introduced an auxiliary reward that models the connections between corrections, and leveraged mixed-integer programming to solve for the best possible sequence of corrections. Our results from online and in-person users demonstrate that our approach outperforms methods that take each correction independently, or wait for the final trajectory.

Limitations and Future Work. While our framework outperforms independent baselines across the users, we acknowledge that our experiments were conducted using only a discrete set of candidate reward parameters. As the number of candidate parameters increases, the inference time for our approach would also increase linearly, which can limit its scalability. Additionally, dealing with a distribution over continuous reward parameters presents further challenge that we intend to explore in future work.

Chapter 4

Influencing Leading and Following in Human-Robot Teams

4.1 Introduction

Humans are capable of seamlessly interacting and collaborating with each other. They can easily form teams and decide if they should follow or lead to efficiently complete a task as a group. This is apparent in sports teams, human driving behavior, or simply having two people move a table together. Similarly, humans and robots are expected to seamlessly interact with each other to achieve collaborative tasks. Examples include collaborative manufacturing, search and rescue missions, and in an implicit way, collaborating on roads shared by autonomous and human-driven cars.

In these collaborative teamwork scenarios, an important challenge for robots is to understand and interact with human agents seamlessly and even further influence a human team to achieve a desired goal. For instance, imagine a mixed human-robot search and rescue mission with no direct communication capabilities similar to Fig. 4.1. When a quadcopter senses valuable information from the environment how should the quadcopter direct the rest of its human teammates toward the desired goal?

One common solution is to assign leading and following roles to the team a priori before starting the search and rescue mission. Many current human-robot interactions determine leader-follower roles beforehand [59, 98, 114, 175, 189, 63, 161]. This include tasks that require learning from demonstrations or preferences, where the human is considered as the leader and the robot is the follower [36, 4, 1, 200, 47, 140, 24, 146], or assistive tasks where the robot teaches or assists human users [157, 95, 120, 89, 108]. However, assigning leadership roles a priori is not always feasible in dynamically changing environments or long-term interactions.

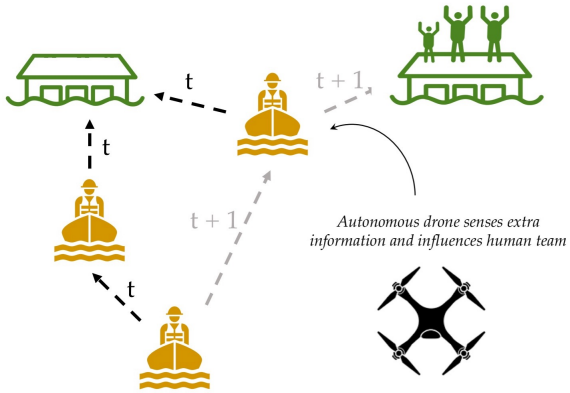


Figure 4.1: A search and rescue example, where a team of humans intend to rescue people from two islands shown in green. The quadcopter collects more information and determines that the team should head towards the island on the right. It guides the human team toward the island on the right using a graph representation that models the human team. We estimate leading and following relationships in human teams (denoted by the arrows), and use this to create influential robot policies. The black arrows represent intended human leading and following behaviors whereas the grey arrows represent updated leading and following behaviors after the influencing robot action.

There has also been significant prior work on how we can construct intelligent robot policies that induce desired behaviors from people [163, 73, 138, 139, 23, 193, 116]. However, all of these works optimize for robot policies that influence only a single human in one-on-one interactions. These works are able to successfully produce influencing behaviors by keeping an estimate of the human's state and optimize for actions based on the estimation, which is often computationally intractable with larger groups of humans.

Instead of keeping track of each individual's state in a team, we propose a more scalable method that estimates the collective *team's* state. Similar to individuals, teams exhibit behavioral patterns and structures that robots can use to create intelligent influencing policies. One particular feature of human teams we will focus on in this work is *leading and following relationships.*

> *Our key insight is that there exists an underlying graphical structure encoding the larger and more complex interactions between humans in team settings.*

In this chapter, we develop a scalable approach to extract meaningful latent structures in teams of humans that represent their leading and following behaviors. We extract an underlying graph, *leader-follower graph (LFG)*, to represent the *global* pattern of leader-follower dynamics using information from *local*, pairwise leader-follower interactions that we learn using supervised learning techniques. This structure provides a concise and informative representation of the current state of the team and can be used in planning. We then develop novel strategies for robots who join the human team to efficiently estimate the leader-follower graph and further influence this structure to more

efficiently achieve the team's goals. For instance, as shown in Fig. 4.1, there is an underlying team structure between the humans who are collaboratively navigating towards the left goal. However, a robot capable of estimating this underlying structure through the leader-follower graph can follow strategies that collectively influence the team to instead navigate the team towards the right goal, which could lead to a more desirable outcome.

We demonstrate the generalizability of our approach by applying our framework to a second type of group dynamics: predator-prey relationships. We show that we are able to successfully model predator-prey relationships using leader-follower graphs (LFGs). We also demonstrate that a robot using this LFG model is able to influence predator-prey dynamics.

Our contributions in this chapter are as follows:

- Formalizing and learning a graphical structure that captures complex relationships between members in human teams.
- Developing optimization-based robot strategies that leverage the graph representation to influence the team towards a more efficient objective.
- Providing simulation experiments in a pursuit-evasion game demonstrating the robot's influencing strategies to reverse a leader-follower relationship, distract a team, and lead a team towards an optimal goal based on its learned leader-follower graph.
- Generalizing our framework to a predator-prey domain and showing that our framework can still successfully model group dynamics, scalably deal with different group sizes, and can be used to design influencing policies.

In the rest of this chapter, we first discuss relevant work on modeling teams, influencing teams, and ad hoc teamwork in Section 4.2. We then describe our formalism and algorithm for learning graphical representations of human teams in the leader-follower domain (Sections 4.3-4.5) followed by the predator-prey domain(Sections 4.6-4.8). Finally, we describe our experiments in the leader-follower domain (Section 4.9) and the predator-prey domain (Section 4.10) followed by a discussion of limitations and future works.

4.2 Related Work

4.2.1 Modeling Teams

Finding computationally efficient ways to model human teams is an important part of this work. These models can be used to design intelligent policies that allow an agent to influence or coordinate with the team. We review ways in which prior works have modeled groups of agents.

Flocks and Swarms. Many works model flocks and swarms inspired by animal flocking behavior [61, 40, 160, 178]. These models describe how groups reach consensus in orientation when

navigating a space. They generally assume that all agents are homogenous and that they follow the same, relatively simple, update rule. Important components of this update rule include aligning orientation with their neighbors, positional attraction and repulsion towards neighbors, and some noise [61]. For example, Cristiani and Piccoli are able to replicate many self-organized patterns found in nature by modeling long-range cohesion, short-range repulsion, and the agents' visual fields [40]. Rosenthal et al. show that all agents are not equally susceptible to being influenced. They show that individuals with relatively few strongly connected neighbors are both more socially influential and susceptible to being influenced [160]. While these models are computationally efficient, they are too simplistic to be able to capture social dynamics that occur in human teams.

Attention and Graph Neural Networks. Recently, graph neural networks that use attention have become popular for modeling agent interactions [80, 109, 84, 91]. Attention is generally used to learn edge weights between agents. Vertex Attention Interaction Network (VAIN) uses attention to capture local structure by allowing the network to determine which agents will share information [80]. Li et al. uses self-attention to find structure in a coordination graph and then uses graph neural networks to integrate information among all agents [109]. Jiang et al. uses multi-head dot product attention to extract relations among neighboring agents in order to increase agents' receptive fields. Latent features are then extracted from these enlarged receptive fields to learn cooperative policies [91]. Compared to our approach, attention-based methods generally have more parameters and thus require more data to train. However, using attention-based methods to model human teams could be promising future work.

Modeling Humans. While there are many works that model multiagent systems, the extent to which these models can generalize to groups of humans remains underexplored. Many works in cognitive science, psychology, and behavioral economics have created predictive models of humans by modeling their biases and suboptimalities. For instance, Ordonez and Benson III investigated how humans make decisions under time constraints [143]. Simon developed the concept of bounded rationality to reflect limited humans' limited cognitive resources [172]. Tversky and Kahneman developed Cumulative Prospect Theory to capture human-decision making under risk and uncertainty [187]. In robotics, being able to successfully predict human behavior has shown to improve performance on tasks such as assistive robotics [108, 120, 89, 50], autonomous driving [165, 164, 14], collaborative games [136], and motion planning [201, 126]. The noisy rational choice model has been an extremely popular choice due to its simplicity [25, 24, 21, 54, 17]. Other models include the adoption of Cumulative Prospect Theory for human-robot interaction [101], models of human driving [66, 112], as well as learning-based models [131, 145].

In addition to explicitly modeling human behavior, robots have also been able to infer human preferences through interactions using partially observable Markov decision processes (POMDPs) which allow reasoning over uncertainty on the humans' internal state or intent [31, 48, 103, 124, 88, 164]. Human's intent inference has also been achieved through human-robot cross-training [140] as

well as various other approximations to POMDP solutions such as augmented MDPs, belief space planning, approximating reachable belief space, and decentralization [3, 99, 100, 142, 149, 162]. However, these methods usually focus on modeling a single human agent and do not capture social dynamics that occur among humans.

4.2.2 Influencing Teams

Given a model of a team, an important next question is how a more informed agent can use this model to coordinate with or influence the team.

Flocks and Swarms. Literature on influencing flocks and swarms looks at how informed agents can guide the group towards a preferred direction. This is similar to some of our evaluation tasks where the robot agent attempts to guide the human team towards a particular goal. The homogeneity and simple nature of agents in flocks and swarms allow for leader agents to implicitly influence the group. More specifically, implicit leadership algorithms allow a group of agents to reach consensus where each agent can observe their neighbors' states within a particular radius. As agents attempt to align their orientation with their neighbors', this empowers informed agents to lead [197, 58]. Prior work has also examined properties that make a swarm more susceptible to influence. Couzin et al. show that in groups of animals, only a small proportion of informed agents are required, and the larger the group, a smaller the proportion of informed individuals are needed [39]. Celikkanat et al. study the extent to which informed individuals can lead a flock by varying three factors: (1) the weight of the direction of preference (2) the ratio of informed individuals and (3) the size of the flock. They find that a flock is easier to control when moderate weight is put on the direction of preference (2) larger flock sizes and (3) more agents attempt to align their states with neighboring agents' states [34]. It is difficult to apply these findings to human teams due to the simplicity of flock and swarm models.

Human-Swarm Interaction. There has also been considerable work on how humans can influence flocks and swarms. Tiwari et al. consider the problem of leader placement when steering a large robot swarm [183]. Robots can either be controlled by a human or behave according to a swarm model. The authors consider which robots are positionally best equipped to influence the swarm (front, middle, or periphery). Kerman et al. and Brown et al. also consider how humans can influence swarms by controlling a subset of them [97, 30]. They show that humans are able to lead the swarm to switch from torus to flock formations and vice versa. Our work tackles the reverse problem where a robot agent must influence a team of humans.

4.2.3 Ad Hoc Teaming

An autonomous ad hoc agent must both model and influence a team that it has never seen before [176]. The ad hoc setting is similar to ours in that we expect our robot agent to influence a

human team that it has never worked with before. Ad hoc teaming has been studied in the multi-armed bandit setting where a teacher needs to trade off between teaching a new learning agent and exploitation [177, 20]. Role assignment in ad hoc teams have also been studied [28, 60]. Typically, an ad hoc agent needs to select a role such that it maximizes the team's utility. For instance, in Bowling and McCracken's work, teammates assign a role to the agent and the agent's job is to infer its role by simulating plays and selecting the one that is most similar to current teammate behavior [28]. Liemhetcharat models how well agents work together in ad hoc teams using a graph; nodes represent agents, their value represent the agent's capabilities, and agent synergy is determined by their capabilities and how far apart they are located from other agents in the graph [113]. Liemhetcharat describes how to learn this graph based on observations of team performance and then use this model to plan for creating effective ad hoc teams. Barrett et al. introduce model-based and model-free algorithms that allows ad hoc agents to collaborate with a variety of different teammates [19]. The algorithms either learn models about prior teammates or policies on how to collaborate with prior teammates, and uses this knowledge to interact with current teammates. Albrecht assumes that agents can be characterized into a set of policies drawn from some unknown distribution [5]. The author uses a Bayesian approach where agents update their posterior beliefs about types of other agents which can then be used for planning. While many of these ad hoc teaming works focus on modeling different types of potential teammates, in this work, we focus on modeling a specific type of *latent group dynamics* — leading and following graphs — in order to enable a robot to interact with an unknown team.

4.3 Formalism for Modeling Leading and Following in Human Teams

Running Example: Pursuit-Evasion Game. We define a multi-player pursuit-evasion game on a 2D plane as our main running example. In this game, each pursuer is an agent in the set of agents I that can take actions in the 2D space to navigate. There are a number of stationary evaders, which we refer to as *goals*. The objective of the pursuers is to collaboratively capture the evaders (goals). Fig. 4.2 shows an example of a game with three pursuers, shown in orange, and three goals, shown in green. The action space of each agent is identical, $A_i = \{$move up, move down, move left, move right, stay still$\}$; the action spaces of all agents collectively define the joint action space A. All pursuers must jointly and implicitly agree on a goal to target, and a goal will be captured when all pursuers collide with it as shown in Fig. 4.2 (b).

Leaders and Followers. We define a set of goals $g \in G$, which abstracts the idea of the agents reaching a set of states in order to fully optimize the joint reward function. For instance, in a pursuit-evasion game, the goals informally correspond to the evaders that need to be captured by all the pursuers, i.e., all the agents (pursuers) need to reach a state corresponding to the goals (evaders)

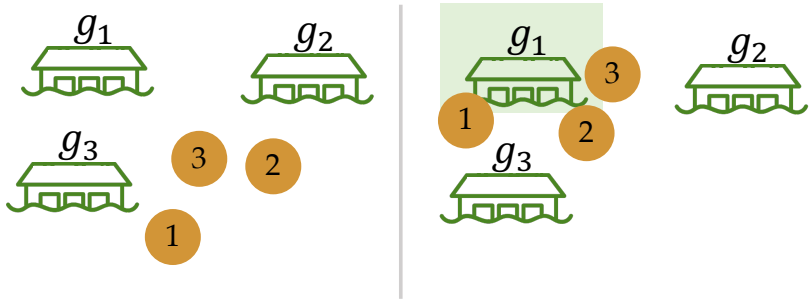


Figure 4.2: Pursuit-evasion game. (Left) we demonstrate a pursuit-evasion game with three goals (green circles), and three pursuers (orange circles). The pursuers must jointly agree on moving toward a target. (Right) The three pursuers move to g_1 to capture it.

being captured. A goal in G intuitively signifies a way for the agents to coordinate strategies with each other. For instance, in a pursuit-evasion game, the agents should collaboratively plan on actions that capture the goals. To put this in the context of leading and following, when agents capture a goal, *the goal can be thought of as being followed.*

Each agent $i \in I$ *follows* a goal or another agent, which we refer to as a *leader*. Formally we let $l_i \in G \cup I$, where l_i is either an agent or a fixed goal g who is the leader of agent i (agent i follows l_i). This is shown in Fig. 4.3 (a), where agent 2 follows goal g_1 ($l_2 = g_1$) and agent 3 follows agent 2 ($l_3 = 2$).

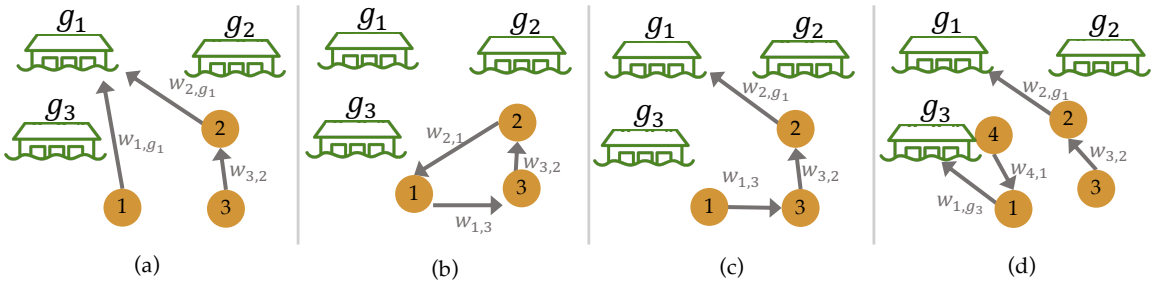


Figure 4.3: (a) Leader-follower graph. Green islands are the goals that need to be captured. Orange circles are the pursuers. (b) Cyclic leader-follower graph. We design policies that avoid such cyclic behaviors. (c) Chain behavior in the leader-follower graph. (d) Multiple teams.

Leader-Follower Graph. The set of leaders and followers form a directed *leader-follower graph* as shown in Fig. 4.3 (a). Each node represents an agent $i \in I$ or goal $g \in G$. The directed edges represent leading-following relationships, where there is an outgoing edge from a follower to its leader. The weights on the edges represent a *leadership score*, which is the probability that the tail node is the head node's leader. For instance, in Fig. 4.3 (a), $w_{3,2}$ represents the probability that 2 is 3's leader. The leader-follower graph is dynamic in that agents can decide to change their leaders

at any time. We assume that there could be an implicit transitivity in a leader-follower graph, i.e., if an agent i follows an agent j, implicitly it could be following the agent j's believed ultimate goal.

Some patterns are not desirable in a leader-follower graph. For instance, an agent would never follow itself, and we do not expect to observe cycling leading-following behaviors (Fig. 4.3 b). Other patterns that are likely include: chain patterns (Fig. 4.3 c) or patterns with multiple teams where multiple agents directly follow goals (Fig. 4.3 d). We describe how to construct a leader-follower graph that is scalable with the number of agents and avoids the undesirable patterns in Sec. 4.4.

Partial Observability. The leader of each agent, l_i, is a latent variable. We assume that agents cannot directly observe the leading and following dynamics of other agents. Thus, constructing leader-follower graphs can help robot teammates predict who will follow whom, allowing them to strategically influence teammates to adapt roles. We assume agents have full information on the observations of themselves and all other agents. (e.g. positions and velocities of agents).

4.4 Construction of a Leader-Follower Graph

In this section, we focus on constructing the leader-follower graph that emerges in collaborative teams. We will first focus on learning pairwise relationships between agents using a supervised learning approach. We then generalize our dyadic scoring to multi-player settings using graph theoretic algorithms. This combination of data-driven and graph-theoretic approaches allows the leader-follower graph to efficiently scale up with the number of agents. Our aim is to leverage this leader-follower graph to enable robot teammates to produce helpful leading behaviors.

4.4.1 Pairwise Leadership Scores

We first focus on learning the probability of any agent i following any goal or agent $j \in G \cup I$. The pairwise probabilities help us estimate the leadership score $w_{i,j}$, i.e., the weight of the edge (i, j) in the leader-follower graph.

We develop a general framework of estimating the leadership scores using a supervised learning approach. Consider a two-player setting where $I = \{i, j\}$, we collect labeled data where agent i is asked to follow j, and agent j is asked to optimize for the joint reward function assuming it is leading i, i.e., following a fixed goal g in the pursuit-evasion game ($l_i = j$ and $l_j = g$). We then train a LSTM network with a softmax layer to predict each agent's most likely leader.

Data Collection. We collect labeled human data by asking participants to play a pursuit evasion game. We recruited pairs of humans and randomly assigned leaders l_i to them (i.e., another agent or a goal). Participants played the game in a web browser using their arrow keys and were asked to move toward their assigned leader, l_i. In order to create a balanced dataset, we collected data from all possible configurations of leaders and followers in a two-player setting (the configurations are shown in Fig. 4.7). We collected a total of 186 games.

Since human data is often noisy and difficult to collect in large amounts; we further augmented our dataset with synthetic data, where we had simulated humans play the game. We simulated humans based on a potential field path planner [18]. Agents at location q plan their path under the influence of an artificial potential field $U(q)$, which is constructed to reflect the environment. Agents moved toward their leaders by following an attractive potential field. Other agents and goals that are not their leaders are treated as obstacles that emit a repulsive potential field. In our game setting, the position of agent's assigned leader l_i is given an attractive potential field. The rest of the goals and agents are expressed as repulsive potentials.

Potential Field for Simulated Human Planning. We denote the set of attractions as $\mathcal{A}$, and the set of repulsive obstacles as $\mathcal{R}$. The overall potential field is a weighted sum of potential fields from all attractive and repulsive obstacles. θ_i is the weight for attractive potential field from $i \in \mathcal{A}$, and θ_j is the weight for repulsive potential field from $j \in \mathcal{R}$.

$$U(q) = \sum_{i \in \mathcal{A}} \theta_i U^i_{\text{att}}(q) + \sum_{j \in \mathcal{R}} \theta_j U^j_{\text{rep}}(q) \tag{4.1}$$

The optimal action a that an agent would take lies in the direction of the potential field gradient.

$$a = -\nabla U(q) = -\sum_{i \in A} \theta_i \nabla U^i_{\text{att}}(q) - \sum_{j \in R} \theta_j \nabla U^j_{\text{rep}}(q)$$

In our implementation, the attractive potential field increases as the distance to goal becomes larger to help the agent reach the goal. On the other hand, the repulsive potential field has a fixed effective range, within which the potential field increases as the distance to the obstacle decreases. The attractive and repulsive potential fields are constructed in the same way for all attractive and repulsive obstacles. Specifically, the attractive potential field of attraction i, denoted as $U^i_{\text{att}}(q)$, is constructed as the square of the Euclidean distance $\rho_i(q)$ between agent at location q and attraction i at location q_i. In this way, the attraction increases as the distance to goal becomes larger. ϵ is the hyper-parameter for controlling how strong the attraction is and has consistent value for all attractions.

$$\begin{aligned} \rho_i(q) &= \|q - q_i\| \\ U^i_{\text{att}}(q) &= \tfrac{1}{2}\epsilon \rho_i(q)^2 \\ -\nabla U^i_{\text{att}}(q) &= -\epsilon \rho_i(q)(\nabla \rho_i(q)) \end{aligned}$$

The repulsive potential field $U^j_{\text{rep}}(q)$ is used for obstacle avoidance. It usually has a limited effective radius since we do not want the obstacle to affect agents' planning if they are far way from each other. Our choice for $U^j_{\text{rep}}(q)$ has a limited range γ_0, where the value is zero outside the range. Within distance γ_0, the repulsive potential field increases as the agent approaches the obstacle.

Thus, to compute the repulsive potential field to obstacle j at location q, we first identify the minimum distance $\gamma_j(q)$ between q and the obstacle j as in Eq.(4.2). Coefficient η and range γ_0 are the hyper-parameters for controlling how conservative we want our collision avoidance to be and is consistent for all obstacles. Larger values of η and γ_0 mean that we are more conservative with collision avoidance and want the agent to keep a larger distance to obstacles.

$$\begin{aligned}
\gamma_j(q) &= \min_{q' \in \text{obs}_j} \|q - q'\| \\
U^j_{\text{rep}}(q) &= \begin{cases} \frac{1}{2}\eta(\frac{1}{\gamma_j(q)} - \frac{1}{\gamma_0})^2 & \gamma_j(q) < \gamma_0 \\ 0 & \gamma_j(q) > \gamma_0 \end{cases} \\
\nabla U^j_{\text{rep}}(q) &= \begin{cases} \eta(\frac{1}{\gamma_j(q)} - \frac{1}{\gamma_0})(\frac{1}{\gamma_j(q)^2})\nabla\gamma(q) & \gamma(q)_j < \gamma_0 \\ 0 & \gamma_j(q) > \gamma_0 \end{cases}
\end{aligned} \tag{4.2}$$

In our experiments, we find that our simulations are good approximations of human behavior. The simple nature of the task given to humans (i.e., move directly toward your assigned leader l_i) is easily replicated in simulation.

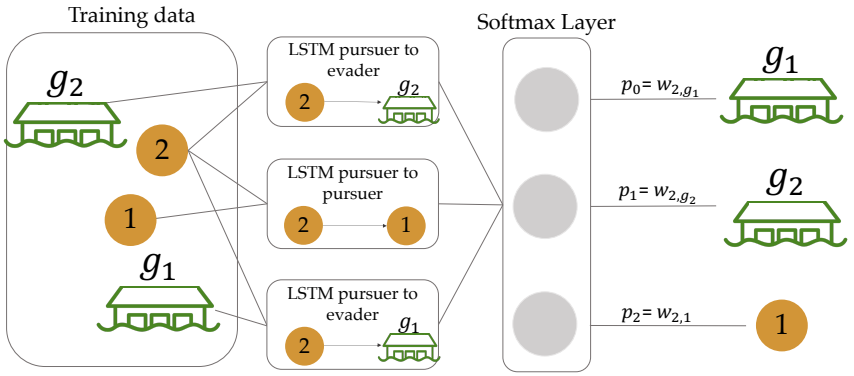


Figure 4.4: Scalable neural network architecture. This example predicts the probability of another agent j being agent 2's leader, $w_{2,j}$. There are three LSTM submodules used because there are two possible evaders and one possible agent that could be agent 2's leader. This architecture demonstrates how one can select P-P and P-E modules and discover the leader-follower relationships in a more scalable and compositional manner.

Training with a Scalable Network Architecture. Our network architecture consists of two LSTM submodules, one to predict player-player leader-follower relationships (P-P LSTM) and one to predict player-evader relationships (P-E LSTM). We use a softmax output layer with a cross-entropy loss function to get a probability distribution over j and all goals $g \in G$ of being i's leader. We take the leader (an agent or a goal) with the highest probability and assign this as the leadership

score. The P-P and P-E submodules allow us to scale training to a game of any number of players and evaders as we can add or remove P-P and P-E submodules depending on the number of players and evaders in a game. An example of our scalable network architecture is illustrated in Fig. 4.4.

Evaluating Pairwise Scores. Our network trained on two-player simulated data successfully captured the pairwise leading-following relationship (training accuracy: 80%, validation accuracy: 83%). We also experimented with training with three-player simulated data as well as a combination of two-player simulated and human data (two-player mixed data) resulting in (training accuracy: 97%, validation accuracy: 75%).

Validation results are shown in Fig. 4.5. Our model trained with mixed two-player data was first trained on simulated data and then trained on human data. For this reason, we have represented the mixed-data model as a horizontal line in Fig. 4.5 demonstrating the final validation accuracy.

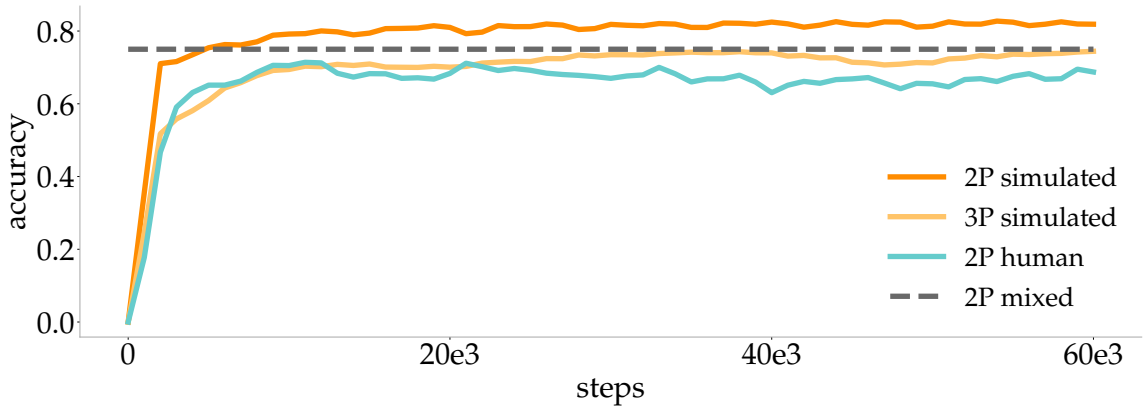


Figure 4.5: Validation accuracy when calculating pairwise leadership scores trained on simulated, human, and mixed data (simulated & human), described in Sec. 4.4.1

4.4.2 Maximum Likelihood Leader-Follower Graph

To build a leader-follower graph in settings with more than two players, we compute pairwise weights $w_{i,j}$ of leader-follower relationships between all possible pairs of leaders i and followers j. The pairwise weights (leadership scores) can be computed based on the supervised learning approach described above, indicating the probability of one agent or goal being another agents' leader. After computing $w_{i,j}$ for all combinations of leaders and followers, we can create a directed graph $\mathcal{G} = (V, E)$ where $V = I \cup G$ and $E = \{(i,j) | i \in I, j \in I \cup G, i \neq j\}$, and the weights on each edge (i,j) correspond to $w_{i,j}$. In addition, we add a special root node, where all the goals $g \in G$ have an outgoing edge to the root node. This produces a fully connected graph with each edge corresponding to the probability of one agent leading another, as shown in Fig. 4.6 (a).

Our model builds the graph based on the the pairwise scores, and thus can generalize to groups

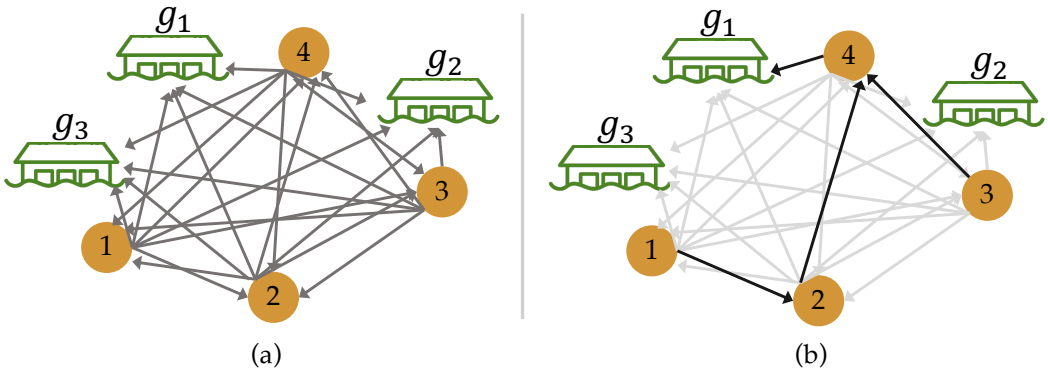
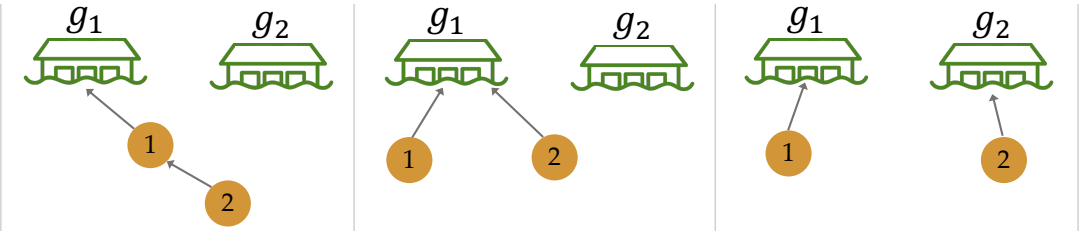


Figure 4.6: (a) Graph $\mathcal{G}$. The directed edges represent pairwise likelihoods that the tail node is the head node's leader. (b) Maximum-likelihood leader-follower graph, $\mathcal{G}^*$. For each node, we select the outgoing edge that has the highest weight as shown by the bold edges.

with different sizes. The computation increases quadratically with the size of the graph along with the number of pairs.

To create a more useful graph, we extract the maximum likelihood leader-follower graph $\mathcal{G}^*$ by pruning the edges of our constructed graph $\mathcal{G}$. We prune the graph by greedily selecting the outgoing edge with highest weight for each agent node. In other words, we select the edge associated with the agent or goal that has the highest probability of being agent i's leader, where the probabilities correspond to edge weights as in Fig. 4.6 (b). When pruning, we make sure that no cycles are formed. If we find a cycle, we will choose the next probable edge. Our pruning approach is inspired by Edmonds' algorithm [53, 37], which finds a maximum weight arborescence [96] in a graph. An arborescence is an acyclic directed tree structure, where there is exactly one outgoing edge from a node to another. We use a modified version of Edmonds' algorithm since, compared to our approach, a maximum weight arborescence is more restrictive; it requires the resulting graph to be a tree.

Evaluating the Leader-Follower Graph. We evaluate how accurate our leader-follower graph with three or more agents is when trained on simulated two-player and three-player data, as well as a combination of simulated and human two-player data (shown in Table 4.1). We evaluated our leader-follower graph on simulated three, four, and five-player games, as well as two and three-player human games. In each of these multi-player games, we extracted a leader-follower graph at each timestep and compared our leader-follower graph's predictions against the ground-truth labels. Our leader-follower graph performs better than random guessing by a large margin. The random policy selects a leader $l_i \in I \cup G$ for agent i at random, where $l_i \neq i$. The chance of being right is thus $\frac{1}{|G|+|I|-1}$. We then take the average of all success probabilities for all leader-follower graph configurations to compute the overall accuracy. As an example, for two-player games, there are in total three possible configurations as shown in Fig. 4.7. We compute the overall accuracy of the game by averaging $\frac{1}{2}$, $\frac{1}{2}$ and $\frac{1}{3}$, giving 0.44 (line 4, Table 4.1).



(a) chain structure (b) two agents go to the same goal (c) two agents go to separate goals

Figure 4.7: All possible leader-follower graph configurations for two-player settings.

Table 4.1: Generalization accuracy (Acc) of leader-follower graph (LFG) trained and tested with various data sources.

Training Data	**Testing Data**	**LFG Acc**	**Random Acc**
2 players, simulated	3 players, simulated	0.67	0.29
2 players, simulated	4 players, simulated	0.45	0.23
2 players, simulated	5 players, simulated	0.41	0.19
2 players, simulated	2 players, human	0.68	0.44
2 players, simulated	3 players, human	0.47	0.29
3 players, simulated	4 players, simulated	0.53	0.23
3 players, simulated	5 players, simulated	0.50	0.19
3 players, simulated	3 players, human	0.63	0.29
2 players, mixed	3 players, simulated	0.44	0.29
2 players, mixed	4 players, simulated	0.38	0.23
2 players, mixed	5 players, simulated	0.28	0.19
2 players, mixed	2 players, human	0.69	0.44
2 players, mixed	3 players, human	0.44	0.29

In all experiments shown in Table 4.1, our trained model clearly outperforms the random policy. Most notably, the models trained on simulated data scale naturally to settings with large numbers of players as well as human data. We use the model trained on three-player simulated data for our experiments in Section 4.5.

4.5 Planning based on Inference over Leader-Follower Graphs

With a representation for latent leadership structures in human teams, we use a leader-follower graph $\mathcal{G}^*$ to positively influence human teams, i.e., move the team towards a more desirable outcome. We describe how a robot can use the leader-follower graph to infer useful team structures. We then describe how a robot can leverage these inferences to plan for a desired outcome.

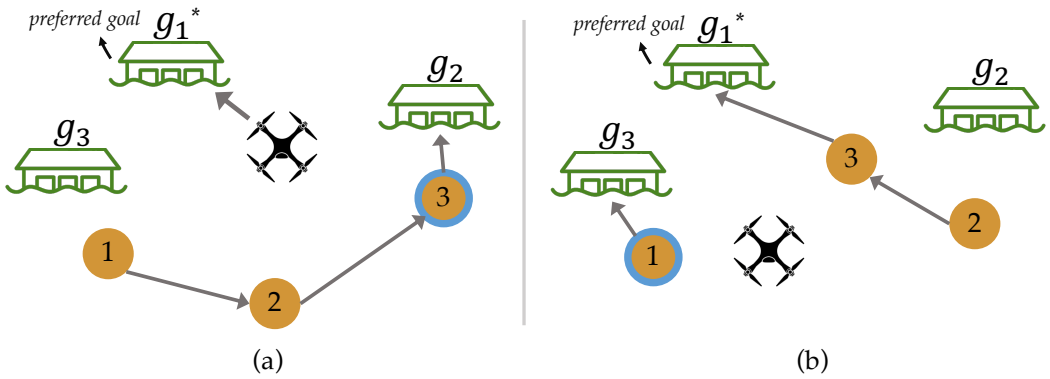


Figure 4.8: (a) In this graph, the most influential leader is agent 2. (b) The most influential leader trivially becomes agent 1 since agents 2 and 3 are already targeting the optimal goal g_1^*.

4.5.1 Inference Based on Leader-Follower Graph

Leader-follower graphs enable a robot to infer useful information about a team such as agents' goals or who the most influential leader is. These pieces of information allow the robot to identify key goals or agents that are useful in achieving a desired outcome (e.g., identifying shared goals in a collaborative task). A robot can then plan for a desired outcome by influencing or following these key goals and agents. We begin by describing different inferences a robot can perform on the leader-follower graph.

Goal Inference in Multiagent Settings. One way a robot can use structure in the leader-follower graph is to perform goal inference. An agent's goal can be inferred by the outgoing edges from agents to goals. In the case where there is an outgoing edge from an agent to another agent (i.e., agent i follows agent j), we assume transitivity, where agent i can be implicitly following agent j's believed ultimate goal. Being able to quickly infer the goal of multiple agents enables the robot to plan efficiently.

Influencing the Most Influential Leader. In order to lead a team toward a desired goal, the robot can also leverage the leader-follower graph to predict who the *most influential leader* is. We define the most influential leader to be the agent $i \in I$ with the most number of followers. Identifying the most influential leader allows the robot to strategically influence a single teammate that also indirectly influences the other teammates that are following the most influential leader. For example, in Fig. 4.8 (a) and (b), we show two examples of identifying the most influential leader from $\mathcal{G}^*$. In the case where some of agents are already going for the preferred goal, the one that has the most followers among the remaining players becomes the most influential leader, as shown in Fig. 4.8 (b).

4.5.2 Optimization Based on Leader-Follower Graph

The leader-follower graph allows the robot to single out key players and goals to follow or influence. A robot can then use this information to directly optimize for actions that help it achieve a desired outcome: Outcomes such as following the crowd or influencing the crowd's decision through utilizing the leader-follower graph. For instance, the probability of the robot becoming an agent i's leader can be expressed as $w_{i,r}$. The probability of the robot following a goal g is $w_{r,g}$.

To select actions $a \in A$ that maximize an objective involving weights $w_{i,j}$ in the leader-follower graph, we generate graphs $\mathcal{G}_{t+k}^{a_t}$ that simulate what the leader-follower graph would look like at timestep $t+k$ if the robot takes an action a_t at current timestep t. Over the next k steps, we assume human agents will continue along the current trajectory with constant velocity.

From each graph $\mathcal{G}_{t+k}^{a_t}$, we can obtain the weights $w_{i,j}^{t+k}$ corresponding to an objective that the robot is optimizing for (e.g., the robot becoming agent i's leader). We then optimize over the robot's actions to find the action a_t^* that maximizes a reward/outcome r that can be expressed in terms of $w_{i,j}^{t+k}$'s and $w_{i,g}^{t+k}$'s.

$$a_t^* = \underset{a_t \in A}{\operatorname{argmax}}\ r\left(\{w_{i,j}^{t+k}(a_t)\}_{i,j \in I}, \{w_{i,g}^{t+k}(a_t)\}_{i \in I, g \in G}\right) \tag{4.3}$$

We describe three specific tasks that we will plan for using the optimization described in Eqn. (4.3).

Reversing a Leader-Follower Relationship. A robot can directly influence team dynamics by changing leader-follower relationships. Given a directed edge between agents i and j, the robot can use the optimization outlined in Eqn. (4.3) for actions that reverse an edge or direct the edge to a different agent. For instance, to reverse the direction of the edge from agent i to agent j, the robot will select actions that maximize the probability of agent j following agent i:

$$a_t^* = \underset{a_t \in A}{\operatorname{argmax}} \quad w_{j,i}^{t+k}(a_t),\ i, j \in I$$

The robot can also take actions to eliminate an edge between agents i and j by *minimizing* $w_{i,j}$. One might want to modify edges in the leader-follower graph when trying to change the leadership structure in a team. For instance, in a setting where agents must collectively decide on a goal, a robot can help unify a team with sub-groups (an example is shown in Fig. 4.3 (d)) by re-directing the edges of one sub-group to follow another. On the other hand, the robot can also redirect edges such that the team is dispersed, or reverse edges such that the edges form a cycle as shown in Fig. 4.3 (b).

Distracting a Team. In adversarial settings, a robot might want to prevent a team of humans from reaching a collective goal g. In order to stall the team, a robot can use the leader-follower graph to identify who the current most influential leader i^* is. The robot can then select actions that maximize the probability of the robot becoming the most influential leader's leader and minimize

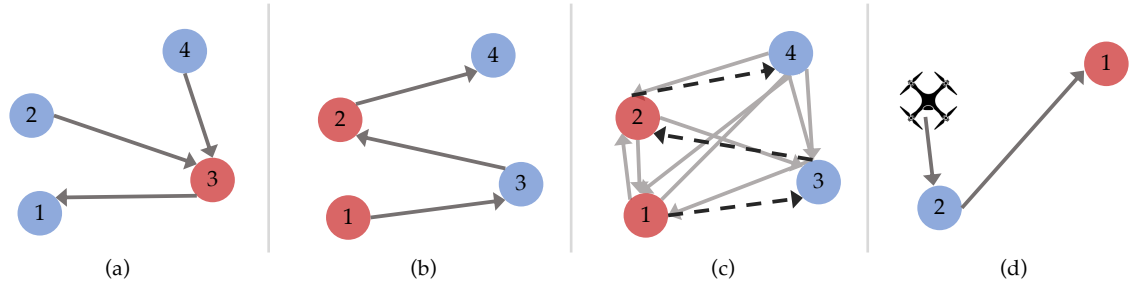


Figure 4.9: (a) Example of a predator-prey dynamic in capture the flag. Red and blue circles represent agents on different teams. (b) Another example of a predator-prey dynamic between members of red and blue teams. (c) Predator-prey graph where the most likely edges are bolded. (d) The robot joins the game as agent 2's predator.

the probability of the most influential leader following the collective goal g:

$$a_t^* = \underset{a_t \in A}{\text{argmax}} \quad w_{i^* r}^{t+k}(a_t) - w_{i^* g}^{t+k}(a_t), \; i^* \in \mathcal{I} \tag{4.4}$$

Distracting a team from reaching a collective goal can be useful in cases where the team is an adversary. For instance, a team of military drones masquerading as enemy drones may want to prevent the enemy team from reaching a joint goal.

Leading a Team Towards the Optimal Goal. In collaborative settings where the team needs to agree on a goal $g \in G$, a robot that knows where the optimal goal $g^* \in G$ is should maximize joint utility by leading all of its teammates to reach g^*. To influence the team, the robot can use the leader-follower graph to infer who the current most influential leader i^* is. The robot can then select actions that maximize the probability of the most influential leader following the optimal goal g^*:

$$a_t^* = \underset{a_t \in A}{\text{argmax}} \quad w_{i^* g^*}^{t+k}(a_t), \; i^* \in \mathcal{I}$$

Being able to lead a team of humans to a goal is useful in many real-life scenarios. For instance, in search-and-rescue missions, robots with more information about the location of survivors should be able to lead the team in the optimal direction.

4.6 Modeling Predator-Prey Relationships

We test the generalizability of our framework by modeling a different type of group dynamics: predator-prey relationships. Each agent has either a prey that they are trying to capture, predators they are eluding, or both. Predator-prey relationships are different from leader-follower relationships in that agents are tasked with eluding their predator. In some cases, agents must simultaneously capture their prey while eluding their predator, giving way to more complex dynamics than leading

and following. In human groups, predator-prey relationships can be found in games such as capture-the-flag. In capture-the flag, two teams guard regions that contain each team's flag. The goal of a team is to steal the other team's flag while protecting their own. Predator-prey relationships emerge when team members attempt to tag out members of the opposing team. An example is shown in Fig. 4.9 (a), where members of the blue team (agents 2 and 4) help their teammate (agent 1) escape from the opposing team.

Predator-Prey Game. We modify the pursuit-evasion game setup described in Sec. 4.3. In the modified version, there are no stationary evaders (goals). Instead, each agent in the set of agents I acts as either a predator, prey, or both. Predator agents are assigned a prey and are required to capture it by colliding with them. Likewise, prey agents have assigned predators and their goal is to avoid being captured. The action space of each agent $i \in I$ is identical as in Sec. 4.3, $A_i = \{$move up, move down, move left, move right, stay still$\}$.

Predator-Prey Graph. Using the set of predators and preys, we can form a directed predator-prey graph. Each node represents an agent $i \in I$. The directed edges represent predator-prey relationships where there is an outgoing edge from a predator to its prey. The weights on the edges represent the probability that the tail node is the head node's predator. Similar to leader-follower graphs, there can be many configurations of predator-prey graphs; two examples are shown in Figs. 4.9 (a) and (b). In this work, we experiment with various configurations of the predator-prey to validate that our framework can effectively capture these relationships between the agents in this predator-prey domain.

4.7 Construction of a Predator-Prey Graph

In this section, we describe how we learn these graphs. Similar to our leader-follower graphs, we use a supervised learning approach where we collect pairwise predator-prey data to train a predictive model. Like the leader-follower graph, our aim is to use this model to scalably construct a predator-prey graph for multi-agent settings. Ultimately, we hope to use this graph to build robot algorithms that can understand and influence predator-prey dynamics.

4.7.1 Pairwise Prey Scores

Data Collection. We recruited dyads to play the predator-prey game. We assumed a chain-structured predator-prey relationship. Predator and prey roles were randomly assigned to each partner. Participants played the game in the web browser where predators tried to collide with their prey as many times as possible within the time limit. We collected a total of 1.5 hours of data where we collected trajectories and scores of each participant.

We also generated synthetic human data using the same potential field simulator as described in Sec. 4.4.1). At the beginning of each game, each agent was randomly assigned one prey or no prey.

Only configurations that contain no loops are considered valid. By randomly assigning preys, we effectively covered all possible valid configurations. In our simulator, predators moved towards their prey by following an attractive potential field and prey moved away following a negative potential field. We simulated 1000 three-agent predator prey games. We chose three-agent games for data collection because to have agents that being both a predator and a prey, the minimal number of agents in the game is three.

Training the Model. To test the generalization of our framework in this new domain, we use the same LSTM submodules as in Sec. 4.4 to predict player-player predator-prey relationships. For each agent $i \in I$ in a game, we train our model to predict agent i's prey by feeding their and their partner's trajectory data into our submodules. We add an additional submodule where the agent i's trajectories are fed in twice to represent the event that the agent does not have prey. We use a softmax output layer with a cross-entropy loss function to compute a probability distribution over all agents of being agent i's prey. Before training, we pre-processed the data by normalizing it, shifting it to have a zero-centered mean, and down-sampled it. When training our network, we used the same hyperparameters as described in Sec. 4.4.

Evaluating Pairwise Scores. We evaluated the accuracy of our model on held out test sets of simulated and human data. Our network trained on three-player data performed with a validation accuracy of 95.51% in simulation with randomized predator-prey graph structure and 96.24% on human data with a chain structure. Both results indicate that our framework can accurately capture the predator-prey pairwise relationship.

4.7.2 Constructing and Evaluating the Predator-Prey Graph

In settings with more than three players, we construct a predator-prey graph based on the pairwise scores. For each agent i, we compute pairwise weights between agent i and all of its possible preys $j \in I, j \neq i$. In this Predator-Prey domain, we also compute an additional weight for agent i having no prey. With all of these scores, we then construct the graph as described in Sec. 4.4.

Evaluating the Predator-Prey Graph. We tested the generalization accuracy of the predator-prey graph by constructing the graph at each time step and comparing it against the ground truth labels. The results for testing on real human data and simulated data are demonstrated in Fig. 4.10 respectively. We found that the model trained with three agent data can successfully generalize to settings with more players with only a minor decrease on the accuracy. Similar patterns to Table 4.1 can also be observed here where the accuracy drops with larger numbers of agents. This is because as the number of agents increases, the task becomes more challenging and it becomes more difficult for the model to distinguish which agent is the prey.

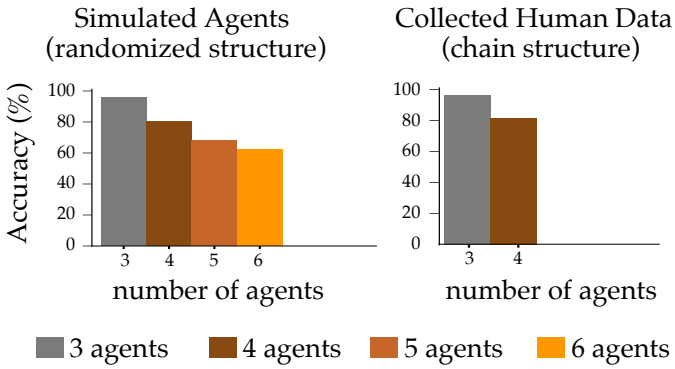


Figure 4.10: Generalization accuracy of the predator-prey graph tested with simulated agents and human data. Both models are trained with three-agent data and we tested models in games that contain more agents.

4.8 Planning with the Predator-Prey Graphs

We now leverage the information from the predator-prey graph to plan for robot behaviors that can influence group dynamics. In this work, we focus on the task of becoming the only dominant predator among the chain-structured predator-prey group. Accomplishing this task requires two steps. First we need to identify which agent is the top predator, and then we want the robot to hunt for that identified top predator.

Inference Based on Predator-Prey Graph. One direct way to identify which agent is the top predator is to identify the agent that has no predator based on the estimated predator-prey graph. For example, as in Fig. 4.9 (b), agent 1 is the top predator. In this work, we experiment with a predator-prey chain, and therefore, there is only one top predator among the group.

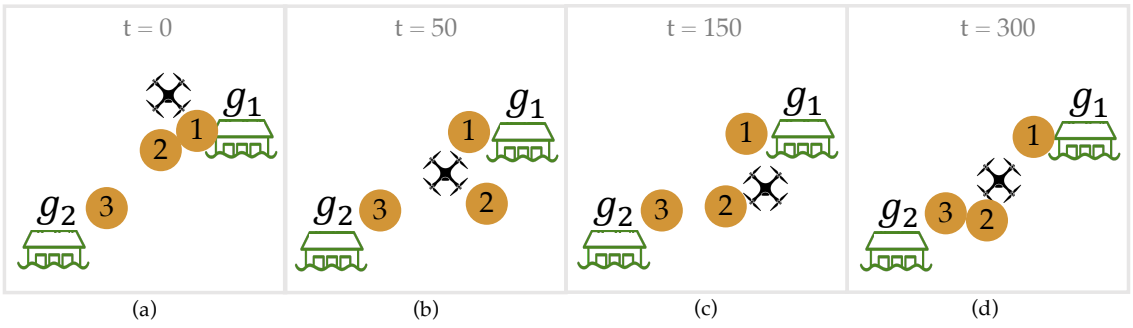


Figure 4.11: Adversarial game snapshots for a 300 second horizon. The orange circles are human agents. (a) Agents 1 and 2 start very close to g_1. (b-c) The robot prevents agent 2 from converging on g_1. (d) The robot leads agent 2 to another goal, successfully extending the game time.

Optimization Based on Predator-Prey Graph. After identifying the top predator, we can

now again use the predator-prey graph for optimization. At each time step t, we infer the top predator X_t based on the current predator-prey graph. Then, similar to Sec. 4.5, we generate the predator-prey graph k time steps ahead $\mathcal{G}_{t+k}^{A_t}$, assuming the robot takes actions $A_t = (a_t, \ldots, a_k)$ in the next k time steps. We then extract the weight w_{j,X_t}^{t+k} representing robot j being the predator of the identified top predator X_t from the graph $\mathcal{G}_{t+k}^{A}$. By maximizing this weight, we can compute the optimal robot actions:

$$A_t = (a_t, \ldots, a_k) = \underset{A_t=(a_t,\ldots,a_k)}{\operatorname{argmax}} \; w_{j,X_t}^{t+k} \tag{4.5}$$

We perform this optimization in a model predictive control fashion [57], where we find the optimal sequence of actions at time step t, and execute a_t. We then replan for a k time-step horizon at the next time step running the same optimization.

Other Tasks. In this work, we only demonstrate how to leverage the predator-prey graph for inference and optimization for the task of becoming the dominant predator in a chain-structured predator-prey group. However, like the leader-follower graph, we emphasize that the predator-prey graph is a general representation that can be combined with many other tasks as well, e.g., in more complex settings where the relationship is not limited to a chain structure. For example, the robot can help to protect another agent by identifying who its predators are and interfere with their actions.

4.9 Experiments: Leading and Following

We first evaluate our framework in the leader and follower domain. Through our experiments, we demonstrate the efficacy of the leader-follower graph in representing the agents and further in enabling better planning and optimization for the robot actions.

We evaluate our LFG on three different tasks that involve influencing multiagent human teams. For each task, we compare task performance of robot policies that use the LFG against robot policies that do not have access to the LFG. Across all tasks, we find that robot policies that use the leader-follower graph perform better, showing that our graph can easily be generalized to different settings.

Task Setup. Our tasks take place in the pursuit-evasion domain. Within each task, we conduct experiments with simulated human behavior. Humans move along a potential field as shown in Eqn. (4.1), where there are two sources of attraction: the agent's goal (a_g) and the crowd center (a_c). We also specify weights associated with these attractions to be $\theta_g = 0.6$ and $\theta_c = 0.4$. In this way, a simulated human would trade off between following the crowd and moving toward a target goal.

In each iteration of the task, the initial position of agents and goals are randomized. For all of

our experiments, game canvas is 500×500. At every time step, the human can move 1 unit in one of the four directions: up, down, left, right, or stay at its position. The robot's action space is the same but with larger move amount 5. We let the maximum game time limit be 1000.

Implementation Detail. We simulated 5000 games of each possible configuration, totaling 15000 games for the two-player setting (as shown in Fig. 4.7) and 35000 for the three-player setting. Each game stored the position of each agent and goal at all timesteps. Before training, we pre-processed the data by normalizing it, shifting it to have a zero-centered mean, and down-sampled it. Each game was then fed into our network as a sequence. Based on our experiments, hyperparameters that worked well for our training were a batch size of 250, learning rate of 0.0001 and hidden dimension size of 64. In addition, we used gradient clipping and layer normalization [13] to stabilize gradient updates.

4.9.1 Reversing a Leader Follower Relationship

We evaluate a robot's ability to change an edge of a leader-follower graph. In this task, the end goal of the robot is not to affect the environment as some of the other tasks we describe below (e.g., influence humans toward a particular goal). Instead, this experiment serves as a preliminary to others where we evaluate how well a robot is able to manipulate the leader-follower graph.

Methods. Given a human agent i who is predisposed to following a goal with weights $\theta_g = 0.6$, $\theta_c = 0.4$, we created a robot policy that encouraged the human agent to follow the robot r instead. The robot optimized for the probability $w_{i,r}$ that it would become agent i's leader.

Metrics. We evaluated the performance of the robot based on the leadership scores, i.e., probabilities $w_{i,r}$, computed by the leader-follower graph.

Results. We show that the robot can influence a human agent to follow it. Fig. 4.12 contains averaged probabilities over ten tasks. The probability of the robot being the human agent's leader $w_{i,r}$ increases over time, and averages to 73%, represented by the orange dashed line. Our approach performs well compared to the random method, which has an average performance of 26%, represented by the grey dashed line.

4.9.2 Adversarial Task: Distracting a Team

We now consider a task where the robot is an adversary that is trying to distract a team of humans from reaching a goal.

In this task, there are m goals and n players in the pursuit-evasion game. Among the n players, we have 1 robot agent and $n-1$ homogeneous human agents. $n-2$ human agents must collide with a goal at the same time to capture it, allowing 1 human to be absent. The game ends if all goals are captured or the game time exceeds the limit.

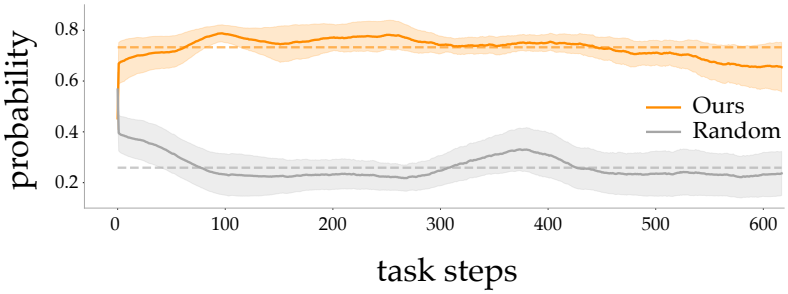


Figure 4.12: Probabilities of a human agent following the robot over 10 tasks. The robot is successfully able to become the human agent's leader as the task progresses.

Table 4.2: Average game time over 50 adversarial games with varying number of players

	number of goals (m=2)			
Model	n=3	n=4	n=5	n=6
LFG Closest Pursuer (**ours**)	**233.04±51.82**	**305.08±49.48**	**461.18±55.73**	**550.88±51.67**
LFG Influential Pursuer (**ours**)	201.94±45.15	286.44±48.54	414.78±50.98	515.92±48.80
Random	129.2±32.66	209.40±39.86	388.92±53.24	437.16±43.17
To One pursuer	215.04±50.00	231.42±44.69	455.16±58.35	472.36±49.75
To Farthest Goal	132.84±34.22	198.5±36.14	382.08±52.59	445.64±46.77

The adversarial robot's goal is to intentionally distract a team of human players so that they cannot converge to the same goal quickly and thus extending the game time. Note that simply blocking a single agent's way would not be a desirable solution, since we allow for an agent to be absent when capturing the goal.

Methods. We test our optimization methods based on the constructed leader-follower graph along with other baseline models.

We experimented with 3 baseline strategies without knowledge of LFG. In the *Random* strategy, the robot picks an action at each time step with uniform probability. *To One Pursuer* strategy is that the robot agent selects a random human agent and then goes towards it trying to block its way. The *To Farthest Goal* strategy selects the goal that the average distance to human players are largest and then goes to that goal in the hope that human agents would get influenced or may further change their goal by observing that some players are heading for another goal.

We also experimented with two optimization models based on the LFG. *LFG Closest Pursuer* involves the robot selecting the closest pursuer and choosing an action to maximize the probability of the pursuer following it (as predicted by the LFG). Similarly, *LFG Influential Pursuer* strategy involves the robot targeting the most influential human agent predicted by the LFG described in Sec. 4.5 and then conducting the same optimization of maximizing the following probability, as shown in Eqn. (4.4).

Metrics. We evaluated the performance of the robot with game time as metric. Longer game time

Table 4.3: Average game time over 50 adversarial games with varying number of goals

	number of players (n=4)			
Model	m=1	m=2	m=3	m=4
LFG Closest Pursuer (**ours**)	210.94±33.23	**305.08±49.48**	**289.22±52.99**	**343.00±55.90**
LFG Influential Pursuer (**ours**)	**239.04±39.73**	286.44±48.54	219.56±41.00	301.80±52.00
Random	155.94±21.42	209.40±39.86	205.74±43.05	294.62±54.01
To One Pursuer	123.58±9.56	231.42±44.69	225.52±41.47	317.92±54.75
to Farthest Goal	213.36±34.83	198.5±36.14	218.68±43.67	258.30±50.64

indicates that the robot does well in distracting human players.

Results. We conduct experiments with different game settings by varying n (number of players) and m (number of goals). For each specified game setting, we run the same 50 randomly initialized games for different robot strategy and compute the mean and standard deviation for game time over the 50 games. Across all the game settings we experimented with, our models based on LFG consistently outperforms methods without knowledge of LFG. The experimental results with varying number of players are summarized in Table 4.2. We also visualized the results of Table 4.2 in Fig. 4.13.

As shown in Fig. 4.13, average game time goes up as the number of players increases. This is because it is more challenging for more players to reach agreement on which goal to capture and thus takes longer time. The consistent advantageous performance suggests the effectiveness of LFG for inference and optimization in this scenario.

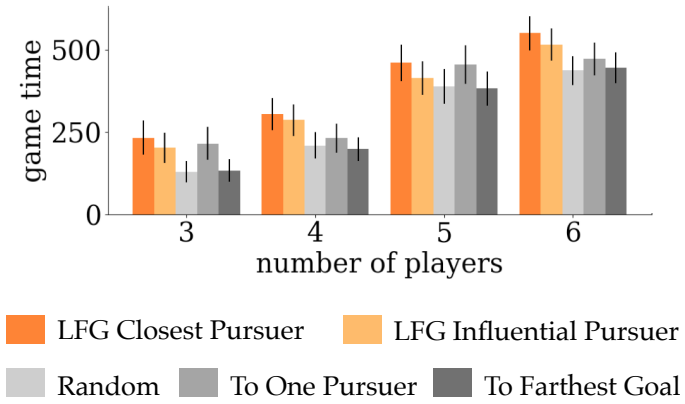


Figure 4.13: Visualization of results in Table 4.2 For adversarial task, average game time over 50 games with 2 goals (as in Fig. 4.11) with different number of players across all baseline methods and our model.

To demonstrate robot behavior in the adversarial game, we also took snapshots of one game as in Fig. 4.11. Player 1 and player 2 started very close to goal g_1 and thus it's very easy for them to

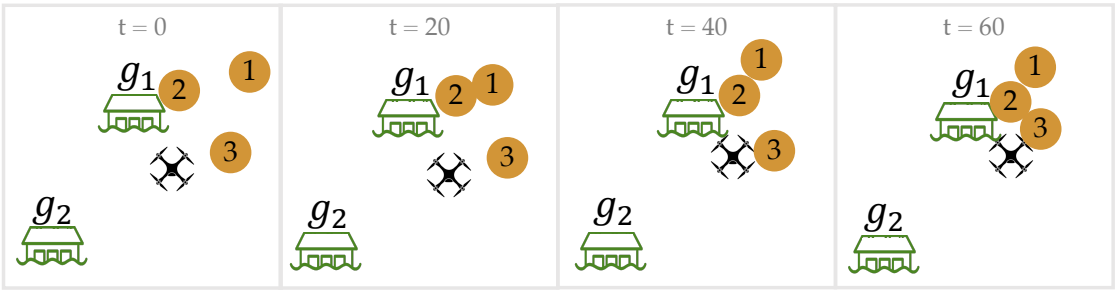


Figure 4.14: Collaborative game snapshots for a 60 second horizon. The orange circles are human agents. The robot moves towards agent 3 in order to help all the agents converge on g_1.

capture it. The robot approached agent 2 and tried to block its way, leading it to another goal g_2. In this way, the robot successfully extended the game time.

4.9.3 Cooperative Task: Leading a Team toward the Optimal Goal

Finally, we evaluate the robot in a cooperative setting where the robot tries to be helpful for human teams. The goal of the robot is to lead its teammates so that everyone can reach the target goal that gives the team the largest joint reward $g^* \in G$. g^* is not immediately observable to all teammates. We assume a setting where only the robot knows where g^* is (e.g. due to its better sensing capabilities as in Fig. 4.1).

The experiment setting is the same as the *Adversarial Task* where $n-2$ human agents need to collide with a goal to capture it. In this scenario, the task is considered successfully completed if the goal with the largest joint reward g^* is captured, and it is considered failed if any other suboptimal goal is captured or the game time exceeds the maximum limit.

Methods. Similar to the case in *Adversarial Task*, we explore two models where the robot chooses to influence its closest human agent or the most influential agent predicted by the LFG. Different from the *Adversarial Task*, here, the robot is optimizing the probability of the target agent following itself and the probability of them going to the desired goal.

We also experimented with three baseline methods. *Random* strategy is taking random actions. *To Target Goal* strategy is that the robot agent goes directly to the optimal goal g^* and then stays there trying to attract other human agents. *To Goal Farthest Player* strategy is that the robot goes to the player that is farthest away from g^* in the hope that it can influence the target back to g^*.

Metrics. We evaluated the performance of the robot strategy using the game success rate over 100 games.

Results. We experimented with varying number of goals and the results are summarized in Table 4.4. In this scenario, going directly to the desired goal is a very strong method since it already

conveys the message to other players that the robot is going for a specific goal. This method is especially effective when the game is not complex, i.e., the number of goals is small. However, our model based on the LFG still demonstrates competitive performance compared to it. Specially when the number of goal increases, the advantage of LFG gradually becomes dominant. This indicates that, in complex scenarios, brute force methods that do not have knowledge of human team hidden structure do not suffice. High-level understanding of human teams are necessary for better human-robot teaming in complex systems. Another thing to note is that the difference between all of the methods becomes smaller as the number of goals increases. This is because the game difficulty increases for all methods, and thus whether a game would succeed depends more on the game's initial conditions. We took snapshots of one game as in Fig. 4.14. In this game, the robot approaches other agents

Table 4.4: Success rate over 100 collaborative games with varying number of goals m.

number of players (n=4)					
Model	m=2	m=3	m=4	m=5	m=6
LFG Closest Pursuer	0.59	0.38	0.29	**0.27**	**0.22**
LFG Influential Pursuer	0.57	0.36	**0.32**	0.24	0.19
Random	0.55	0.35	0.24	0.21	0.20
To Target Goal	**0.60**	**0.42**	0.28	0.24	0.21
To Goal Farthest Player	0.47	0.29	0.17	0.19	0.21

and the desired goal in the collaborative pattern, trying to help catch the goal g_1.

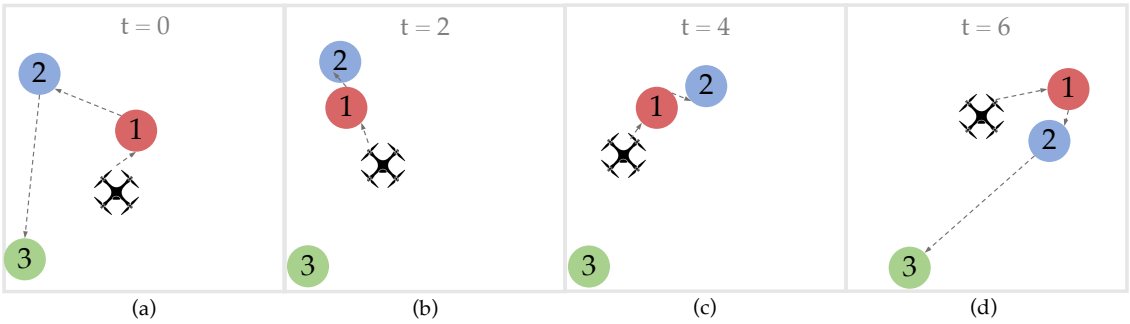


Figure 4.15: Predator-prey game snapshots. The different colored circles represent agents in different teams. (a) Agents follow a chain structure in the predator-prey game. (b) As agent 2 attempts to capture agent 3, agent 1 intervenes and attempts to capture agent 2. (c) Agent 2 flees. (d) Agent 2 attempts to capture agent 3 again.

4.10 Experiments: Predator-Prey

We next evaluate our framework in the predator-prey domain. We investigate whether our robot can utilize the predator-prey graph to insert itself into the game as the top predator.

Task Setup. We evaluate our approach with both simulated and real human agents in the modified pursuit evasion environment. In all of our experiments, agents follow a chain structure as shown in Fig. 4.15. We reference each (simulated and real) human agent by their ids $1 \dots n$, where n is the number of human agents. Each agent is instructed to capture the agent above it and run away from the agent below it. Thus agent 1 will always be the top predator and agent n will be the bottom-most prey. The robot's task is to join the game and become the top predator, i.e., capture agent 1.

An example of a 4 player game is shown in Fig. 4.15. Agent 1 is the top predator that tries to capture agent 2. Agent 2 aims to capture agent 3 but also tries to avoid being captured by agent 1. Agent 3 is at the bottom of the predator-prey chain and simply tries to avoid being captured. The robot joins and tries to become the top predator by capturing agent 1. Importantly, we do not inform the robot that its goal is to capture agent 1. The robot has to figure this out by relying on the learned graph structure. Similarly, we also do not explicitly inform the other agents about the robot's goal, i.e. the other agents will treat the robot neither as its predator nor prey.

The initial position of all agents are randomized. Our experiments are conducted on a canvas of size 500 x 500. At each time step, each agent can move 3 units in one of the cardinal directions or choose to stay in place.

Methods. In order to become the dominant predator, the robot first identifies the top predator. It then optimizes for the probability that it becomes that agent's predator, as described in Sec. 4.8. We compare against two methods with our predator-prey graph: a random agent (*Random*) and an agent that optimizes for moving towards the center of the other three agents (*Center*). *Center* encourages the robot agent to stay closer to the group without knowing which agent is the top predator. By including the *Center* method, we hope to verify that the robot is actually following agent 1 and not following other agents.

Metrics. We evaluated the performance of the robot based on the average number of time steps that the robot agent is in collision with agent 2. Longer collision time indicates that the robot performs well by capturing the top predator among the other agents.

Results with Simulated Humans. We conduct experiments with different settings by varying the number of agents. For each specified game setting, we run the same 100 randomly initialized games and compute the mean and standard error for the time the robot agent is capturing the top predator. The experimental results are summarized in Fig. 4.16. Across all the settings, the predator-prey graph that our method used was trained only with three-player data. We leverage this graph to optimize robot behavior in various multi-agent games. We can see that our method captures prey

for a longer amount of time compared to both *Random* and *Center* methods in the three-agent and four-agent settings. When the number of agents gets larger, e.g. when we have five agents, the performance degrades. This is because as the number of agents increases, the task becomes more challenging and correspondingly, the predator-prey generalization error also accumulates both in the inference and the optimization process.

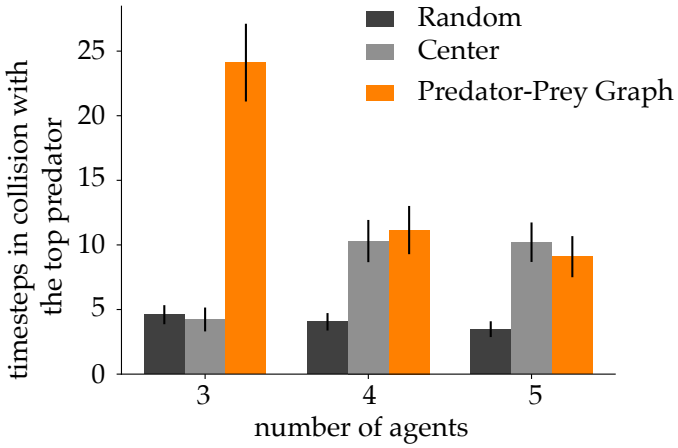


Figure 4.16: Average time the robot agent capturing (in collision with) the top predator over 100 games with varying number of agents.

Results with Human Participants. To evaluate the effectiveness of our framework against real human users, We recruited human participants to play 3 player and 4 player versions of the game. We conducted 6 games with different groups of participants. Each group played the game three times with a robot following our algorithm as well as the *Center* and *Random* methods in a randomized order. Results are shown in Table 4.5.

In the 3 player setting, we report the mean time the robot and agent 1 were in collision with their prey as well as the ratio of the two means. We do not report agent 2's score because they had no assigned prey. The ratio highlights the effectiveness of the robot over the other human predator, and thus acts as a good metric for assessing the robot policy. Looking at the results, our method achieves a higher ratio than *Random* demonstrating the effectiveness of the robot policy when interacting with real humans. In 3 player settings, *Center* places the robot in between the two human agents, making it impossible for agent 1 to capture agent 2 without being captured by the robot. This makes the robot's job as a predator trivial. *Center* is therefore a special case of the 3 player setting. In practice, this often leads to stalemates where all agents remained far apart from each other, as shown by agent 1's 0 mean in Table 4.5. However, in two out of the six games, agent 1 came close enough to the robot, which explains the robot's large average and standard deviation for *Center*.

In the 4 player setting, we report the mean time the robot were in collision with their prey. With

larger number of agents, our method demonstrates its advantage of capturing the group structure more and achieves highest performance. Compared to the special case in 3 player settings, the crowd's center was less correlated with its prey's position and thus *Center* demonstrates inferior performance compared to our method.

Table 4.5: Average number of time steps an agent is in collision with its prey over 6 games with 2 and 3 human participants.

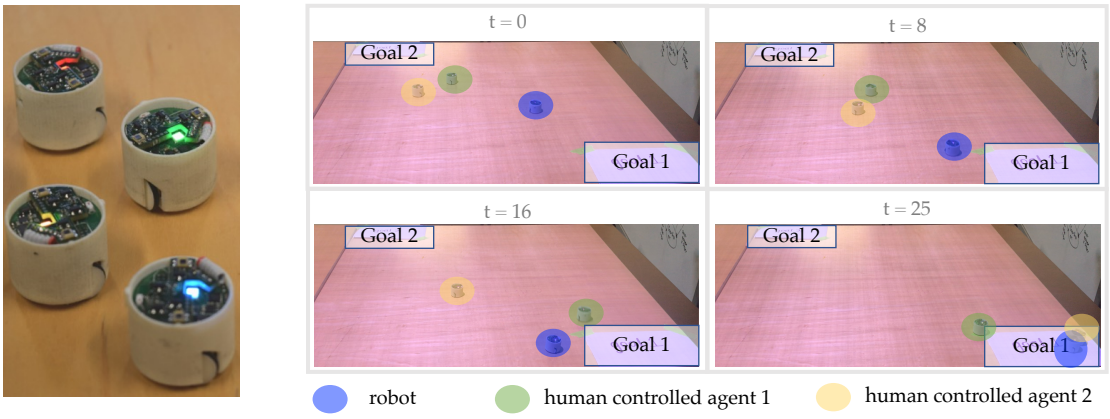
Game	Agent	Ours	Center	Random
3 player game	Robot	229.67 ± 164.4	**657.44 ± 1365.04**	103.78 ± 100.94
	Human Agent 1	132.17 ± 265.9	0	438.89.83 ± 858.63
	Robot/Human Agent 1	**1.74**	N/A	0.24
4 player game	Robot	**230.83 ± 93.32**	136.64± 102.26	7.13 ± 44.59

4.11 Conclusion and Discussion

Summary. We propose an approach for modeling group behavior in multi-agent human teams. We use a combination of data-driven and graph-theoretic techniques to learn a graph-based representation for leading-following and predator-prey dynamics. This graph representation encoding human team hidden structure is scalable with the team size (number of agents) since we base the model on *local*, pairwise relationship prediction and combine them to create a *global* model. We demonstrate the effectiveness of our graph structure by testing optimization-based robot policies that leverage the graph to influence human teams in different scenarios. Our policies are general and perform well across all tasks compared to other high-performing task-specific policies.

There are several ways in which our framework can be applied to more complex real-world settings. First, we can extend our approach to partially observable settings. When human agent positions are partially observable (i.e., the robot can only access the positions of its nearest neighbors), our framework can still be applied locally. For instance, the robot can determine local leader-follower structures that can be updated as the robot moves around and gathers more information.

We include preliminary results on what running our framework on real robots might look like in Fig. 4.17. We use Zooids robots for our experiment, which is a collection of custom-designed wheeled micro tabletop robots and can be used for various tasks including swarm drawing, interactive swarm visualization [102]. Users control the movement of Zooids through a GUI on computers by dragging the zooids icons to the intended moving directions from the interface. The video can be found here: https://youtu.be/6t_IfJ82EvE. One robot (highlighted in blue) was controlled by our framework and the rest were controlled by human users. In this cooperative task, the team would only receive reward if all agents go to the same goal within the maximum game time limit. There are two goals in the game, goal 1 in the bottom right and goal 2 in the upper left as shown in Fig. 4.17. The robot agent knows that goal 1 has largest reward and tries to lead the team towards the optimal goal.



(a) (b)

Figure 4.17: (a) The Zooids robots. (b) Cooperative Zooids robot game snapshots for a 25 second horizon. The highlighted blue robot is controlled by our framework, the rest were controlled by human users. There are two goals in the game, goal 1 in the bottom right and goal 2 in the upper left. The robot tries to aggressively to lead the human team towards the more optimal goal 1.

Limitations and Future Work. We view our work as a first step into modeling latent, dynamic human team structures. Although our framework is general to different group dynamics and can scale to various team sizes, we do recognize that the performance degrades when the task complexity increases. Examples include when the number of goals or number of agents becomes too large, as shown in Fig. 4.16. We observe that when increasing the number of agents, the assumption that group dynamics can be explained through local pairwise interactions weakens due to the complexity of interactions. For instance, when we recruited 5 humans to play the predator-prey game, "alliances" emerged where agents that were non-adjacent in the predator-prey chain would team up. These types of dynamics were not observed in two-player games. These types of scenarios are challenging for our models, and the prediction error also accumulates both in the inference process and the optimization process. Further exploration in these complex scenarios is needed to enable our model to be self-aware and corrective.

Another limitation is the reliance on simulated human behavior to test our framework. Further experiments with large-scale human data are needed to support our framework's effectiveness for understanding of noisier human behavior.

Finally, the robot policies that use the graph representation are fairly simple. Although this may be a limitation, it is also promising that simple policies were able to perform well using the proposed graph structures.

For future work, we plan to test our model on large scale human-robot experiments in both simulation and on real robot platforms. Specifically, we plan on using navigation platforms of robot swarms to further improve our model's generalization capacity. We also plan on experimenting with

combining the graph representation with more advanced policy learning methods such as reinforcement learning. We think our graph representation could contribute to multi-agent reinforcement learning in various ways such as reward design and more effective state representation.

Part II

Adaptation to Robots

Chapter 5

Learning from My Partner's Actions: Roles in Decentralized Robot Teams

5.1 Introduction

When teams of robots are deployed in our homes, warehouses, and roads, often these robots must collaborate to accomplish their task. Imagine two robots working together to move a heavy table across a room (see Fig. 5.1). Due to occlusions, each agent can only see some of the obstacles within this room. Thus, the robots need to *communicate* to inform their partner about the obstacles they see. One option is for the robots to explicitly communicate by directly sending and receiving messages: i.e., we could tell our teammate where the obstacles are. But humans utilize more than just explicit communication—we also implicitly communicate through our actions. For example, if our partner guides the table away from our intended trajectory, we might infer that we were moving towards an obstacle, and that there is a better path to follow. Collaborative robot teams—like humans—should also leverage the information contained within their partner's actions to learn about the world.

Unfortunately, interpreting the meaning behind an action is hard. Robots can take actions for *many different reasons*: to exploit what they know, convey information to their partner, elicit information from their partner, or explore the environment. So when we observe our partner applying some force to the table, what (if anything) should we learn from that action? And how do we select actions that our partner can also interpret? In this chapter, we show that assigning roles alleviates these challenges:

Teams of robots can correctly interpret and learn from each other's actions when the team is separated *into* roles*, and each role provides the robots with a distinct reason for acting.*

Returning to our example, imagine that our partner's role is to exploit their current information: they move the table towards the goal while avoiding the obstacles that they can observe. If we know this role, we can now interpret their actions: when our partner applies an unexpected force, it must be because of some obstacle that we did not see. Hence, assigning roles enables us to *learn* from our partner's actions and update our estimate of the system state *naturally*, without requiring additional, explicit communication. In this chapter, we make the following contributions:

Roles in Two-Player Teams. We focus on decentralized two-player teams where each agent sees part of the current state, and together the agents observe the entire state. We show that—without roles—the agent policies are interdependent, and interpreting the actions of our partner leads to infinite recursion (what do you think I think you think, etc.). To remove this interdependence and reach interpretable actions, we introduce two classes of policies: a *speaker* role, where agents exploit what they know, and a *listener* role, where agents learn by modeling their partner as a speaker.

Mimicking Explicit Communication. We explore how roles can be used to make our decentralized team behave like a *centralized* team that communicates explicitly. We find that decentralized teammates which alternate between roles can match the centralized team, but if the agents always maintain the same roles, the team may become unstable. We also reveal that speakers trade-off between stochasticity and communication: to improve overall team performance, speakers should choose more deterministic actions than the centralized policy to clearly convey their observations.

Comparing Implicit to Explicit. We implement roles both in a simulated game and a two-robot manipulation task. Our simulations compare implicitly communicating through actions to explicitly communicating by sending messages, and demonstrate that—when robots leverage roles—implicit communication is almost as effective as explicit communication. In robot experiments, teams that alternated roles successfully communicate their observations and collaborate to avoid obstacles.

Overall, this work is a step towards learning from our partner's actions in decentralized robot teams.

5.2 Related Work

Multi-Agent Teams. Teams of robots have performed tasks such as localization [150], navigation [12], and manipulation [173, 174]. While many works still rely on centralized coordination [29], decentralized multi-agent teams are receiving increasing attention [41, 12]. Within *decentralized control*, the problem is reformulated as a decentralized partially-observable Markov decision process (Dec-POMDP) [6, 22] or coordinator-POMDP [135]. In practice, solving these optimization problems is frequently intractable: related works instead rely on high-level abstractions [7], sparse

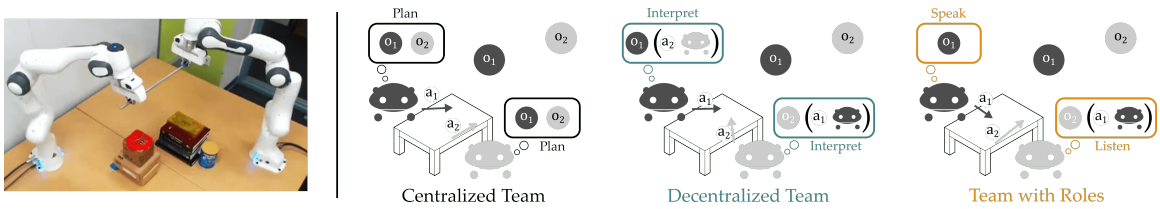


Figure 5.1: (Left) Our problem setting, where two robots must work together to complete a task. Each robot observes different parts of the true system state. (Right) Unlike **centralized teams**, **decentralized teammates** should implicitly communicate through actions: here each agent sees one obstacle, and tries to infer where the other obstacle is based on their partner's actions. Interpreting our partner's actions is hard: there are many reasons why an agent might choose an action. We introduce speaker and listener **roles** so that each agent has a distinct reason for acting, enabling the robots to learn from and implicitly communicate through their actions.

interactions between agents [129], specific problem instances [182], or control approximations [41]. Alternatively, with multi-agent *reinforcement learning* (MARL), agents can learn decentralized policies from trial-and-error [32]. Today's MARL approaches often leverage actor-critic methods to scale to high dimension state-action spaces [121, 69, 56]; however, these methods require offline, centralized training, and must deal with the credit assignment problem (i.e., who is contributing to the team's success?).

Communication. Across both control and learning approaches, communication between agents is key to effective collaboration. The protocol that the agents use to communicate can be either based on pre-defined triggers [170, 92] or learned from training data [180, 55, 148]. But the method that the agents use to communicate is generally *explicit*: the agents have a separate channel with which they directly broadcast and receive messages from their teammates [196]. Unlike these recent works, we will focus on leveraging *implicit* communication through actions.

Roles in Human Teams. Our research is inspired by human-human teams, where different roles naturally emerge in collaborative tasks [133, 74]. For example, when two humans are working together to physically manipulate an object, the agents can identify and assume complementary roles based on their partner's force feedback alone [155]. Roles have also been applied to human-robot interaction: here the robot dynamically adjusts its level of autonomy (i.e., becomes a leader or follower) in response to the force feedback from the human partner [130, 117, 73]. We will *extend* these ideas to robot-robot teams, where we believe that roles can similarly improve collaboration.

5.3 Interpreting Actions via Roles

Consider a decentralized, two-agent team where the agents share a common objective. Each agent observes part of the system state. To make good decisions, both agents need an accurate estimate of the entire state; for instance, we need to know whether there is an obstacle behind us, if there is free space to our right, etc. While we know the aspects of the state that we directly observe, how can we estimate the parts of the system state that our partner sees? One natural solution is to

learn about the environment based on the *implicit* communication contained within our partner's *actions.* In this section, we show that correctly interpreting the meaning behind our partner's actions is challenging when both agents try to learn from their partner and exploit what they observe at the same time. Returning to the table moving example: when our partner applies a force, is this because of what they have learned from our own actions, because of an obstacle behind us, or some combination of both? To correctly infer the unobserved state, each agent must reason over their partner's behavior, and this behavior may in turn depend on the first agent's actions. Accordingly, we here introduce *speaker* and *listener* roles to remove this interdependency: the speaker implicitly communicates relevant parts of the state that they observe to the listener, who learns a more accurate state estimate.

Two-Agent Team. We formulate our two-player team as a decentralized Markov decision process (Dec-MDP) [6]. Let the state space be $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2$ and let the action space be $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$, where $\mathcal{S}_i$ and $\mathcal{A}_i$ are the state and action space for agent i. The team has dynamics $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and receives reward $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. At the current timestep t, agent i observes its own state component s_i^t, and collectively the team observes $s^t = (s_1^t, s_2^t)$. We therefore have that the state is *jointly* fully observable: s^t is known given the current observations of both agents, s_1^t and s_2^t. When making decisions, agent i has access to its history of observations, $s_i^{0:t} = (s_i^0, s_i^1, \ldots, s_i^t)$, as well as the history of actions taken by both agents, $a^{0:t-1}$. For simplicity, we assume that each agent observes its partner's current action selection (our results still hold if they observe the previous action). Hence, the agents have policies of the form $\pi_1(a_1^t \mid s_1^{0:t}, a^{0:t-1}, a_2^t)$ and $\pi_2(a_2^t \mid s_2^{0:t}, a^{0:t-1}, a_1^t)$.

Mimicking Centralized Teams. One approach to control decentralized teams is solving this Dec-MDP; however, the problem is NEXP-complete [22], and often intractable for continuous state and action spaces. We study a different approach: we find policies for both decentralized agents to collectively *mimic* the behavior of a *centralized* team [46, 148]. Imagine that—when moving a table—both teammates know exactly what their partner sees; when we explicitly communicate our observations, we can solve the problem together, and collaborate perfectly to carry the table. We will treat this centralized policy that uses explicit communication as the *gold standard* for our decentralized team, while recognizing that the decentralized team only has access to implicit communication, and may not be able to completely match the centralized team. Importantly, the optimal centralized policy is the solution to an MDP, and can be tractable computed offline (P-complete) [22].

Interdependent Policies. Let the centralized policies be $\pi_1^*(a_1^t \mid s_1^t, s_2^t)$ and $\pi_2^*(a_2^t \mid s_1^t, s_2^t)$. When both decentralized agents choose their actions to best mimic this centralized behavior, we

reach:

$$\begin{aligned}\pi_1(a_1^t \mid s_1^{0:t}, a^{0:t-1}, a_2^t) &\propto \sum_{s_2^{0:t}} \pi_1^*(a_1^t \mid s_1^t, s_2^t) \cdot \pi_2(a_2^t \mid s_2^{0:t}, a^{0:t-1}, a_1^t) \cdot P(s_2^{0:t} \mid s_1^{0:t}, a^{0:t-1}) \\ \pi_2(a_2^t \mid s_2^{0:t}, a^{0:t-1}, a_1^t) &\propto \sum_{s_1^{0:t}} \pi_2^*(a_2^t \mid s_1^t, s_2^t) \cdot \pi_1(a_1^t \mid s_1^{0:t}, a^{0:t-1}, a_2^t) \cdot P(s_1^{0:t} \mid s_2^{0:t}, a^{0:t-1})\end{aligned} \tag{5.1}$$

See the appendix for our complete derivation. We cannot solve Eqn. (5.1) as is because the policies are interdependent: π_1 and π_2 both appear in each other's policy, so that solving for π_1 requires solving π_2 for all possible $s_2^{0:t}$, which requires an inner loop that solves for π_1 over all possible $s_1^{0:t}$, and so on. Going back to our table example: imagine that our partner observes whether there is an obstacle behind us, and we want to infer the likelihood of this obstacle from their actions. This is easy when our partner's actions are only based on this obstacle—but if our partner's behavior is also a response to our own actions, how to we know which aspects of our partner's behavior to learn from? To break this interdependence and recover interpretable actions, we separate our team into two roles: an agent that exploits what it observes (the *speaker*), and an agent that learns from its partner's (the *listener*).

Speaker. A speaker is an agent that makes decisions by purely exploiting their observations $s_i^{0:t}$. Let Agent 1 be the speaker; the speaker policy that best matches π_1^* is:

$$\pi_1(a_1^t \mid s_1^{0:t}) = \sum_{s_2} \pi_1^*(a_1^t \mid s_1^t, s_2^t) \cdot P(s_2^t \mid s_1^{0:t}) \tag{5.2}$$

More generally, any policy of the form $\pi_i(a_i^t \mid s_i^{0:t})$ is a speaker. Because speakers react to their observations, these actions convey information about the parts of the state they see to their teammates.

Listener. A listener makes decisions based on its own observations while also learning an improved state estimate from its partner's behavior. If Agent 2 is the listener, the policy that matches π_2^* is:

$$\pi_2(a_2^t \mid s_2^{0:t}, a^{0:t-1}, a_1^t) \propto \sum_{s_1^{0:t}} \pi_2^*(a_2^t \mid s_1^t, s_2^t) \cdot \pi_1(a_1^t \mid s_1^{0:t}) \cdot P(s_1^{0:t} \mid s_2^{0:t}, a^{0:t-1}) \tag{5.3}$$

Compared to (5.1), now the listener models its partner as a speaker, and solving for π_1 (the speaker policy) does not depend on π_2 (the listener policy). More generally, we consider any policy of the form $\pi_i(a_i^t \mid s_i^{0:t}, a^{0:t-1}, a_j^t)$ as a listener if it interprets its teammate's actions with $\pi_j(a_j^t \mid s_j^{0:t})$.

5.4 Leveraging Roles Effectively

Now that we have defined roles, let us return to our table carrying example. Imagine that we are the speaker and our partner is the listener: what is the best speaker policy for us to follow? Should we always remain a speaker, or do we need to switch speaker and listener roles with our teammate? And if our decentralized team uses roles, when can we fully match the behavior of a centralized team? In this section we explore these questions, and analyze how roles operate within simplified settings.

5.4.1 When Can We Use Roles to Match a Centralized Team?

Roles enable implicit communication through actions. This implicit communication is typically less informative than explicitly sharing observations; but when an agent's actions can completely convey their observed state, robots can leverage roles to fully match the behavior of a centralized team:

Theorem 1. *If there exist surjective functions $g_1 : \mathcal{A}_1 \to S_1$ and $g_2 : \mathcal{A}_2 \to S_2$, a decentralized team using speaker and listener roles can match the optimal actions of a centralized team.*

Proof. The decentralized team matches the centralized team's performance by communicating with actions while rapidly alternating between speaker and listener roles. Define a_1^* as the optimal centralized action for agent 1, and let $\bar{a}_1$ be a naive action that completely conveys what agent 1 observes: $g_1(\bar{a}_1) = s_1$. We choose the speaker action to be $\bar{a}_1$ and the listener action to be $a_2^* + (a_2^* - \bar{a}_2)$, where the listener can compute a_2^* because it observes s_2 and infers s_1 from $g_1(\bar{a}_1)$. The agents change roles: agent 1 is the speaker for time $[t, t + dt)$ and agent 2 is the speaker for time $[t + dt, t + 2dt)$. Taking the limit as $dt \to 0$ (i.e., as the roles change infinitely fast), the team's action a becomes:

$$a^t = \lim_{dt \to 0} \frac{a^t + a^{t+dt}}{2} = \lim_{dt \to 0} \frac{1}{2} \left(\begin{bmatrix} \bar{a}_1^t \\ a_2^{*t} + (a_2^{*t} - \bar{a_2}^t) \end{bmatrix} + \begin{bmatrix} a_1^{*t+dt} + (a_1^{*t+dt} - \bar{a}_1^{t+dt}) \\ \bar{a}_2^{t+dt} \end{bmatrix} \right) = a^{*t}$$

and so the decentralized team's action converges to the optimal action of the centralized team. □

Intuitively, consider the table example with two dynamic obstacles, one of which is observed by each teammate. When we are the speaker, we apply a force with a direction and magnitude that conveys our obstacle's position to the partner; likewise, when our partner is the speaker, their action informs us where their obstacle is. By quickly switching speaker and listener roles we can both understand the state, and match the behavior of a team that explicitly tells one another about the obstacles.

5.4.2 Analyzing Roles in Linear Feedback Systems

To better understand how roles affect decentralized teams, we specifically focus on teams controlled using linear feedback. Here the centralized policy is $a^* = -K^*s$, where K^* is the desired control gain; for instance, this policy could be the solution to a linear-quadratic regulator (LQR). Assuming our decentralized team is similarly controlled with $a = -Ks$, we first determine whether alternating speaker and listener roles is necessary to ensure system stability. We next consider situations where the centralized policy includes stochastic behavior, and we identify how speakers should optimally trade-off between mimicking this desired noise and effectively communicating with the listener.

Do We Need to Change Roles? Imagine that we are the speaker within the table-carrying example. In the best case, our actions completely convey our observations to our partner. But if we are never a listener, we never know what our partner observes; and, if the team's behavior depends on our own actions, this can lead to situations where we are unable to collaboratively accomplish the task:

Proposition 1. *If the teammates never change speaker and follower roles, and the team is attempting to mimic a centralized controller $a^* = -K^*s$, there exist controllable system dynamics for which the decentralized team $a = -Ks$ becomes unstable for any choice of K.*

Proof. We prove this by example. Let agent 1 always be the speaker and let agent 2 be the listener. Consider the following controllable system with linear dynamics $\dot{s} = As + Ba$, where $a = -Ks$:

$$\begin{bmatrix} \dot{s}_1 \\ \dot{s}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} K_{11} & 0 \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \tag{5.4}$$

Note that $K_{12} = 0$; this is because the speaker makes decisions based purely on their own state, s_1. The listener is able to perfectly infer s_1 since it observes the speaker's action a_1, and $s_1 = -K_{11}^{-1}a_1$. For this team to have stable dynamics the eigenvalues of $A - BK$ must have a strictly negative real part. But here the eigenvalues are: $1 \pm \sqrt{-K_{11}}$. Hence, no matter what speaker and listener gains K_{11}, K_{21}, and K_{22} we choose, the decentralized team becomes unstable. □

How Should We Speak and Listen with Noise? Returning to our table carrying example, we now know that we should alternate speaker and listener roles even when our actions completely convey our observations. But what if we cannot perfectly measure the actions of our partner? For instance, imagine that instead of knowing exactly what force our teammate applies, we have a noisy estimate. As long as this estimate is unbiased, we can still match the expected behavior of the centralized team by treating these noisy actions as perfect measurements, and speaking or listening normally:

Proposition 2. *If the teammates incorrectly sense their partner's action a_i as $a_i + n_i$, where n_i is unbiased noise, the decentralized team can match the centralized controller action $a^* = -K^*s$ in*

expectation by speaking and listening as if $a_i + n_i$ were the teammate's true action.

Proof. Let agent 1 be the speaker, let agent 2 be the listener, and let the decentralized controller be $a = -Ks$, where K is shown in Eqn. (5.4). The speaker takes action $a_1 = -K_{11}s_1$, but the listener measures $a_1 + n_1$, and infers the speaker's state as $\hat{s}_1 = s_1 - K_{11}^{-1}n_1$. Accordingly, when the listener acts based on $\hat{s}_1$, their action has an additional term $K_{21}K_{11}^{-1}n_1$. Next the agents switch roles, and the first agent (i.e., the listener) acts with an additional term $K_{12}K_{22}^{-1}n_2$. When the agents rapidly alternate between speaker and listener roles, the team's behavior includes these terms:

$$a = -K^*s + \begin{bmatrix} K_{12}^* K_{22}^{*,-1} n_2 \\ K_{21}^* K_{11}^{*,-1} n_1 \end{bmatrix}, \quad K^* = \begin{bmatrix} K_{11}^* & K_{12}^* \\ K_{21}^* & K_{22}^* \end{bmatrix}, \quad \mathbb{E}_n[a] = -K^*s = a^* \tag{5.5}$$

This matches the centralized team's action in expectation if n_1 and n_2 are unbiased. □

Noisy action measurements are undesirable but often unavoidable. Conversely, there are cases where the centralized policy itself *recommends* stochastic actions: i.e., when an agent sees an obstacle in front of it, it should go around on the right-side 30% of the time, and on the left otherwise. The speaker and listener can *choose* whether or not to incorporate this stochasticity. Intuitively, we might think that both the speaker and listener should match the optimal centralized policy; but, when the speaker's actions become more stochastic, it is harder for the listener to correctly interpret what the speaker has observed. Imagine we are the listener: the more random the speaker's action is, the less information we have about what the speaker sees, which makes it more challenging for us to learn the system state. This leads to a trade-off between speaker noise and overall team performance:

Proposition 3. *If the centralized policy is stochastic, so that $a^* = -K^*s + n$, and n is sampled from a Gaussian distribution $n \sim \mathcal{N}(0, diag(w_1^2, w_2^2))$, the speaker policy that minimizes the Kullback-Leibler (KL) divergence between the decentralized and centralized policies has a variance less than or equal to the corresponding variance of the centralized policy.*

Proof. Let the first agent be the speaker and let the second agent be the listener. The decentralized team takes action $a = -Ks + m$, where $m \sim \mathcal{N}(0, diag(\sigma_1^2, \sigma_2^2))$. We solve for the variances σ_1^2 and σ_2^2 that minimize the KL divergence between the decentralized and centralized policies:

$$\min_{\sigma_1^2, \sigma_2^2} \ \mathrm{KL}\Big[\mathcal{N}\big(-Ks, diag(\sigma_1^2, \sigma_2^2)\big),\ \mathcal{N}\big(-K^*s, diag(w_1^2, w_2^2)\big)\Big] \tag{5.6}$$

Selecting K to best match K^*, and taking the expectation of Eqn. (5.6), the optimal variances are:

$$\sigma_1^2 = \frac{K_{11}^2 w_1^2 w_2^2}{K_{11}^2 w_2^2 + K_{21}^2 w_1^2} \leq w_1^2, \quad \sigma_2^2 = w_2^2 \tag{5.7}$$

See the Appendix Chapter A for our full derivation. Hence, while the listener should match the variance of the corresponding centralized policy, the speaker should intentionally choose a variance

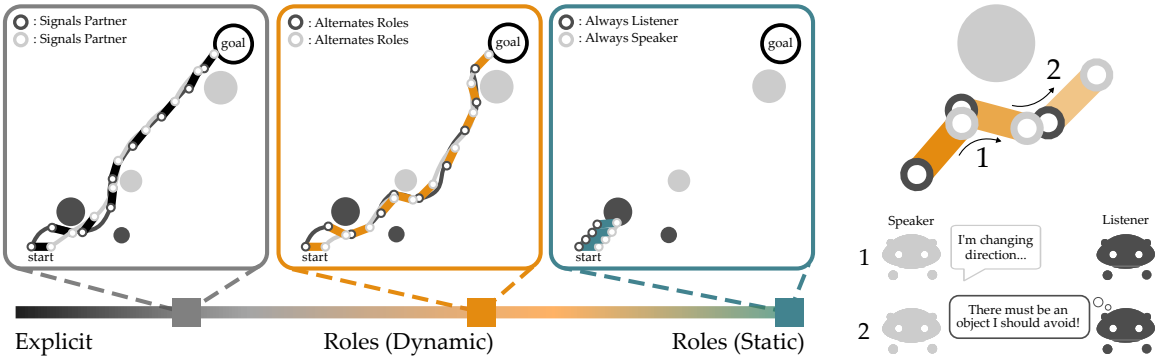


Figure 5.2: Simulated task. (Left) Two agents collaborate to carry a rigid object while avoiding obstacles. Each agent can only see the obstacles matching their color. In this example, the *explicit* and *dynamic roles* teams are able to negotiate the obstacles to reach the goal, while the *static roles* team fails. (Right) Example of implicit communication via actions: here the speaker sees an obstacle, and abruptly moves to the right (1). The listener updates its state estimate based on this unexpected motion, and also moves to avoid the unseen obstacle (2).

$\sigma_1^2 \leq w_1^2$. □

Summary. If two decentralized agents are collaborating with roles, they can match the behavior of a centralized team when their actions completely convey their observations. But even in these cases, alternating between speaker and listener roles is necessary; the team may become unstable if their roles never change. When choosing how best to speak and listen, greedily mimicking the centralized policy is effective, and robust to noisy, unbiased measurements of the partner's actions. In situations where the centralized policy is stochastic, however, a trade-off emerges: speakers should select more deterministic actions in order to better convey their observations to the listener.

5.5 Experiments

Roles are sufficient when actions can completely convey the state an agent observes. But what about more general situations, where the observation space has a higher dimension than the action space? And how do our proposed roles function on actual robot teams? Here we explore these challenging cases in a simulated game (see Fig. 5.2) and a two-robot manipulation task (see Fig. 5.4). We compare fixed and dynamic role allocations to different amounts of explicit communication. Overall, we find that there is a *spectrum* across explicit and implicit communication, and that implicit communication via roles approaches the performance of equivalent explicit communication.

5.5.1 Simulated Table-Carrying Game with Implicit and Explicit Communication

We simulated a continuous state-action task in which two point robots carried a table across a plane (see Fig. 5.2). Each robot observed half of the obstacles within this plane, and so together

Table 5.1: Success rate λ when both robots knew the geometry of the obstacles *a priori*, but not their locations. Each agent implicitly communicated the positions of the obstacles that they could see through their actions. Importantly, the agents could completely communicate the position of the closest obstacle with their current action: hence, the success rate of *explicit* teams (not listed) is almost identical to that of *dynamic* teams.

	Roles (Dynamic)			**Roles (Static)**	
Obstacles	$T = 1$	$T = 4$	$T = 16$	Speaker-Listener	Speaker-Speaker
$n = 2$	0.995	0.914	0.827	0.872	0.757
$n = 4$	0.982	0.786	0.656	0.735	0.545
$n = 8$	0.924	0.586	0.434	0.539	0.305

the team needed to communicate to obtain an accurate state estimate. We compared using *explicit* and *implicit* communication. During explicit communication, agents sent and received messages that contained the exact location and geometry of the closest obstacle. By contrast, during implicit communication the agents leveraged roles to learn from their partner's actions. An example is shown in Fig. 5.2: here the speaker sees an obstacle—that the listener cannot observe—and changes its motion to implicitly indicate to the listener that there is an obstacle directly ahead.

Independent Variables. We varied the (a) communication strategy and (b) task complexity. There were three levels of communication strategy: *explicit*, *dynamic roles*, and *static roles*. Within *explicit* and *dynamic roles*, we also varied the number of timesteps T between communication; i.e., the *explicit* teams could only send messages about the closest obstacle every T timesteps, or, analogously, the speaker and listener roles alternated after T timesteps. In order to adjust the task complexity, we increased n, the total number of obstacles in the environment.

Dependent Measures. Within each simulation, two agents held a rigid table, and tried to carry that table to the goal without colliding with an obstacle. We measured the *success rate* (λ) of reaching the goal over 1000 randomly generated environments. As a baseline, we also tested fully centralized teams: these centralized teams reached the goal in all environments ($\lambda = 1$).

Hypotheses:

H 6. *Dynamically alternating roles leads to better performance than fixed role allocations, even in cases where actions can completely convey the location of the closest obstacle.*

H 7. *In environments where actions cannot completely convey the closest obstacle, dynamic role allocations perform almost as well as explicitly communicating the closest obstacle.*

H 8. *Implicit communication via roles is robust to noisy action observations.*

We provide additional details about this simulated game in the appendix and supplementary code.

5.5.2 Rapidly Changing Roles Outperforms Static Roles

We first tested hypothesis **H6** in a setting where each agent knew the geometry of all the obstacles in the environment (i.e., all obstacles were circles with the same radius). Since the obstacle shape is

known and fixed, agents could fully convey the closest obstacle's (x, y) location through their 2-DoF actions. Our results are shown in Table 5.1; we point out that *dynamic roles* and *explicit* are almost the same in this case, and so we focus on comparing *dynamic roles* to *static roles*. For each tested number of obstacles, the teams that rapidly alternated roles outperformed teams with fixed roles. Indeed, when there are only $n = 2$ obstacles, the mapping from action space to observation space was surjective, and the *dynamic roles* team converged towards $\lambda = 1$ as $T \to 0$ (Theorem. 1). But dynamically alternating roles was not always better: when the teammates changed roles too slowly $(T = 16)$, their performance was actually worse than maintaining a constant speaker and listener.

5.5.3 Implicitly Communicating via Roles Competes with Explicit Communication

Next, we explored hypothesis **H7** in more complex settings where the agent's current action could not completely convey what they observed to their partner. Here the robots did not know the obstacle geometry *a priori*; instead, each obstacle radius was randomly sampled from a uniform distribution. Teams with *roles* had to try and implicitly communicate about both the obstacle location and shape—by contrast, *explicit* teams could send messages to their partner that completely conveyed the closest obstacle. Our findings are visualized in Fig. 5.3. We see a spectrum in performance across *explicit*, *dynamic roles*, and *static roles*. Although directly sending obstacle information is always better than learning from actions, implicitly communicating through *dynamic roles* is almost as successful as *explicitly* communicating with our partner at the same time interval T. This suggests that leveraging roles to learn from our partner's actions can be nearly as effective as direct communication, even in cases where the observations cannot be completely conveyed in a single action.

5.5.4 Roles with Noisy Action Observations Match Noisy Messages During Explicit Communication

So far we have conducted simulations in settings where the agents can perfectly measure the communication of their partner. Now we take a step back, and consider hypothesis **H8** when the communication channel *itself* is noisy. This encompasses scenarios where the partners cannot perfectly measure each other's actions, or, analogously, when the explicit messages are corrupted. Our results with zero mean Gaussian noise are listed in Table 5.2. As expected, including noise decreased performance across both *explicit* and *dynamic roles*. But teams with *dynamic roles* that rapidly alternated were still almost on par with *explicit* teams that communicated in realtime, demonstrating that roles were as robust to noisy actions as explicit teams were to noisy messages. Similar to before, fixed speaker-listener teams were more successful than teams that slowly alternated roles.

Table 5.2: Success rate λ when the communication was noisy. We simulated zero-mean Gaussian noise, and chose the variance so that the coefficient of variation was 0.1. Both explicit communication (noisy messages) and implicit communication (noisy action observations) had the same noise ratio. Dynamic roles performed almost on par with realtime explicit communication, where both agents exchanged messages at every timestep.

	Explicit	**Roles (Dynamic)**			**Roles (Static)**	
Obstacles	$T = 0$	$T = 1$	$T = 4$	$T = 16$	Speaker-Listener	Speaker-Speaker
$n = 2$	0.925	0.884	0.818	0.787	0.829	0.757
$n = 4$	0.833	0.747	0.669	0.606	0.655	0.545
$n = 8$	0.553	0.512	0.416	0.357	0.398	0.305

5.5.5 Manipulation Experiments with Two Robot Arms

We implemented our *dynamic role* and *static role* strategies on two decentralized robot arms (Panda, Franka Emika). Our experimental setup is shown in Fig. 5.4: the robots were controlled using separate computers, and were tasked with placing a rod on the table top without any explicit communication. In order to complete this task, the robots had to negotiate two obstacles—but, like in our simulations, each robot could only see one of these obstacles. Because of their differing knowledge about the system state, the robots originally had opposite plans about how to move the rod to the table: one robot wanted to move the rod forwards (with respect to the camera), while the other robot planned to move the rod backwards. We expect intelligent robots to recognize that there is a *reason* why their partner disagrees with them, and *learn* from their partner's actions in realtime to update their estimate of the system state. We controlled the robots using static speaker-speaker and speaker-listener roles, as well as dynamic roles that alternated every 0.5 s. During the experiments, only the *dynamic roles* team inferred the obstacles that their partners observed, and aligned their actions to successfully place the rod on the table (see Fig. 5.4). Please also refer to our video submission.

5.6 Conclusion and Discussion

Summary. Decentralized robot teams should learn about what their partner observes based on their partner's actions, but this is not possible when both agents attempt to simultaneously exploit their observations and communicate with their partner. We have therefore introduced separate speaker and listener roles: our analysis shows that teammates which dynamically alternate roles theoretically and experimentally approach the performance of teammates that can explicitly communicate.

Limitations and Future Work. This work is limited to Dec-MDPs, where the agents collectively observe the full system state. But we recognize that often there are parts of the state that neither agent can fully observe; accordingly, our future work will focus on extending roles to Dec-POMDPs.

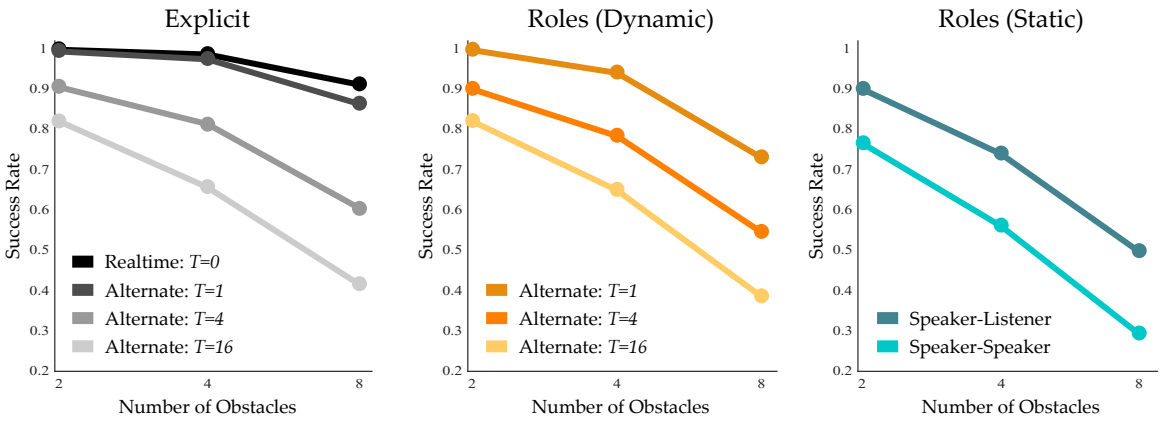


Figure 5.3: Success rate λ when the robots did not know the obstacle geometry *a priori*. Not only was the observation space higher dimensional than the action space, but agents could not even convey the closest obstacle's position and radius in a single action. There is a spectrum in performance across both communication type and frequency. For realtime explicit communication, the agents fully convey the closest obstacle at every timestep.

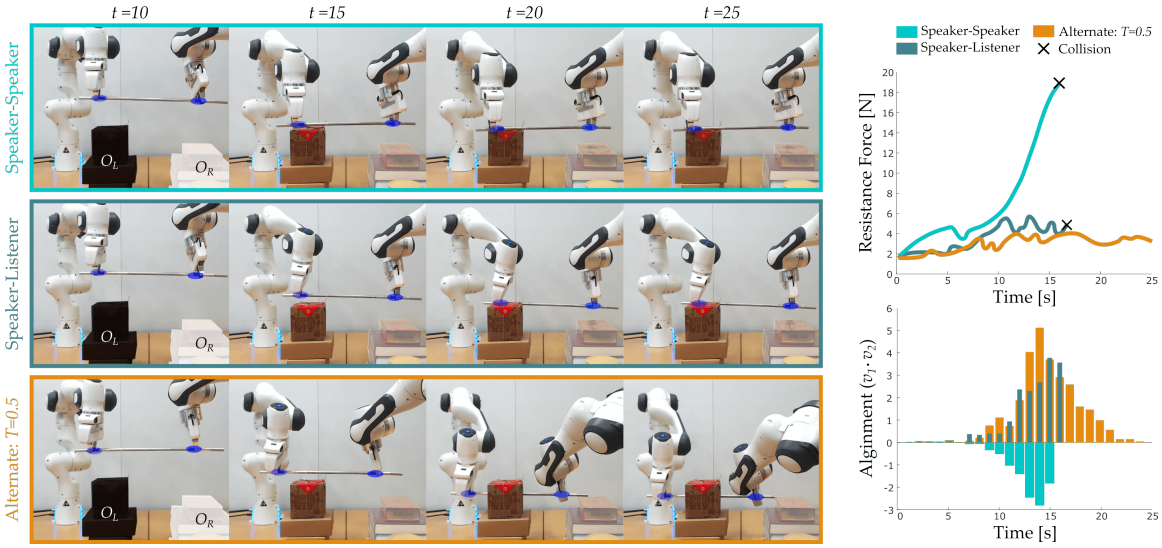


Figure 5.4: Two decentralized robot arms tasked with placing a rod on the table while implicitly communicating via roles: the left robot sees obstacle O_L and the right robot sees O_R. (Left) The behavior of *static role* and *dynamic role* teams after t s. Both static role teams collided with obstacle O_L because they failed to mutually communicate their observations. The team that alternated roles successfully reasoned over each other's actions, conveyed the obstacle positions, and aligned their actions. (Right) We plot the norm of the force the two agents exerted on one another, as well as the alignment between the two agent's end-effector movements.

Part III

Adaptation to Tasks

Chapter 6

Learning Tool Morphology for Contact-Rich Manipulation Tasks with Differentiable Simulation

6.1 Introduction

Humans are distinct from other species in that tool use is a defining and universal characteristic for us [62]. This suggests that in the pursuit of equipping robots with human-like dexterity, tools may play an important role. Robots are already using various tools in a range of contact-rich manipulation tasks. For example, to make knots, robots can use a tri-needle to maintain the loop [166]. For cooking, robots use spatulas to flip pancakes [186] and skewers to pick food for assistive feeding [181, 65]. While tools greatly influence how robots interact with the environment in these contact-rich tasks, most works focus on learning how to use existing tools. Little attention is paid to optimal tool design. Rather than forcing robots to use pre-defined tools, we aim to intelligently adapt the tools to the tasks, thus helping robots become more effective.

In this work, we aim to develop a general framework for learning robust tool morphology for contact-rich tasks. Relevant to our setting, works on aerodynamic design [123] and vehicle component design also tackle the problem of finding an optimal shape. However, these prior works do not focus on tool design for contact-rich tasks and thus do not require complex contact modeling that is necessary in our scenarios. Another relevant line of research investigates robot gripper design. Some of these works aim to discover gripper designs for grasping a wide range of objects [198] or executing re-orientation primitives [159]. However, they do not give a way to optimize the gripper shape for more complex tasks. Another recent work [194] gives a way to optimize morphology for a given task objective. However, as we will show in our experiments, it does not necessarily generalize to task

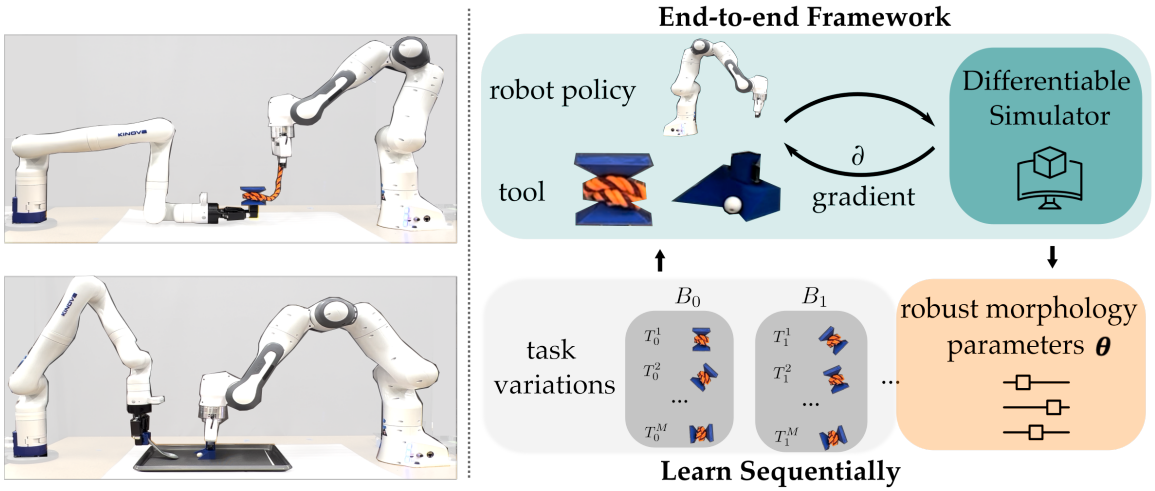


Figure 6.1: We build an end-to-end framework for learning tool morphologies suitable for contact-rich tasks. Our goal is to learn a tool morphology for a given scenario that is robust to task variations. We achieve this with a method based on continual learning that trains on a sequence of task variations.

variations, such as different initial object poses. Overall, we aim to automatically design tools for a given task objective, such that these tools are robust to task variations.

To obtain an optimal shape that minimizes a task-relevant objective, previous work on gripper design employed heuristics for guiding their search [64], or needed Monte Carlo estimation of the gradients to attempt gradient-based optimization [167]. When a robot uses a tool in a contact-rich manipulation task and knows the corresponding dynamics model, it is possible to directly get the exact gradients by differentiation; this enables better numerical stability and faster convergence [42]. To this end, we leverage recent advances in differentiable physical simulators [76, 44, 194] and build an end-to-end framework for learning tool morphology suitable for contact-rich tasks. To design an effective tool for a specific scenario while maintaining ability to handle task variations, we optimize the tool morphology over a distribution of task variations. This brings in two challenges. First, the training process for learning to handle the entire distribution can be computationally expensive. Second, due to the complex dynamics of contact-rich tasks, the underlying optimization landscape is highly non-linear and rugged, which makes it more difficult for optimization to converge to a good solution. To tackle these challenges, we propose an approach based on continual learning that samples task variations and conducts optimization in a sequential manner. Our insight is to re-interpret continual learning as robust optimization framework for problems with challenging loss landscapes. Compared to prior work, we consider a wider range of tasks and objects, including deformable objects, and achieve an improved task performance in simulation.

Furthermore, we demonstrate that the tools obtained with our approach are effective for helping real robots complete the given tasks in reality.

6.2 Related Work

Morphology Optimization in Manipulation. In this work, we explore the promise of fully differentiable end-to-end optimization of tool morphology with the help of differentiable physics. *Gripper design* is one related problem. Some classic works provide guidelines for manually designing grippers guided by practical insights [33]. Recent works aim to learn an optimal design within a given design space. Evolutionary strategies are employed in [127] to optimize both the robot morphology and the controller. In [191], the authors optimize gripper quality using a set of manually designed metrics. Another popular paradigm for gripper customization is imprint-based methods [168, 188]. However, these methods only produce customized finger geometry, where an object 3D model, grasp pose, and task description are specified by the user. A survey in [79] reviews other examples of automated finger design. Further recent examples include a gradient-free method [147], and a gradient-based shape generation method with non-differentiable simulation for training [70]. Most closely related to our work is DiffHand [194], where a differentiable simulator is developed to enable co-design of robot morphology and control by optimizing task-specific objectives. However, this work only considers specific object initial states, and thus does not learn a morphology that would generalize over various initial object states.

Tool design is related to gripper design but unlike grippers and fingers, tools are typically not rigidly attached to robots. This opens future possibilities to study a) re-grasping of tools to improve versatility and dexterity of manipulation robots, and b) the use of multiple tools at the same time (e.g. for dual-arm manipulation). While we do not focus on these aspects in our current work, our formulation facilitates exploring them in the future. Works that consider selecting tools and optimizing policies for tool use are common in robotics literature, e.g. [171, 184], where [184] employs a differentiable simulator. However, the vast majority of these works do not optimize tool shape. Related work for tool *morphology optimization* includes MacGyvering [132]. However, it assumes access to a 'reference tool', and aims to construct a tool from a set of available parts. Instead, we consider the problem of evolving tool morphology from an initial shape without assuming prior knowledge about the optimal tool shape. Furthermore, we use end-to-end differentiable simulation, and leverage differentiability at all levels of the optimization pipeline.

Differentiable Simulation. Instead of using heuristics or search algorithms for optimizing morphology, we leverage differentiable simulation for directly obtaining analytical gradients with respect to the final task objective. Differentiable simulators have been developed for rigid bodies [194, 76, 190] and deformable objects [111, 134]. In this work, we adopt the differentiable simulator and morphology representation from DiffHand [194]. However, our framework is agnostic to the choice of differentiable simulator. Differentiable simulation already enabled some impressive results: e.g. system identification (real-to-sim) and control optimization for cutting [75]; solving a dynamic ball-in-cup task in 4 minutes on a real robot [122]. These works consider advanced phenomena, such

as modeling deformation. However, they do not address the aspect that could be especially challenging for computing gradients – making and breaking contacts in contact-rich tasks. Contact-rich scenarios bring a new level of complexity. They yield sharp changes in the loss landscapes and could make gradient-based optimization difficult [179, 8]. Our focus on tool morphology requires us to address this challenge because tools interact with objects in the scene. Hence, we propose a method that not only employs differentiability, but also leverages a continual learning formulation of the problem to tackle the optimization challenges that arise in contact-rich scenarios.

Continual Learning. Continual Learning considers the problem of learning to solve a sequence of tasks (or task variations), with the objective to perform well on the current task without forgetting what has been learned from previous tasks [45, 70, 104]. The tasks are usually provided as a stream, i.e. data from previous tasks is usually not retained due to memory limitations. There are different categories of continual learning methods. For example, replay methods [154, 115] usually store a fixed number of samples in the replay buffer and use these to construct a distillation loss between previous and current model predictions. This distillation loss encourages the model not to forget what it previously learned. Parameter isolation methods [125, 169] divide model parameters into different subsets and fix the subset of parameters learned with previous tasks when learning a new task. In this way, they prevent the model from forgetting previous tasks and also improve the training stability. We build upon these works and leverage the insight that continual learning could be re-interpreted as a framework for robust optimization suitable for problems with non-smooth dynamics that yield challenging loss landscapes, such as optimizing manipulation tool morphology.

6.3 End-to-end Framework with Differentiable Simulation

In this section, we describe how we learn tool morphology in an end-to-end manner. In Sec. 6.3.1, we first formalize the task where the tool will be used. With the deformation based morphology parameterization described in Sec. 6.3.2, we obtain a low-dimensional design space. Finally, we show the end-to-end morphology parameter learning pipeline for a single task variation in Sec. 6.3.3.

6.3.1 Problem Statement

We formulate the overall manipulation task as a discrete-time Markov Decision Process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, reward r, discount factor γ, and ρ_0 as the distribution of the initial state $\boldsymbol{s}_0$: $(\mathcal{S}, \mathcal{A}, \mathcal{T}_{\boldsymbol{\theta}}, r, \gamma, \rho_0)$. We parameterize the transition function $\mathcal{T}_{\boldsymbol{\theta}}(s, a)$ by $\boldsymbol{\theta}$, which denotes a vector of tool morphology parameters as will be detailed in Sec. 6.3.2. Different tool morphologies influence how the robot has to interact with an object to maximize reward. Imagine we are pushing peas onto a scoop with a tool that has a simple rectangular shape. It is likely that the peas will roll off to the side requiring the robot to re-orient the tool while pushing. However, if the tool is shaped to have a concave recess in the center, the peas would be less likely to roll off

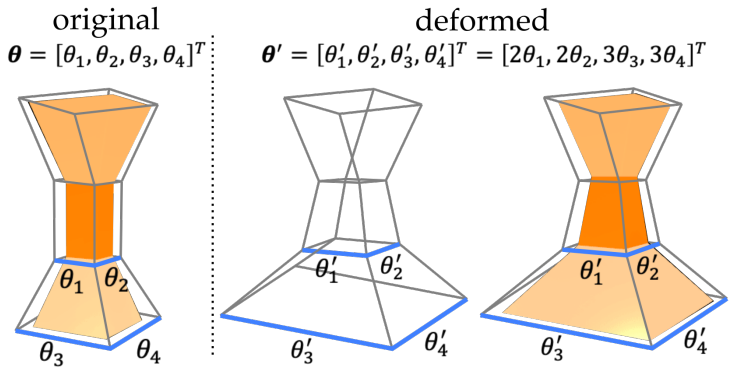


Figure 6.2: An example of morphology parameter vector $\boldsymbol{\theta}$ and shape deformation. Here, the morphology is parameterized by $\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \theta_4]^T$, where $\theta_1, \ldots, \theta_4$ represent the lengths of the four segments highlighted in blue. Upon updating morphology parameters $\boldsymbol{\theta}$ to $\boldsymbol{\theta}'$, we first get the deformed cage vertices, then get the full mesh, as described by Eqn. (6.1).

and escape while being pushed. This illustrates that different $\boldsymbol{\theta}$s yield different MDPs, since they change the transition function $\mathcal{T}_{\boldsymbol{\theta}}(s, a)$. Our goal is to learn the optimal morphology parameters $\boldsymbol{\theta}^*$ that maximize the task reward when robot executes a control policy π. This framework is general enough in principle to allow joint learning of morphologies and control policies. In practice, in this work we focus on learning the tool morphology[1]. To be consistent with notation in the literature on differentiable simulation, instead of reward maximization we describe the optimization problem as loss minimization.

6.3.2 Morphology Parameterization using Cage-Based Deformation

Cage-based deformation techniques are a common tool for deforming meshes in graphics applications [137] and has previously been used to learn optimal hand morphology in a robotics context [194]. We adopt the same morphology parametrization. Typically, a cage is a closed, low resolution mesh that envelopes the high-resolution mesh of the object we want to deform. Given an initial mesh $\mathcal{M}$ of the object and the cage $\mathcal{C}$ around it, we use $\mathbf{m_i} \in \mathcal{M}$ to denote the position of the i^{th} mesh vertex, and $\mathbf{c_j} \in \mathcal{C}$ to denote the position of the j^{th} vertex in the cage. Cage-based defromation establishes a linear mapping from the cage vertices $\mathbf{c_j} \in \mathcal{C}$ to each mesh point $\mathbf{m_i} \in \mathcal{M}$ by computing deformation weights w_{ij} defined by Eqn. (6.1) below:

$$\mathbf{m_i} = \sum_{j, \mathbf{c_j} \in \mathcal{C}} w_{ij} \mathbf{c_j}, \qquad \sum_{j, \mathbf{c_j} \in \mathcal{C}} w_{ij} = 1, \qquad \forall i, \mathbf{m_i} \in \mathcal{M} \tag{6.1}$$

[1] In this work, we focus on morphology learning and regard policy learning as a separate line of work. However, our framework is compatible with policy learning as well. We show in the supplement video that existing methods, e.g. DiffHand [194] have significant difficulties with joint optimization of policy and morphology when presented with task variations. Hence, this is still an open problem for future work. Please see sites.google.com/stanford.edu/learning-tool-morphology for more details.

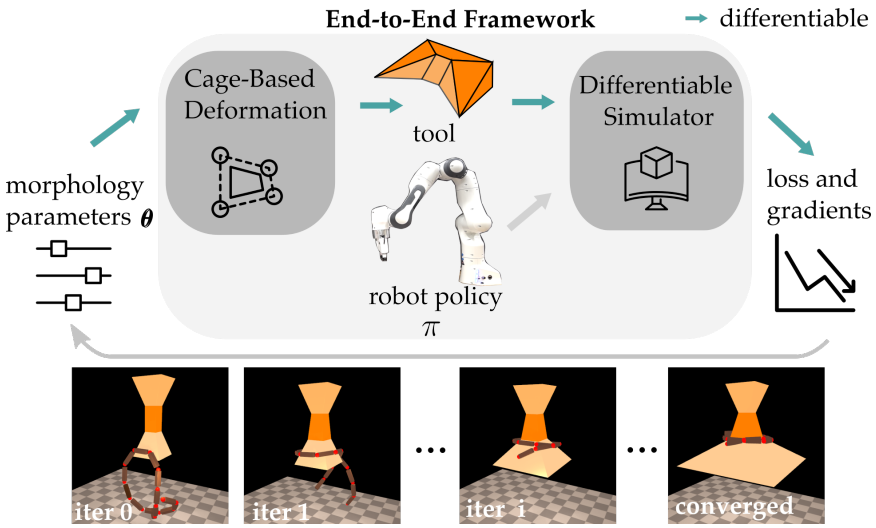


Figure 6.3: Visualization of our end-to-end pipeline for learning tool morphology. We first obtain the tool parametrized by morphology parameters $\boldsymbol{\theta}$ with cage-based deformation. Then we execute the policy π with this tool in the simulator and output the loss. Since all of these operations are differentiable, we can get the analytical gradients to update $\boldsymbol{\theta}$. We also show the evolving shape for a winding tool. The bottom part gradually becomes larger, which prevents the rope from slipping off.

In this work, we adopt the Mean Value Coordinate method [94, 194] for computing the deformation weights w_{ij}. Using these weights, we can manipulate the low-resolution cage vertices to deform the high-resolution mesh. Suppose we deform the cage $\mathcal{C}$ and obtain new cage vertices $\mathbf{c}'_\mathbf{j} \in \mathcal{C}'$. Then, we can compute the deformed mesh vertices as $\mathbf{m}'_\mathbf{i} \in \mathcal{M}'$, with $\mathbf{m}'_\mathbf{i} = \sum w_{ij}\mathbf{c}'_\mathbf{j},\ \forall j, \mathbf{c}'_\mathbf{j} \in \mathcal{C}'$.

To get an even more compact representation of the tool morphology, we further extract the high-level morphology attributes (e.g. tool segment length, height, width) and denote these morphology parameters as $\boldsymbol{\theta} \in \mathbb{R}^d$. Fig. 6.2 shows a basic example. When we change the morphology parameters corresponding to segment width from $\boldsymbol{\theta}$ to $\boldsymbol{\theta}'$, we first map $\boldsymbol{\theta}'$ to a deformed cage $\mathcal{C}'$, obtaining vertices $\mathbf{c}'_\mathbf{j} \in \mathcal{C}'$. Then, we compute the deformed tool mesh $\mathbf{m}'_\mathbf{i} \in \mathcal{M}'$ using Eqn. (6.1). The mapping from $\boldsymbol{\theta}'$ to cage $\mathcal{C}'$ and that from cage $\mathcal{C}'$ to object mesh $\mathcal{M}'$ are both linear, so the overall mapping from $\boldsymbol{\theta}'$ to mesh $\mathcal{M}'$ is linear as well.

6.3.3 End-to-end Pipeline Based on Differentiable Simulation

With morphology parameterized by Cage-Based Deformation, we can now build our end-to-end tool morphology learning framework with differentiable simulation. In this work, we adopt DiffHand [194] as our backbone simulator and build our pipeline upon it. However, our framework is agnostic to the choice of differentiable simulators and thus compatible with other differentiable simulators as well.

We visualize our end-to-end pipeline in Fig. 6.3. With the cage-based deformation, we effectively parametrize the tool with morphology parameter $\boldsymbol{\theta}$. We then instantiate the task with the deformed tool, initialize it with initial state $\boldsymbol{s}_0$, and execute the robot policy π in the differentiable simulator. We then compute the gradients of the task-dependent loss $\mathcal{L}(\pi; \boldsymbol{s}_0, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$:

$$\frac{\partial \mathcal{L}(\pi; \boldsymbol{s}_0, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{m_i \in \mathcal{M}, c_j \in \mathcal{C}} \frac{\partial \mathcal{L}(\pi; \boldsymbol{s}_0, \boldsymbol{\theta})}{\partial m_i} \cdot \frac{\partial m_i}{\partial c_j} \cdot \frac{\partial c_j}{\partial \boldsymbol{\theta}}. \tag{6.2}$$

$\frac{\partial \mathcal{L}(\pi; \boldsymbol{s}_0, \boldsymbol{\theta})}{\partial m_i}$ is provided by the differentiable simulator. $\frac{\partial m_i}{\partial c_j}$ and $\frac{\partial c_j}{\partial \boldsymbol{\theta}}$ are derived from the cage-based deformation in Sec. 6.3.2. We then take a gradient step on the tool morphology parameter $\boldsymbol{\theta}$ and repeat the process until convergence.

6.4 Continual Learning for Robust Tool Morphology

In this section, we describe how we learn a robust tool morphology using the end-to-end pipeline from Sec. 6.3. We first elaborate on the challenges of learning robust tool morphology for contact-rich scenarios from several perspectives in Sec. 6.4.1, and then formulate the learning problem as continual learning and introduce our proposed method in Sec. 6.4.2.

6.4.1 Challenges for Learning Robust Tool Morphology

We aim to learn a tool morphology that is customized for a specific contact-rich manipulation task, while also being robust and generalizable to task variations, e.g. different start states for the task. Instead of being able to handle just one initial state, as in [194], we propose to learn tool morphology parameter vector $\boldsymbol{\theta}$ that generalizes across a distribution of initial states $\boldsymbol{s}_0 \sim \rho_0(\boldsymbol{s})$. Since evaluating the expectation of the loss would be intractable, we sample a set of N initial states, $S = \{\boldsymbol{s}_0^i | \boldsymbol{s}_0^i \sim \rho_0(\boldsymbol{s}), i = 1 \dots N\}$ and instead optimize the empirical expectation of the loss:

$$\begin{aligned} \boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{\boldsymbol{s}_0^i \in S} \left[\mathcal{L}(\pi; \boldsymbol{s}_0^i, \boldsymbol{\theta})\right], \\ S = \{\boldsymbol{s}_0^i | \boldsymbol{s}_0^i \sim \rho_0(\boldsymbol{s}), i = 1 \dots N\} \end{aligned} \tag{6.3}$$

Directly optimizing this loss with our pipeline in Sec. 6.3 by unrolling episodes in the simulator with different initial states $\boldsymbol{s}_0^i \in S$ comes with several challenges. First of all, compared to tasks that do not have complex dynamics, contact-rich manipulation tasks usually have a highly non-convex optimization landscape, which causes gradient-based optimization algorithms to easily get stuck in local minima. Fig. 6.4 illustrates this by comparing the contact-rich task of *Winding* a rope on a spool to a free-space *Reaching* task. Therefore, for contact-rich tasks, directly optimizing Eqn. (6.3) might converge to local optima leading to suboptimal $\boldsymbol{\theta}$ values that determine the tool morphology. Second, there is a tradeoff when choosing N, the number of samples in our empirical estimate of the

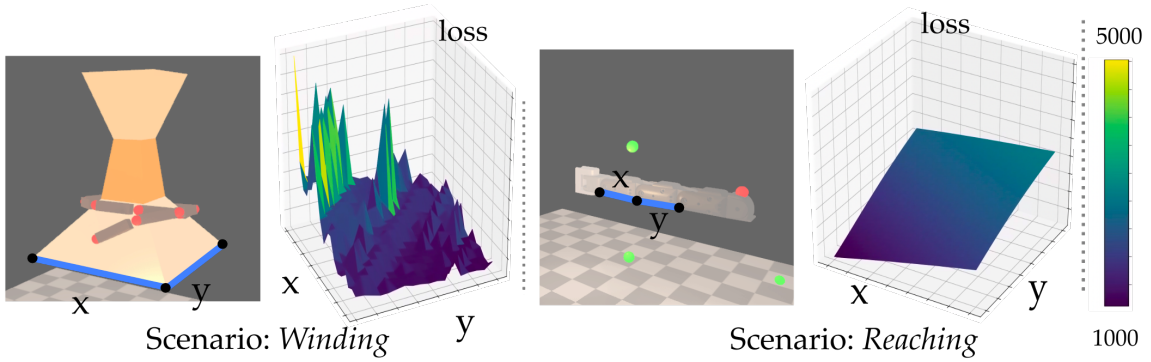


Figure 6.4: We visualize the 2D slices of the loss landscape for the contact-rich *Winding* scenario and the no-contact *Reaching* scenario. For *Winding*, the goal is to prevent the rope from falling. For *Reaching*, the goal is to optimize the arm so that the end-effector can reach the green dots. The variables we optimize over are illustrated in the figure as x,y: for *Winding* these are the lengths of two sides of the tool base; for *Reaching* these are the lengths of two of the robot links. The details for the loss functions are given in the Appendix. It is clear that the optimization landscape of the contact-rich task is significantly more complex than that of the no-contact task.

loss in Eqn. (6.3): Small values of N correspond to an insufficient number of samples, and would fail to capture the distribution $\rho_0(\boldsymbol{s})$. Large values of N are expensive computationally, since evaluating even one gradient step requires unrolling N episodes in the differentiable simulator.

6.4.2 Continual Learning Based Algorithm

Given the challenges of learning a robust tool morphology with our end-to-end pipeline in Sec. 6.3, we draw inspiration from continual learning [104, 154, 125].

Formally, we refer to an MDP defined in Sec. 6.3 with the initial state distribution as a scenario. Within a given scenario, a task variation T^i denotes a restricted MDP with one initial state $\boldsymbol{s}_0^i$ instead of a distribution over initial states. Meaning, for this work we consider task variations that are MDPs with different initial states. In general, task variations could be defined in other ways, e.g. variations in the physical parameters, variations in the goal states, etc. With sampling task variation T^i using $\boldsymbol{s}_0^i \sim \rho_0(\boldsymbol{s})$, optimizing Eqn. (6.3) is equivalent to minimizing the average loss on the sampled task set $T = \{T^1 \dots T^N\}$:

$$\begin{aligned} \boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \textstyle\sum_{T^i \in T} [\mathcal{L}(\pi; \boldsymbol{s}_0^i, \boldsymbol{\theta})] \\ T &= \{T^1 \dots T^N\} \end{aligned} \tag{6.4}$$

Morphology Optimization over a Sequence of Tasks. We can now formulate the problem of learning the morphology for a set of initial states as a continual multi-task learning problem, where the morphology learned for one initial state can inform the optimization of morphology in other task

variations with different initial states. To tackle the issue of this optimization being intractable for large values of N, we draw inspiration from continual learning and solve the problem for the sampled task set T in a sequential manner. Task variations are solved sequentially in batches $B_0, B_1...$, where each batch B_t contains M task variations, $B_t = \{T_t^{(1)} \dots T_t^{(M)} | T_t^{(i)} \in T, i = 1 \dots M\}$. We select $M \ll N$ so that optimizing for each batch becomes tractable. This addresses the second challenge regarding the trade-off for choosing a small number of samples in Sec. 6.4.1, while still recovering the task set $T = \bigcup_{t=1}^{N/M} B_t$ by processing the batches sequentially.

Constructing the Loss with Knowledge Distillation Regularization. We denote the morphology parameter vector we obtain after optimizing over a sequence of batches $B_1...B_{t-1}$ by $\boldsymbol{\theta}_{t-1}$. At timestep t, our method aims to learn $\boldsymbol{\theta}_t$ using an incremental learning strategy by processing the current batch B_t. Intuitively, we aim to minimize the task loss L_t^{task} on the current batch B_t: $L_t^{\text{task}}(\boldsymbol{\theta}) = \frac{1}{M} \sum_{T_t^{(i)} \in B_t} \mathcal{L}(\pi; T_t^{(i)}, \boldsymbol{\theta})$. To avoid forgetting what has already been learned, we construct a regularization term L_t^{distill} for distilling previous knowledge. To this end, we maintain a distillation task set $\mathcal{D}$, which is obtained by randomly sampling M task variations from the previously seen ones, i.e. from $\bigcup\{B_1, ..., B_{t-1}\}$. When optimizing for $\boldsymbol{\theta}_t$ on the current batch, we still want the updated parameters $\boldsymbol{\theta}_t$ to perform similarly to $\boldsymbol{\theta}_{t-1}$ on the previous task variations in distillation set $\mathcal{D}$. Thus, we define the regularization term L^{distill} as the squared error between the simulated trajectories generated when using $\boldsymbol{\theta}_t$ vs $\boldsymbol{\theta}_{t-1}$:

$$L_t^{\text{distill}}(\boldsymbol{\theta}_t) = \frac{1}{M} \sum_{T^{(i)} \in \mathcal{D}} \Big(sim(\pi; T_t^{(i)}, \boldsymbol{\theta}_{t-1}) - sim(\pi; T_t^{(i)}, \boldsymbol{\theta}_t) \Big)^2.$$

We could compute the distillation loss on a part of the trajectory that matters for the task, e.g. height of the rope for the *Winding* scenario (Appendix gives further details).

Finally, we get the overall loss by combining L_t^{task} and L_t^{distill} with a regularization coefficient α: $L^t(\boldsymbol{\theta}_t) = L_t^{\text{task}}(\boldsymbol{\theta}_t) + \alpha L_t^{\text{distill}}(\boldsymbol{\theta}_t)$.

Simplifying Optimization with Dimensionality Reduction. As visualized in Fig. 6.4, the contact-rich tasks considered in this chapter result in a complex optimization landscape that is difficult to optimize over. To alleviate this issue, we draw inspiration from Coordinate Descent [192] and propose to simplify the optimization problem by reducing the dimensionality of the decision variable. For a new batch B_t, we evaluate the gradient with respect to the currently optimal $\boldsymbol{\theta}_{t-1}$ and select $d' < d$ dimensions of parameters $\boldsymbol{\theta}_{t-1}$ with the largest gradient magnitude. Then we only update these $d' < d$ dimensions of parameters $\boldsymbol{\theta}_{t-1}$ to get $\boldsymbol{\theta}_t$. Once a dimension has been optimized in previous batches, we do not select it anymore. When all dimensions have been optimized, we restart (i.e. mark all dimensions as available for optimization) and repeat the process, decaying the learning rate with $\epsilon = e^{-1}$.

6.5 Experiments

To test the performance of our proposed algorithm, we learn the tool morphology for three scenarios in simulation and evaluate the learned tool both in simulation and on a real robot manipulation setting. We first discuss the aspects that are the same for the three scenarios we consider, and then further elaborate on each of the scenarios separately.

Setup. We learn the tool morphology for all three scenarios with the differentiable simulator DiffHand [194]. Across all scenarios, for our algorithm (*Ours*), we set the regularization coefficient $\alpha = 0.1$. The implementation details including hyperparameters are given in Appendix.

Scenarios. We consider three scenarios shown in Fig. 6.5.

1. *Winding*: In this scenario, a rope is wound around a tool, then if the rope does not slip off the tool for a range tool's orientations – this is considered a success. Here we do not consider adapting the robot policy, since the rope can be wound by a simple circular motion. Hence, we compute the loss after the rope is wound and the end of it is left hanging free. With this, the rope would fall, unless it is supported by the tool. Since the simulator we use in this work can only model rigid objects, we use a chain of cuboids to serve as the rope in the task.

2. *Flipping*: In this scenario, we aim to learn a robot arm morphology that can flip a box by 90°. For control, here we first learn a basic flipping policy π (described in Appendix). This scenario is similar to that in previous work [194], except we consider a distribution of initial box poses, while in prior work the box is placed at one fixed position and orientation.

3. *Pushing*: This scenario is based on bimanual scooping used for food acquisition as in [65]. We aim to learn the morphology of a pusher that can push a pea on a table into a scoop. Peas are modelled as spheres. Task variations T^i are instantiated with different initial pea positions. For the trajectory of the pusher we use a forward motion with a zig-zag shape, which can make it challenging for the pusher to prevent the peas from rolling away.

We selected these scenarios because they include (i) various types of objects (rope, box, spherical peas), (ii) objects with different physical properties (boxes with sharp edges in scenario *Flipping*, smooth spheres in scenario *Pushing*), and (iii) varying contact conditions including a rope sliding on a winding tool, robot holding on to a sharp box edge for pivoting, and spheres rolling over a tool's surface.

Baselines. We implemented two baselines for comparison:

1. *Baseline-DiffHand*: we optimize $\boldsymbol{\theta}$ for only one batch of tasks: $\boldsymbol{\theta} = \text{argmin}\, L_0^{\text{task}}(\boldsymbol{\theta})$. This baseline is a direct extension of DiffHand [194] obtained by replacing the single initial state with one batch of initial states and optimizing until convergence, without bringing in the continual learning aspect.

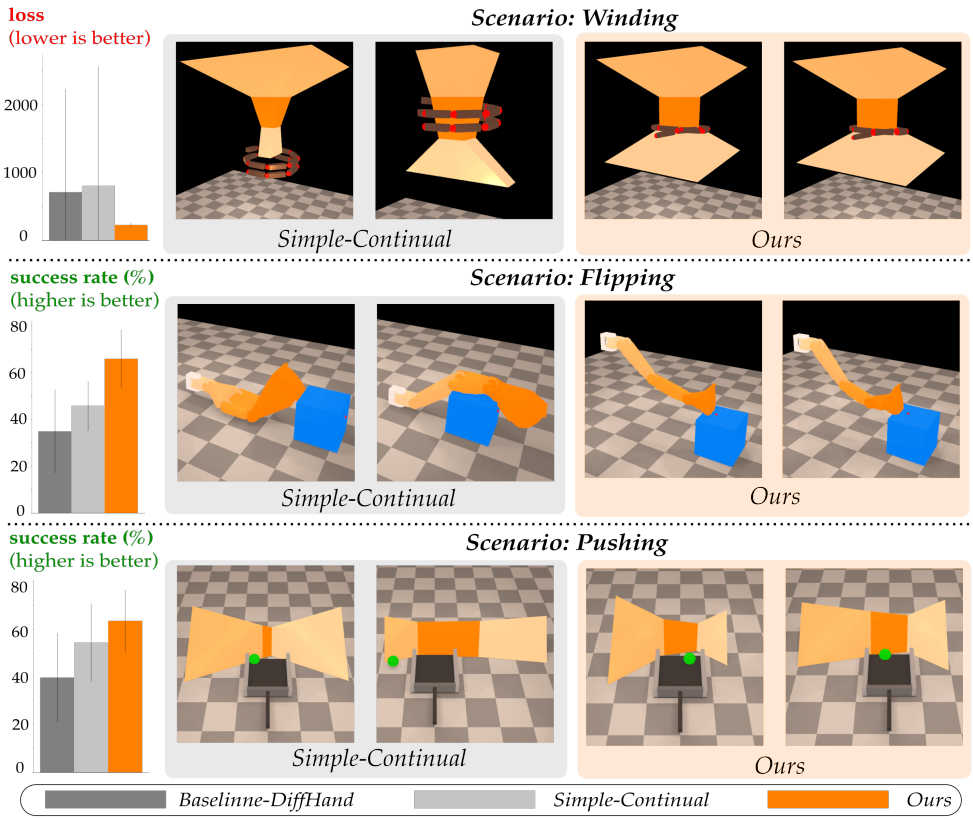


Figure 6.5: Evaluation results on scenarios *Winding* (top row), *Flipping* (middle row) and *Pushing* (bottom row). For each scenario, we implement *Baseline-DiffHand*, *Simple-Continual* and our algorithm (*Ours*) in simulation. For quantitative evaluation, we report the mean and standard deviation over ten runs for the task loss for *Winding* and test accuracy for *Flipping* and *Pushing*. For qualitative evaluation we visualize examples of two optimized morphologies from baseline *Simple-Continual* and our algorithm (*Ours*).

2. *Simple-Continual*: we optimize $\boldsymbol{\theta}$ by minimizing the task loss sequentially for batches $B_1, ..., B_t, ...$ as we would in a continual learning setting, making this a stronger baseline. At batch B_t, we start from $\boldsymbol{\theta}_{t-1}$ and obtain $\boldsymbol{\theta}_t$ using $\boldsymbol{\theta}_t = \text{argmin}\, L_t^{\text{task}}(\boldsymbol{\theta})$. After optimizing for all the batches, we get the final morphology parameter $\boldsymbol{\theta}_{t=N/M}$. With the continual learning setting introduced, the difference between *Ours* and this baseline is that *Ours* uses knowledge distillation regularization in addition to the task loss, and simplifies the optimization with dimensionality reduction.

6.5.1 Results & Analysis in Simulation

As shown in Fig. 6.5, the simulation results demonstrate that the tool morphology learned with our algorithm consistently outperforms both *Baseline-DiffHand* and *Simple-Continual* in terms of a lower test loss and higher test success rate. Across all the scenarios, *Baseline-DiffHand* demonstrates

the worst performance with the largest variance. This is because, compared to optimizing over sequential batches of task variations, approximating the task distribution with only one batch will inherently be noisier. In scenario *Flipping* and *Pushing*, the baseline *Simple-Continual* outperforms the *Baseline-DiffHand*. This also indicates that learning from a sequence of task variations instead of one batch is beneficial. In scenario *Winding*, baseline *Simple-Continual* and *Baseline-DiffHand* show poor performance with a high average loss and large standard deviation, while *Ours* achieves a low loss and small variance (visualized as standard deviation in the bar plots). For some of the runs, *Simple-Continual* and *Baseline-DiffHand* have an exceptionally high loss due to completely failing to hold the rope, so the rope quickly falls down. For both baselines, the performance deteriorates significantly for some batch sequences by converging to a suboptimal morphology parameter. In contrast, our method is not sensitive to how batches are sampled.

We visualize two of the learned tool morphologies for *O*urs and baseline *Simple-Continual* for each scenario in Fig. 6.5. In scenario *Winding*, one of the learned shapes from *Simple-Continual* has one pointed end and one flat end, making the tool not effective for a certain range of orientations. This shows that the shape of the tool learned with baseline *Simple-Continual* sometimes does not perform well across all task variations. In scenario *Flipping*, in contrast to the wide arm tip learned by the *Simple-Continual* baseline, our method learns a finer triangular tip, which can exert enough pressure close to the edge of the box to cause it to flip, thus achieving a higher success rate. In scenario *Pushing*, our method learns a tool morphology with widening ends and a middle segment has a similar width as the scoop. The baseline *Simple-Continual* either learns a narrow middle segment or a wide one. An appropriate width for the middle segment can better facilitate pushing the peas all the way into the scoop. Specifically, a too narrow middle segment might result in a gap between the pusher and the scoop and thus have a hard time pushing the peas all the way into the scoop, while a too wide middle segment can result in the pea rolling away. In summary, across the three scenarios, we observe that the learned morphology from our algorithm demonstrates better performance.

6.5.2 Evaluating Learned Tools in the Real World

We manufactured the tools learned with our method for scenario *Winding* and *Pushing* as shown in Fig. 6.6. The tools were 3D printed on an Ender 3 Pro 3D printer using PLA filament. We tested the functionality of the tool in these two scenarios. In scenario *Winding*, we execute a simple control sequence that winds the rope around the tool. After winding, we rotate the tool in the air with a full circle of 360 degrees around axis x as shown in Fig. 6.6 . A trial is counted as success if the rope successfully stays wrapped around the tool after winding and rotating. We run 5 episodes with the initial shape and the optimized shape. Results show that the optimized shape achieves 100% success rate while the initial shape keeps the rope on the tool only 20% of the time. For scenario *Pushing*, as shown in Fig. 6.6, we use the Franka Panda robot arm on the right to hold the pusher

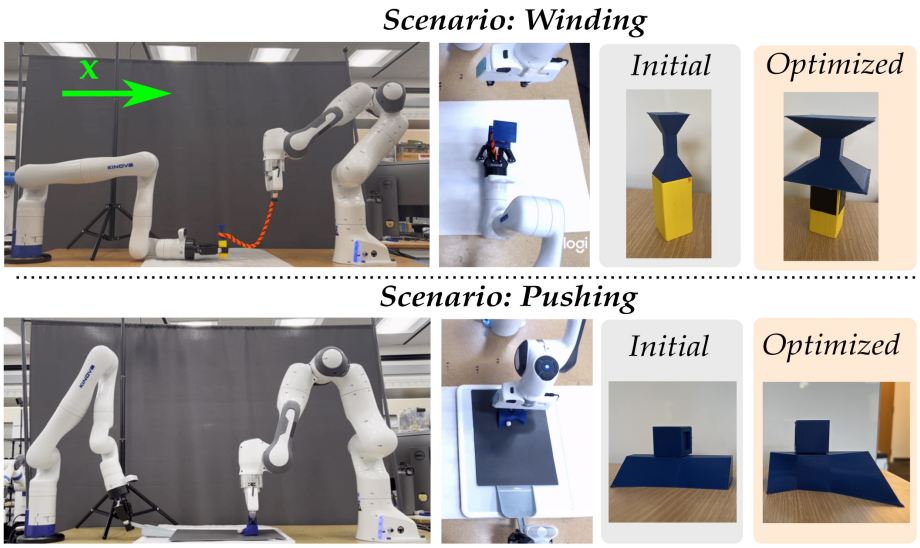


Figure 6.6: Real world experiment set up for *Winding* and *Pushing*. For each scenario, we 3D print the tool with the initial shape and the optimized shape obtained by our approach. For scenario *Winding*, we first wind the rope around the tool and then rotate the tool around the highlighted x axis for 360 degrees to test whether the rope drops. For videos of experiments please see https://sites.google.com/stanford.edu/learning-tool-morphology

and try to push the white sphere to the flat dustpan held by a Kinova Gen3 arm on the left. The arm follows the waypoints of a pre-defined zig-zag trajectory. We also sample 4 initial locations for the pea and run 5 episodes for each initial location, which leads to 20 episodes for one tool. The experiments show that the optimized shape achieves 70% success rate across the 20 trials while the initial shape achieves 15% success rate.

6.6 Conclusion and Discussion

Summary. To summarize, in this work, we approach the problem of custom tool design for robot manipulation in an end-to-end manner by leveraging the advantages of differentiable simulation. To learn versatile tool morphologies, we propose a continual learning approach that enables optimization over task variations, e.g. by varying initial object poses. We show that tools optimized with our method help to achieve an improved task performance compared to baselines in simulation. We also demonstrate that these tools enable successful task completion in reality.

Limitations and Future Work. While we view our work as a step towards enabling tool optimization for contact-rich tasks, we acknowledge several remaining challenges. First, for scenarios with a large sim-to-real gap, seamlessly transferring the tool learned in simulation to the real world is expected to be challenging [199]. Second, optimizing the tool shape and the manipulation policy jointly remains difficult. While DiffHand [194] enables such co-design in principle, we found that

in practice the joint optimization of policy and morphology fails for a large number of task variations. One option is to develop methods that can achieve stable optimization by iteratively adapting morphology and policy during training. We intend to explore this direction in future work.

Chapter 7

Conclusions

7.1 Summary of Contributions

The thesis focuses on enabling robots to adapt to changing circumstances, including robots adapting to humans, robots adapting to robots, and robots adapting to tasks. Instead of relying on pre-defined motion primitives or generic models, we aim at developing adaptive and intelligent robots so that they can operate effectively across a wide range of scenarios where the environment can be dynamic, tasks can be complex, and users can have varying capabilities and preferences.

In the first part of thesis, we begin by investigating how robots can adapt their behavior to better meet the needs and preferences of individual users and groups. In Chapter 2 and Chapter 3, we propose methods for personalizing the control for teleoperating assistive robots, as well as enabling robots to infer the user's intention from sequences of physical corrections and adapt its behavior accordingly. Additionally, We further focus on how robots can better adapt to human groups in Chapter 4 and develop a framework that first reasons over the group structure and then leverage it for planning.

The second part of the thesis extends the investigation from human users to robot agents. We tackle the problem of how decentralized robot teams can adapt to each other by observing only the actions of other agents in Chapter 5. We propose a framework for establishing communication protocols among robots that enables them to adapt their behavior to the behavior of other agents in the team.

In the final part of our dissertation, we explore how robots can adapt to specific tasks by developing customized tools. We proposed a novel tool design approach for improving the performance of robots in manipulation tasks in Chapter 6. By simplifying the problem with dimensionality reduction and reinterpreting continual learning as a robust optimization framework, we effectively optimize the tool morphology for specific task objectives.

Overall, our research has contributed to advancing the field of robotics by proposing new methods

and frameworks for enabling robots to adapt. Our contributions have the potential to enhance the efficiency and effectiveness of robots across various applications, ranging from healthcare assistive robots to disaster response and manufacturing. The thesis presents new avenues for future research in the field and provides valuable insights for the development of more advanced robot learning techniques, facilitating the creation of more intelligent robot behaviors.

7.2 Limitations and Future Work

While this thesis achieves preliminary success in enabling robots for adaptation to humans, robot agents and tasks under various scenarios, there still remain several limitations and challenges to address.

Insufficient Modeling of Human Adaptation. Across most of our works in Part I, we assume the preferences that we aim to capture are stationary. However, humans are highly adaptable, and their non-stationary nature poses challenges when scaling up and deploying robots in real-world scenarios. Assuming that human preferences are fixed and relying solely on the robot to adapt to human users may not be accurate. It is crucial to understand and model humans' adaptive behavior to bridge the gap between robots and humans.

Insufficient Modeling of Non-Stationary Dynamics. In Parts II and III, we focus on enabling robots to adapt to various environments and tasks. However, we make the assumption that the environment and task remain stationary and do not change during the robot's execution. In reality, the dynamics of the environment can change over time, which poses new challenges for robot adaptation. For instance, in a warehouse, the layout of the facility may change as new items arrive or old ones are removed, making it challenging to coordinate and communicate effectively. Similarly, in an agricultural setting, crops grow and change shape over time, which can pose difficulties in selecting or designing appropriate tools and grippers capable of handling novel shapes.

Data Efficiency. With the increasing popularity of deep learning methods in robotics, efficient data utilization becomes essential. Training neural network models requires a large amount of data, which can be expensive to collect when deploying the robot. This cost is even higher or sometimes infeasible when humans are involved, such as with assistive robots for individuals with disabilities or physical limitations. In Chapter 2, we proposed one method to tackle this problem by incorporating intuitive priors. However, more complex and dynamic scenarios require additional investigation. One promising direction is to extract as many priors as possible from large datasets in an unsupervised way. This might have the potential to reduce the need for extensive data collection, making it more feasible and cost-effective to develop and deploy deep learning models in real-world robotic applications.

Overall, the field of robotics presents many challenges and opportunities, and its interdisciplinary nature offers a wide range of avenues for research. We look forward to seeing more innovative and

promising works that push the boundaries towards the ultimate goal of enabling intelligent robots to interact with the world with human-level intelligence.

Appendix A

Chapter 5 Appendix

A.1 Derivations for Eqn. (5.1)

$$\begin{aligned}
\pi_1(a_1^t \mid s_1^{0:t}, a^{0:t-1}, a_2^t) &= \sum_{s_2^{0:t}} P(a_1^t, s_2^{0:t} \mid s_1^{0:t}, a^{0:t-1}, a_2^t) \\
&= \frac{\sum_{s_2^{0:t}} P(a_1^t, a_2^t, s_2^{0:t} \mid s_1^{0:t}, a^{0:t-1})}{P(a_2^t \mid s_1^{0:t}, a^{0:t-1})} \\
&\propto \sum_{s_2^{0:t}} P(a_1^t, a_2^t, s_2^{0:t} \mid s_1^{0:t}, a^{0:t-1}) \\
&= \sum_{s_2^{0:t}} P(a_2^t \mid a_1^t, s_2^{0:t}, s_1^{0:t}, a^{0:t-1}) \cdot P(a_1^t \mid s_2^{0:t}, s_1^{0:t}, a^{0:t-1}) \\
&\qquad \cdot P(s_2^{0:t} \mid s_1^{0:t}, a^{0:t-1}) \\
&= \sum_{s_2^{0:t}} \pi_2(a_2^t \mid s_2^{0:t}, a^{0:t-1}, a_1^t) \cdot \pi_1^*(a_1^t \mid s_1^t, s_2^t) \cdot P(s_2^{0:t} \mid s_1^{0:t}, a^{0:t-1})
\end{aligned} \tag{A.1}$$

Thus we have

$$\pi_1(a_1^t \mid s_1^{0:t}, a^{0:t-1}, a_2^t) \propto \sum_{s_2^{0:t}} \pi_1^*(a_1^t \mid s_1^t, s_2^t) \cdot \pi_2(a_2^t \mid s_2^{0:t}, a^{0:t-1}, a_1^t) \cdot P(s_2^{0:t} \mid s_1^{0:t}, a^{0:t-1}) \tag{A.2}$$

Similarly, we could derive the symmetric formulation for π_2.

$$\pi_2(a_2^t \mid s_2^{0:t}, a^{0:t-1}, a_1^t) \propto \sum_{s_1^{0:t}} \pi_2^*(a_2^t \mid s_1^t, s_2^t) \cdot \pi_1(a_1^t \mid s_1^{0:t}, a^{0:t-1}, a_2^t) \cdot P(s_1^{0:t} \mid s_2^{0:t}, a^{0:t-1}) \tag{A.3}$$

A.2 Derivations for Eqn. (5.7)

For the stochastic centralized policy, we have:

$$\begin{aligned}\pi_1^*(a_1 \mid s_1, s_2) &= \mathcal{N}(K_{11}s_1 + K_{12}s_2, w_1^2)\\ \pi_2^*(a_2 \mid s_1, s_2) &= \mathcal{N}(K_{21}s_1 + K_{22}s_2, w_2^2)\end{aligned} \tag{A.4}$$

The states s_1 and s_2 are constant and independent, and sampled from Gaussian priors: $s_1 \sim \mathcal{N}(\mu_1, \sigma_{s_1}^2)$ and $s_2 \sim \mathcal{N}(\mu_2, \sigma_{s_2}^2)$. Without loss of generality, let agent 1 be speaker and agent 2 is listener. The speaker and listener policies are:

$$\pi_1(a_1 \mid s_1) = \mathcal{N}(K_{11}s_1 + K_{12}\mu_2, \sigma_1^2) \tag{A.5}$$

$$\pi_2(a_2 \mid s_2, a_1) = \mathcal{N}(K_{21}K_{11}^{-1}(a_1 - K_{12}\mu_2) + K_{22}s_2, \sigma_2^2) \tag{A.6}$$

Now we can do the KL divergence to solve for the parameters $\theta = \{\sigma_1, \sigma_2\}$

$$\begin{aligned}\min_\theta KL &= \min_\theta \Bigg[\log\frac{w_1}{\sigma_1} + \frac{\sigma_1^2 + (K_{11}s_1 + K_{12}s_2 - K_{11}s_1 - K_{12}\mu_2)^2}{2w_1^2} + \log\frac{w_2}{\sigma_2} + \\ &\qquad \frac{\sigma_2^2 + (K_{21}s_1 + K_{22}s_2 - K_{21}K_{11}^{-1}(a_1 - K_{12}\mu_2) - K_{22}s_2)^2}{2w_2^2}\Bigg] \\ &= \min_\theta \left[\log\frac{w_1 w_2}{\sigma_1\sigma_2} + \frac{\sigma_1^2}{2w_1^2} + \frac{K_{12}^2(s_2 - \mu_2)^2}{2w_1^2} + \frac{\sigma_2^2 + K_{21}^2(s_1 - K_{11}^{-1}(a_1 - K_{12}\mu_2))^2}{2w_2^2}\right]\end{aligned} \tag{A.7}$$

Taking expectation over Eqn. (A.7), we reach:

$$\min_\theta \mathbb{E}[KL] = \min_\theta \left[\frac{K_{12}^2\sigma_{s_2}^2}{2w_1^2} + \log\frac{w_1 w_2}{\sigma_1\sigma_2} + \frac{\sigma_1^2}{2w_1^2} + \frac{\sigma_2^2 + K_{21}^2\sigma_1^2/K_{11}^2}{2w_2^2}\right] \tag{A.8}$$

The optimal choices of σ_1 and σ_2 are:

$$\sigma_1 = \frac{K_{11}w_1 w_2}{\sqrt{K_{11}^2 w_2^2 + K_{21}^2 w_1^2}}, \quad \sigma_2 = w_2 \tag{A.9}$$

So far, we already derive Eqn. (5.7). Further, the minimum values for the KL divergence are given by:

$$\frac{K_{12}^2\sigma_{s_2}^2}{2w_1^2} + \log\frac{\sqrt{K_{11}^2 w_2^2 + K_{21}w_1^2}}{K_{11}w_2} + 1 \tag{A.10}$$

A.3 Simulation Environment

In our simulation environment, we make the simplification of making agent i velocity v_i as input action, instead of force. And we pose no hard constraint of speed continuity. Formally, Eqn. (A.11)

gives the system dynamics. The team center's translation velocity v_c^{trans} is the average of two agents' input velocity v_1 and v_2, the angular velocity for rotation is computed by first projecting the velocity difference to the stick's perpendicular direction (unit vector u_i^r points from stick center to agent i). And then dividing by r, the length from stick center to agent (half length of the whole stick)

$$v_c^{trans} = \frac{1}{2}(v_1 + v_2), \quad \omega = \frac{1}{r} u_i^r \times (v_i - v_c^{trans}) \tag{A.11}$$

A.4 Planning

We adopt the well-known potential field model [18] for planning. Agents at location q plan their path under the influence of an artificial potential field $U(q)$, which is constructed to reflect the environment. There are two types of potential field sources. We denote the set of attractions $k \in \mathcal{A}$ and the set of repulsive obstacles $j \in \mathcal{R}$. The overall potential field is the sum of all attractive and repulsive potential field sources as in Eqn. (A.12). The action (input velocity v_i) an agent i at location q_i takes lies in the direction of the potential field gradient as in Eqn. (A.13). w_v is a predefined hyperparameter controlling velocity scale. In this way, the potential field planner could naturally combine agent's knowledge of goal g as an attraction and its nearby obstacles as repulsive sources.

$$U(q) = \sum_{k \in \mathcal{A}} U_{att}^k(q) + \sum_{j \in \mathcal{R}} U_{rep}^j(q) \tag{A.12}$$

$$v_i(q_i) = -w_v \nabla U(q_i) = -w_v \Big(\sum_{k \in \mathcal{A}_i} \nabla U_{att}^k(q_i) + \sum_{j \in \mathcal{R}_i} \nabla U_{rep}^j(q_i) \Big) \tag{A.13}$$

In our implementation, we directly give the gradient for attractive potential field $\nabla U_{att}^k(q)$ and repulsive potential field $\nabla U_{rep}^j(q)$ as in Eqn. (A.14) and Eqn. (A.15). w_{att} and w_{rep} are the hyperparameters controlling the relative scale of the attractive and repulsive potential field. q_k is the location for attraction k. q_j is the location for repulsive source j. $\rho_j(q)$ is the minimum distance of agent at location q to obstacle j. ρ_0 is a hyperparameter controlling the effective range of repulsive potential field. The repulsive potential field is 0 outside of the range ρ_0.

$$\nabla U_{att}^k(q) = w_{att} * \frac{q - q_k}{\|q - q_k\|} \tag{A.14}$$

$$\nabla U_{rep}^j(q) = \begin{cases} w_{rep} * \left(\frac{1}{\rho_j(q)} - \frac{1}{\rho_0}\right) \left(\frac{1}{\rho_j(q)}\right) \frac{q - q_j}{\|q - q_j\|} & \text{if } \rho_j(q) \le \rho_0 \\ 0 & \text{if } \rho_j(q) > \rho_0 \end{cases} \tag{A.15}$$

Table A.1: Average centralized game length over failure cases when both robots knew the geometry of the obstacles *a priori*.

	Roles (Dynamic)			**Roles (Static)**	
Obstacles	$T = 1$	$T = 4$	$T = 16$	Speaker-Listener	Speaker-Speaker
n = 2	**164.00**	152.93	146.53	145.96	143.13
n = 4	**181.11**	154.98	149.99	148.04	146.75
n = 8	**167.61**	160.47	156.89	158.13	154.65

A.5 Inference

As discussed in Sec. 5.3 and Sec. 5.4, the listener would perform inference over the speaker i's action $v_i(q_i)$. In our context, since the goal attraction is known and shared, what the listener is trying to infer are the speaker's repulsive obstacles $j \in \mathcal{R}_i$. In our implementation, the listener approximates the speaker i's repulsive potential field with one inferred obstacle $\bar{j}_i$ at location $\bar{q}_i$. More formally, this is equivalent to solving Eqn. (A.16)

$$v_i(q_i) = -w_v(\sum_{k \in \mathcal{A}_i} \nabla U^k_{att}(q_i) + \nabla U^{\bar{j}_i}_{rep}(\bar{q}_i)) \tag{A.16}$$

Combined with Eqn. (A.14) and Eqn. (A.15), numerical solution for obstacle $\bar{j}_i$'s location $q_{\bar{j}_i}$ is obtained by bisection iteration.

A.6 Supplementary Experiment Results

We show our supplementary experiment results in simulation for Sec. 5.5.2 and Sec. 5.5.4 here respectively.

A.6.1 Rapidly Changing Roles Outperforms Static Roles

In this setting, in addition to success rate λ for Table 5.1 in Sec. 5.5.2, we also measure an additional metric, centralized policy's average game length l for dynamic role and static role teams. The results are summarized in Table A.1. The dynamic role team with shortest switch interval has the highest game length. It indicates that its failure cases are the most difficult and thus suggests that it could deal with more complex environment.

A.6.2 Roles with Noisy Action Observations Match Noisy Messages

In Sec. 5.5.4, we explored the noisy action observation case with coefficient of variance 0.1 as in Table 5.2. We also vary the noise level by setting different coefficient of variation $\frac{\sigma}{\mu} \in \{0.001, 0.01, 0.1\}$. The σ is the Gaussian standard deviation, and the μ is the mean value of action observation, i.e. the

groundtruth action value. For explicit communication, we add the same scale communication noise (coefficient of variation) for sending obstacle location. We summarize our full results in Table A.2. As noise level increases, the success rate gradually drops and the gap between dynamic role and explicit communication becomes greater. This is because the error gets amplified by the nonlinearity during inference process. Nevertheless, across all cases, the dynamic role methods with short switch interval ($T = 1, 4$) consistently performs better than fixed speaker-listener team, followed by dynamic role with long switch interval ($T = 16$), and then the speaker-speaker team. Overall, the dynamic role strategy could decently deal with disturbance and the performance is still on par with explicit communication.

Table A.2: Success rate λ over 1000 games when communication is noisy. $\frac{\sigma}{\mu}$ is coefficient of variation that controls the noise level for both action observation and explicit communication. Here S-L represents the static Speaker-Listener team and S-S for static Speaker-Speaker team.

		Explicit	**Roles (Dynamic)**			**Roles (Static)**	
Obstacles	noise level $\frac{\sigma}{\mu}$	$T = 0$	$T = 1$	$T = 4$	$T = 16$	S-L	S-S
n=2	0.001	0.997	0.989	0.89	0.818	0.889	0.758
	0.01	0.996	0.980	0.891	0.814	0.889	
	0.1	0.925	0.884	0.818	0.787	0.829	
n=4	0.001	0.984	0.943	0.778	0.653	0.731	0.558
	0.01	0.981	0.927	0.768	0.651	0.731	
	0.1	0.833	0.747	0.669	0.606	0.655	
n=8	0.001	0.904	0.738	0.531	0.388	0.495	0.296
	0.01	0.900	0.716	0.504	0.388	0.491	
	0.1	0.553	0.512	0.416	0.357	0.398	

Appendix B

Chapter 6 Appendix

Here, we provide additional details for the *Winding*, *Flipping*, and *Pushing* scenarios discussed in Section V and the *Reaching* scenario visualized in Fig. 4. Recall that our objective is to optimize a vector of parameters $\boldsymbol{\theta}$, which encodes the morphology of the tool in each scenario. Across all scenarios in Section V, for our algorithm (*Ours*), we set the regularization coefficient $\alpha = 0.1$. We summarize other hyperparameters for each scenario in Table B.1.

Scenario	N	M	d	d'
Winding	200	5	8	2
Flipping	100	5	9	2
Pushing	100	5	7	2

Table B.1: Hyperparameters.

B.1 Winding

Loss function: In this scenario, we use 15 linked cuboids to approximate a rope. At each simulation time step τ, we denote the position of the rope's center of mass as $(x_\tau(\boldsymbol{\theta}), y_\tau(\boldsymbol{\theta}), h_\tau(\boldsymbol{\theta}))$. The task loss is the height of the rope's center of mass $h_\tau(\cdot)$ squared, summed over all time steps. This is computed by letting the rope fall under gravity for H simulation steps:

$$L^{\text{task}}(\boldsymbol{\theta}) = \sum_{\tau=1}^{H} (h_\tau(\boldsymbol{\theta}) - h_0)^2.$$

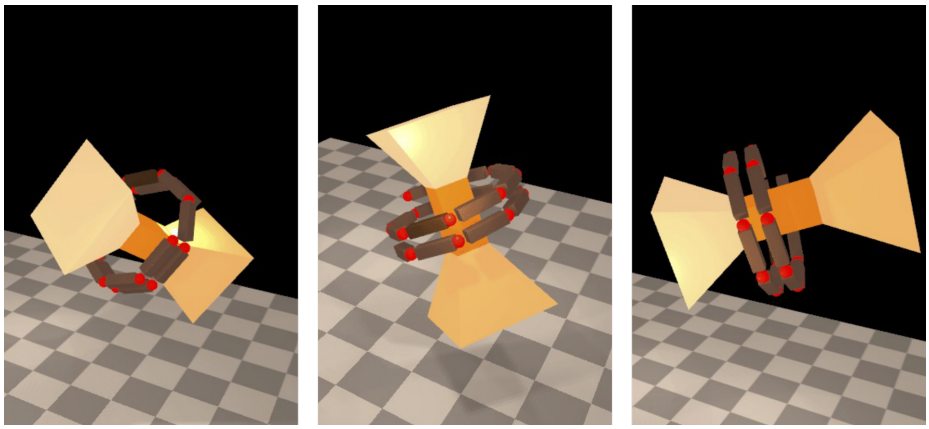


Figure B.1: We visualize three example initial states for the scenario *Winding*. Here, the initial orientation for the rope and the tool are sampled from a uniform distribution on the space of rotations in 3D.

Here, h_0 is the height of the rope's center of mass at time step $\tau = 0$ at the start of the simulation. Suppose the morphology optimized so far is expressed by $\boldsymbol{\theta}_{t-1}$. Our distillation loss for $\boldsymbol{\theta}_t$ to prevent our model from forgetting what has already been learned is defined as:

$$L^{\text{distill}}(\boldsymbol{\theta}_\tau) = \frac{1}{H}\sum_{\tau=1}^{H}(h_\tau(\boldsymbol{\theta}_k) - h_\tau(\boldsymbol{\theta}_{t-1}))^2.$$

Policy: In scenario *Winding*, we initialize the rope to be placed around the tool. The task variations correspond to different initial orientation of the tool and rope as shown in Fig. B.1. We let the rope drop and the rope will not fall if the tool can effectively support the rope.

B.2 Flipping

Loss function: This scenario is adopted from [194] with the difference that we randomize the initial state of the box (see Fig. B.2). We use the same task loss as in [194] for L^{task}:

$$L_{\text{task}}(\boldsymbol{\theta}) = c_{\text{flip}} \left\|\phi_H(\boldsymbol{\theta}) - \frac{\pi}{2}\right\|^2 + \sum_{\tau=1}^{H}\left(c_u \left\|u_\tau(\boldsymbol{\theta})\right\|^2 + c_{\text{touch}} \left\|p_\tau(\boldsymbol{\theta}) - p_{\text{box}}\right\|^2\right)$$

$$\text{with } c_u = 5, c_{\text{touch}} = \begin{cases} 1 & t < H/2 \\ 0 & t \geq H/2 \end{cases}, c_{\text{flip}} = 50.$$

Here, for simulation step τ, $u_\tau(\boldsymbol{\theta}) \in [-1, 1]$ is the robot action, $p_\tau(\boldsymbol{\theta})$ is the finger tip position, and $\phi_\tau(\boldsymbol{\theta})$ is the rotation angle of the box.

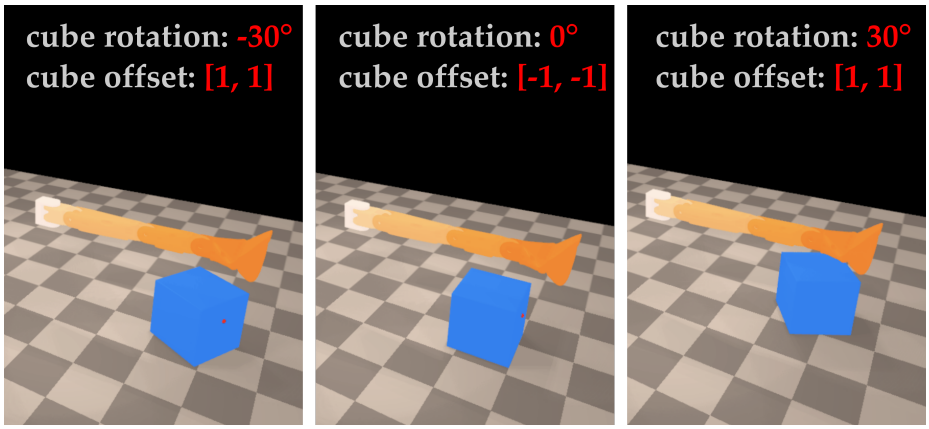


Figure B.2: We visualize three example initial states for the scenario *Flipping*. Here, the initial position is uniformly sampled from a square area in [-2, -2] to [2, 2], and the orientation for the box is sampled from a uniform distribution between $[-90°, 90°]$ around the vertical axis.

For our algorithm, we define the loss $L^{\text{distill}}(\boldsymbol{\theta}_t)$ as:

$$L^{\text{distill}}(\boldsymbol{\theta}_t) = \frac{1}{H}\sum_{\tau=1}^{H}\Big[(u_\tau(\boldsymbol{\theta}_t) - u_\tau(\boldsymbol{\theta}_{t-1}))^2 + (p_\tau(\boldsymbol{\theta}_t) - p_\tau(\boldsymbol{\theta}_{t-1}))^2 + (\phi_\tau(\boldsymbol{\theta}_t) - \phi_\tau(\boldsymbol{\theta}_{t-1}))^2\Big]. \quad \text{(B.1)}$$

Policy: For this task, we train a neural network (NN) to obtain a basic closed-loop policy π. For training data, we first randomly sample a range of starting cube orientations. Then for each pose, we separately run DiffHand [194] to learn a basic morphology and an open-loop policy. In practice, only a small number of such open-loop policies succeed. Hence, we did not attempt to jointly learn a policy and morphology over a distribution of starting poses, since that problem would be even more difficult. Nonetheless, we can use the trajectories that succeed at flipping the cube to construct a training dataset. With that, we train a basic NN policy that learns to imitate successful trajectories. This policy is 'basic' for two reasons. First, only a small number of open-loop policies are successful (as mentioned above), so the training data contains example trajectories for only a small portion of the space. Second, the examples are from policies trained jointly with morphologies. This means the open-loop policies are unlikely to succeed when used with another morphology, unless we can solve the problem of finding a versatile morphology that works for a range of cube poses. The latter is exactly the problem we address in this work.

B.3 Pushing

Loss function: In this scenario, our goal is to push a pea onto a scoop that is placed on the table. We denote the half width of the scoop to be x_{scoop}. Fig. B.3 visualizes the scoop placed such that its opening is located at $\{(x, y)|y = y_{\text{scoop}}, -x_{\text{scoop}} < x < x_{\text{scoop}}\}$. We denote the x coordinate of

the pea's position as $\tilde{x}(\boldsymbol{\theta})$. In the figure the y coordinate of the pea's position is $y = y_{\text{scoop}}$, same as the y coordinate of the tip of the scoop. We construct the task loss by giving a penalty when the pea is outside of the opening of the scoop, i.e. $\tilde{x}_i(\boldsymbol{\theta}) \notin (-x_{\text{scoop}}, x_{\text{scoop}})$:

$$L^{\text{task}}(\boldsymbol{\theta}) = \left\{ \begin{array}{ll} 0 & ||\tilde{x}(\boldsymbol{\theta})|| < x_{\text{scoop}} \\ (||\tilde{x}(\boldsymbol{\theta})|| - x_{\text{scoop}})^2 & ||\tilde{x}(\boldsymbol{\theta})|| \geq x_{\text{scoop}} \end{array} \right\}$$

We define the distillation loss as:

$$L^{\text{distill}}(\boldsymbol{\theta}_t) = \sum_{\tau=1}^{H} (x_\tau(\boldsymbol{\theta}_t) - x_\tau(\boldsymbol{\theta}_{t-1}))^2 + (y_\tau(\boldsymbol{\theta}_t) - y_\tau(\boldsymbol{\theta}_{t-1}))^2.$$

Policy: For this scenario, we use a predefined zig-zag trajectory for the pusher. This motion makes it more challenging for the pusher to prevent the peas from rolling away.

B.4 Reaching

We take this scenario from [194] for visualizing the landscape in Fig. 4. Here, a finger with multiple joints is assumed to be mounted on a wall. The finger aims to sequentially reach the target points represented by the green dots in Fig. 4. For simulation step τ, $u_\tau(\boldsymbol{\theta}) \in [-1, 1]$ is the action, $p_\tau(\boldsymbol{\theta})$ is the finger tip position and $\hat{p}_\tau$ is the target point. The task loss is computed by:

$$\mathcal{L}^{\text{task}}(\boldsymbol{\theta}) = \sum_{\tau=1}^{H} c_u \left\| u_t(\boldsymbol{\theta}) \right\|^2 + c_p \left\| p_t(\boldsymbol{\theta}) - \hat{p}_t \right\| \text{ with } c_u = 0.1, c_p = 10.$$

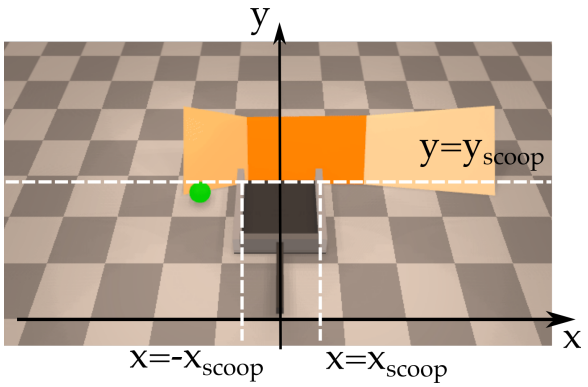


Figure B.3: Visualization of *Pushing* annotated with the scoop position. The tip of the opening of the grey scoop is at $\{(x, y)|y = y_{\text{scoop}}, -x_{\text{scoop}} < x < x_{\text{scoop}}\}$. Here, the green pea falls outside of the scoop, and thus will incur non-zero loss according to Eqn. (B.3).

Bibliography

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

[2] David A Abbink, Tom Carlson, Mark Mulder, Joost CF de Winter, Farzad Aminravan, Tricia L Gibo, and Erwin R Boer. A topology of shared control systems – finding common ground in diversity. *IEEE Transactions on Human-Machine Systems*, 48(5):509–525, 2018.

[3] Ali-Akbar Agha-Mohammadi, Suman Chakravorty, and Nancy M Amato. Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304, 2014.

[4] Baris Akgun, Maya Cakmak, Karl Jiang, and Andrea L Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355, 2012.

[5] Stefano Vittorino Albrecht. Utilising policy types for effective ad hoc coordination in multiagent systems. 2015.

[6] Christopher Amato, Girish Chowdhary, Alborz Geramifard, N Kemal Üre, and Mykel J Kochenderfer. Decentralized control of partially observable Markov decision processes. In *IEEE Conference on Decision and Control (CDC)*, pages 2398–2405, 2013.

[7] Christopher Amato, George Konidaris, Gabriel Cruz, Christopher A Maynor, Jonathan P How, and Leslie P Kaelbling. Planning for decentralized control of multiple robots under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1241–1248, 2015.

[8] Rika Antonova, Jingyun Yang, Krishna Murthy Jatavallabhula, and Jeannette Bohg. Rethinking optimization with differentiable simulation from a global perspective. *arXiv preprint arXiv:2207.00167*, 2022.

[9] Jumpei Arata. Intuitive control in robotic manipulation. In *Human Inspired Dexterity in Robotic Manipulation*, pages 53–60. Elsevier, 2018.

[10] Brenna D Argall. Autonomy in rehabilitation robotics: An intersection. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:441–463, 2018.

[11] 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.

[12] Nora Ayanian and Vijay Kumar. Decentralized feedback controllers for multiagent teams in environments with obstacles. *IEEE Transactions on Robotics*, 26(5):878–887, 2010.

[13] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[14] Haoyu Bai, Shaojun Cai, Nan Ye, David Hsu, and Wee Sun Lee. Intention-aware online pomdp planning for autonomous driving in a crowd. In *2015 ieee international conference on robotics and automation (icra)*, pages 454–460. IEEE, 2015.

[15] Andrea Bajcsy, Dylan P Losey, Marcia K O'Malley, and Anca D Dragan. Learning from physical human corrections, one feature at a time. In *ACM/IEEE International Conference on Human-Robot Interaction*, pages 141–149, 2018.

[16] 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 (CoRL)*, pages 217–226, 2017.

[17] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009.

[18] Jerome Barraquand, Bruno Langlois, and J-C Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2):224–241, 1992.

[19] Samuel Barrett. *Making friends on the fly: advances in ad hoc teamwork*, volume 603. Springer, 2015.

[20] Samuel Barrett and Peter Stone. Ad hoc teamwork modeled with multi-armed bandits: An extension to discounted infinite rewards. In *Proceedings of 2011 AAMAS Workshop on Adaptive and Learning Agents*, pages 9–14, 2011.

[21] Chandrayee Basu, Erdem Biyik, Zhixun He, Mukesh Singhal, and Dorsa Sadigh. Active learning of reward dynamics from hierarchical queries. In *IROS*, pages 120–127, 2019.

[22] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

[23] Aaron Bestick, Ruzena Bajcsy, and Anca D Dragan. Implicitly assisting humans to choose good grasps in robot to human handovers. In *International Symposium on Experimental Robotics*, pages 341–354. Springer, 2016.

[24] Erdem Biyik, Malayandi Palan, Nicholas C. Landolfi, Dylan P. Losey, and Dorsa Sadigh. Asking easy questions: A user-friendly approach to active reward learning. In *Proceedings of the 3rd Conference on Robot Learning (CoRL)*, October 2019.

[25] Michael Bloem and Nicholas Bambos. Infinite time horizon maximum causal entropy inverse reinforcement learning. In *53rd IEEE Conference on Decision and Control*, pages 4911–4916. IEEE, 2014.

[26] RG Boboc, H Moga, and D Talaba. A review of current applications in teleoperation of mobile robots. *Bulletin of the Transilvania University of Brasov. Engineering Sciences. Series I*, 5(2):9, 2012.

[27] Andreea Bobu, Andrea Bajcsy, Jaime F Fisac, Sampada Deglurkar, and Anca D Dragan. Quantifying hypothesis space misspecification in learning from human–robot demonstrations and physical corrections. *IEEE Transactions on Robotics*, 36(3):835–854, 2020.

[28] Michael Bowling and Peter McCracken. Coordination and adaptation in impromptu teams. In *AAAI*, volume 5, pages 53–58, 2005.

[29] Sean L Bowman, Cameron Nowzari, and George J Pappas. Coordination of multi-agent systems via asynchronous cloud communication. In *IEEE Conference on Decision and Control (CDC)*, pages 2215–2220. IEEE, 2016.

[30] Daniel S Brown, Sean C Kerman, and Michael A Goodrich. Human-swarm interactions based on managing attractors. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, pages 90–97, 2014.

[31] Frank Broz, Illah Nourbakhsh, and Reid Simmons. Designing pomdp models of socially situated tasks. In *RO-MAN, 2011 IEEE*, pages 39–46. IEEE, 2011.

[32] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[33] Greg C Causey and Roger D Quinn. Gripper design guidelines for modular manufacturing. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, volume 2, pages 1453–1458. IEEE, 1998.

[34] Hande Çelikkanat and Erol Şahin. Steering self-organized robot flocks through externally guided individuals. *Neural Computing and Applications*, 19(6):849–865, 2010.

[35] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[36] Sonia Chernova and Andrea L Thomaz. Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(3):1–121, 2014.

[37] Yoeng-Jin Chu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

[38] Matei T Ciocarlie and Peter K Allen. Hand posture subspaces for dexterous robotic grasping. *The International Journal of Robotics Research*, 28(7):851–867, 2009.

[39] Iain D Couzin, Jens Krause, Nigel R Franks, and Simon A Levin. Effective leadership and decision-making in animal groups on the move. *Nature*, 433(7025):513–516, 2005.

[40] Emiliano Cristiani and Benedetto Piccoli. A unifying model for the structure of animal groups on the move. *arXiv preprint arXiv:0903.4056*, 2009.

[41] Preston Culbertson and Mac Schwager. Decentralized adaptive control for collaborative manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 278–285, 2018.

[42] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. *Advances in neural information processing systems*, 31, 2018.

[43] Agostino De Santis, Bruno Siciliano, Alessandro De Luca, and Antonio Bicchi. An atlas of physical human–robot interaction. *Mechanism and Machine Theory*, 43(3):253–270, 2008.

[44] Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, page 6, 2019.

[45] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[46] Roel Dobbe, David Fridovich-Keil, and Claire Tomlin. Fully decentralized policies for multi-agent systems: An information theoretic approach. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2941–2950, 2017.

[47] Anca D Dragan Dorsa Sadigh, Shankar Sastry, and Sanjit A Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems (RSS)*, 2017.

[48] Finale Doshi and Nicholas Roy. Efficient model learning for dialog management. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 65–72. ACM, 2007.

[49] Anca D Dragan, Katharina Muelling, J Andrew Bagnell, and Siddhartha S Srinivasa. Movement primitives via optimization. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2339–2346. IEEE, 2015.

[50] Anca D Dragan and Siddhartha S Srinivasa. *Formalizing assistive teleoperation*. MIT Press, July, 2012.

[51] Anca D Dragan and Siddhartha S Srinivasa. A policy-blending formalism for shared control. *The International Journal of Robotics Research*, 32(7):790–805, 2013.

[52] Vincent Duchaine and Clément Gosselin. Safe, stable and intuitive control for physical human-robot interaction. In *2009 IEEE International Conference on Robotics and Automation*, pages 3383–3388. IEEE, 2009.

[53] Jack Edmonds. Optimum branchings. *Mathematics and the Decision Sciences, Part*, 1(335-345):25, 1968.

[54] 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.

[55] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2137–2145, 2016.

[56] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, 2018.

[57] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[58] Kathryn Long Genter et al. *Fly with me: algorithms and methods for influencing a flock*. PhD thesis, 2017.

[59] Katie Genter, Noa Agmon, and Peter Stone. Ad hoc teamwork for leading a flock. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 531–538. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[60] Katie Long Genter, Noa Agmon, and Peter Stone. Role-based ad hoc teamwork. In *Plan, Activity, and Intent Recognition*, pages 1782–1783. Citeseer, 2011.

[61] Irene Giardina. Collective behavior in animal groups: theoretical models and empirical studies. *HFSP journal*, 2(4):205–219, 2008.

[62] Kathleen R Gibson, Kathleen Rita Gibson, and Tim Ingold. *Tools, language and cognition in human evolution*. Cambridge University Press, 1994.

[63] Matthew C Gombolay, Cindy Huang, and Julie A Shah. Coordination of human-robot teaming with human task preferences. In *AAAI Fall Symposium Series on AI-HRI*, volume 11, page 2015, 2015.

[64] Philippe Gorce and Jean Guy Fontaine. Design methodology approach for flexible grippers. *Journal of Intelligent and Robotic Systems*, 15(3):307–328, 1996.

[65] Jennifer Grannen, Yilin Wu, Suneel Belkhale, and Dorsa Sadigh. Learning bimanual scooping policies for food acquisition. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[66] Andrew Gray, Yiqi Gao, J Karl Hedrick, and Francesco Borrelli. Robust predictive control for semi-autonomous vehicles with an uncertain driver model. In *2013 IEEE intelligent vehicles symposium (IV)*, pages 208–213. IEEE, 2013.

[67] Jason M Gregory, Christopher Reardon, Kevin Lee, Geoffrey White, Ki Ng, and Caitlyn Sims. Enabling intuitive human-robot teaming using augmented reality and gesture control. *arXiv preprint arXiv:1909.06415*, 2019.

[68] N Guenard, T Hamel, and V Moreau. Dynamic modeling and intuitive control strategy for an" x4-flyer". In *2005 International Conference on Control and Automation*, volume 1, pages 141–146. IEEE, 2005.

[69] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 66–83, 2017.

[70] Huy Ha, Shubham Agrawal, and Shuran Song. Fit2form: 3d generative model for robot gripper form design. *arXiv preprint arXiv:2011.06498*, 2020.

[71] Sami Haddadin, Alin Albu-Schaffer, Alessandro De Luca, and Gerd Hirzinger. Collision detection and reaction: A contribution to safe physical human-robot interaction. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3356–3363. IEEE, 2008.

[72] Sami Haddadin and Elizabeth Croft. Physical human–robot interaction. In *Springer handbook of Robotics*, pages 1835–1874. Springer, 2016.

[73] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS*, pages 3909–3917, 2016.

[74] Robert XD Hawkins, Noah D Goodman, and Robert L Goldstone. The emergence of social norms and conventions. *Trends in Cognitive Sciences*, 2018.

[75] Eric Heiden, Miles Macklin, Yashraj S Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. Disect: A differentiable simulation engine for autonomous robotic cutting. In *Robotics: Science and Systems*, 2021.

[76] Eric Heiden, David Millard, Hejia Zhang, and Gaurav S Sukhatme. Interactive differentiable simulation. *arXiv preprint arXiv:1905.10706*, 2019.

[77] Laura V Herlant, Rachel M Holladay, and Siddhartha S Srinivasa. Assistive teleoperation of robot arms via automatic time-optimal mode switching. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, pages 35–42. IEEE Press, 2016.

[78] Neville Hogan. Impedance control: An approach to manipulation: Part ii—implementation. 1985.

[79] Mohammadali Honarpardaz, Mehdi Tarkian, Johan Ölvander, and Xiaolong Feng. Finger design automation for industrial robot grippers: A review. *Robotics and Autonomous Systems*, 87:104–119, 2017.

[80] Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017.

[81] Xin-Mao Huang and Can-Rong Zhang. Over-the-air manipulation: An intuitive control system for smart home. In *2017 International Conference on Applied System Innovation (ICASI)*, pages 18–21. IEEE, 2017.

[82] Irfan Hussain, Leonardo Meli, Claudio Pacchierotti, Gionata Salvietti, and Domenico Prattichizzo. Vibrotactile haptic feedback for intuitive control of robotic extra fingers. In *World Haptics*, pages 394–399, 2015.

[83] Shuhei Ikemoto, Heni Ben Amor, Takashi Minato, Bernhard Jung, and Hiroshi Ishiguro. Physical human-robot interaction: Mutual learning and adaptation. *IEEE robotics & automation magazine*, 19(4):24–35, 2012.

[84] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 2961–2970. PMLR, 2019.

[85] Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. Learning preferences for manipulation tasks from online coactive feedback. *The International Journal of Robotics Research*, 34(10):1296–1313, 2015.

[86] Siddarth Jain and Brenna Argall. Robot learning to switch control modes for assistive teleoperation. In *RSS 2016 Workshop on Planning for Human-Robot Interaction: Shared Autonomy and Collaborative Robotics*, 2016.

[87] Siddarth Jain and Brenna Argall. Probabilistic human intent recognition for shared autonomy in assistive robotics. *ACM Transactions on Human-Robot Interaction (THRI)*, 9(1):1–23, 2019.

[88] Shervin Javdani, Henny Admoni, Stefania Pellegrinelli, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization for teleoperation and teaming. *The International Journal of Robotics Research*, 37(7):717–742, 2018.

[89] Hong Jun Jeon, Dylan Losey, and Dorsa Sadigh. Shared autonomy with learned latent actions. In *Proceedings of Robotics: Science and Systems (RSS)*, July 2020.

[90] Hong Jun Jeon, Smitha Milli, and Anca D Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning. *arXiv preprint arXiv:2002.04833*, 2020.

[91] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. *arXiv preprint arXiv:1810.09202*, 2018.

[92] Luke B Johnson, Han-Lim Choi, and Jonathan P How. The role of information assumptions in decentralized task allocation: A tutorial. *IEEE Control Systems Magazine*, 36(4):45–58, 2016.

[93] Rico Jonschkowski and Oliver Brock. State representation learning in robotics: Using prior knowledge about physical interaction. In *Robotics: Science and Systems*, 2014.

[94] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *ACM Siggraph 2005 Papers*, pages 561–566. Proceedings of SIGGRAPH, 2005.

[95] Takayuki Kanda, Takayuki Hirano, Daniel Eaton, and Hiroshi Ishiguro. Interactive robots as social partners and peer tutors for children: A field trial. *Human-computer interaction*, 19(1):61–84, 2004.

[96] Richard M Karp. A simple derivation of edmonds' algorithm for optimum branchings. *Networks*, 1(3):265–272, 1971.

[97] Sean Kerman, Daniel Brown, and Michael A Goodrich. Supporting human interaction with robust robot swarms. In *2012 5th International Symposium on Resilient Control Systems*, pages 197–202. IEEE, 2012.

[98] Piyush Khandelwal, Samuel Barrett, and Peter Stone. Leading the way: An efficient multi-robot guidance system. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, pages 1625–1633. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[99] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application*. MIT press, 2015.

[100] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland., 2008.

[101] Minae Kwon, Erdem Biyik, Aditi Talati, Karan Bhasin, Dylan P Losey, and Dorsa Sadigh. When humans aren't optimal: Robots that collaborate with risk-aware humans. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pages 43–52, 2020.

[102] Mathieu Le Goc, Lawrence H Kim, Ali Parsaei, Jean-Daniel Fekete, Pierre Dragicevic, and Sean Follmer. Zooids: Building blocks for swarm user interfaces. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 97–109, 2016.

[103] Oliver Lemon. Conversational interfaces. In *Data-Driven Methods for Adaptive Spoken Dialogue Systems*, pages 1–4. Springer, 2012.

[104] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information fusion*, 58:52–68, 2020.

[105] Mengxi Li, Rika Antonova, Dorsa Sadigh, and Jeannette Bohg. Learning tool morphology for contact-rich manipulation tasks with differentiable simulation. *arXiv preprint arXiv:2211.02201*, 2022.

[106] Mengxi Li, Alper Canberk, Dylan P Losey, and Dorsa Sadigh. Learning human objectives from sequences of physical corrections. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2877–2883. IEEE, 2021.

[107] Mengxi Li, Minae Kwon, and Dorsa Sadigh. Influencing leading and following in human–robot teams. *Autonomous Robots*, 45:959–978, 2021.

[108] Mengxi Li, Dylan P Losey, Jeannette Bohg, and Dorsa Sadigh. Learning user-preferred mappings for intuitive robot control. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10960–10967. IEEE, 2020.

[109] Sheng Li, Jayesh K Gupta, Peter Morales, Ross Allen, and Mykel J Kochenderfer. Deep implicit coordination graphs for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.11438*, 2020.

[110] Yanan Li, Keng Peng Tee, Wei Liang Chan, Rui Yan, Yuanwei Chua, and Dilip Kumar Limbu. Continuous role adaptation for human–robot shared control. *IEEE Transactions on Robotics*, 31(3):672–681, 2015.

[111] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. *Advances in Neural Information Processing Systems*, 32, 2019.

[112] Martin Liebner, Michael Baumann, Felix Klanner, and Christoph Stiller. Driver intent inference at urban intersections using the intelligent driver model. In *2012 IEEE Intelligent Vehicles Symposium*, pages 1162–1167. IEEE, 2012.

[113] Somchaya Liemhetcharat. Representation, planning, and learning of dynamic ad hoc robot teams. 2013.

[114] Michael L Littman and Peter Stone. Leading best-response strategies in repeated games. In *In Seventeenth Annual International Joint Conference on Artificial Intelligence Workshop on Economic Agents, Models, and Mechanisms*. Citeseer, 2001.

[115] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[116] Dylan P. Losey, Mengxi Li, Jeannette Bohg, and Dorsa Sadigh. Learning from my partner's actions: Roles in decentralized robot teams. In *Proceedings of the 3rd Conference on Robot Learning (CoRL)*, October 2019.

[117] Dylan P Losey, Craig G McDonald, Edoardo Battaglia, and Marcia K O'Malley. A review of intent detection, arbitration, and communication aspects of shared control for physical human–robot interaction. *Applied Mechanics Reviews*, 70(1), 2018.

[118] Dylan P Losey and Marcia K O'Malley. Including uncertainty when learning from human corrections. In *Conference on Robot Learning*, 2018.

[119] Dylan P Losey and Marcia K O'Malley. Trajectory deformations from physical human–robot interaction. *IEEE Transactions on Robotics*, 34(1):126–138, 2017.

[120] Dylan P. Losey, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, and Dorsa Sadigh. Controlling assistive robots with learned latent actions. In *International Conference on Robotics and Automation (ICRA)*, May 2020.

[121] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6379–6390, 2017.

[122] Michael Lutter, Johannes Silberbauer, Joe Watson, and Jan Peters. Differentiable physics models for real-world offline model-based reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4163–4170. IEEE, 2021.

[123] Zhoujie Lyu, Zelu Xu, and JRRA Martins. Benchmarking optimization algorithms for wing aerodynamic design optimization. In *Proceedings of the 8th International Conference on Computational Fluid Dynamics, Chengdu, Sichuan, China*, volume 11, 2014.

[124] Owen Macindoe, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Pomcop: Belief space planning for sidekicks in cooperative games. In *AIIDE*, 2012.

[125] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[126] Christoforos I Mavrogiannis and Ross A Knepper. Decentralized multi-agent navigation planning with braids. In *Algorithmic Foundations of Robotics XII*, pages 880–895. Springer, 2020.

[127] Andre Meixner, Christopher Hazard, and Nancy Pollard. Automated design of simple and robust manipulators for dexterous in-hand manipulation tasks using evolutionary strategies. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 281–288. IEEE, 2019.

[128] Christos Melidis, Hiroyuki Iizuka, and Davide Marocco. Intuitive control of mobile robots: an architecture for autonomous adaptive dynamic behaviour integration. *Cognitive processing*, 19(2):245–264, 2018.

[129] Francisco S Melo and Manuela Veloso. Decentralized MDPs with sparse interactions. *Artificial Intelligence*, 175(11):1757–1789, 2011.

[130] Alexander Mörtl, Martin Lawitzky, Ayse Kucukyilmaz, Metin Sezgin, Cagatay Basdogan, and Sandra Hirche. The role of roles: Physical cooperation between humans and robots. *The International Journal of Robotics Research*, 31(13):1656–1674, 2012.

[131] Alejandro Mottini and Rodrigo Acuna-Agost. Deep choice model using pointer networks for airline itinerary prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1575–1583, 2017.

[132] Lakshmi Nair, Jonathan Balloch, and Sonia Chernova. Tool macgyvering: Tool construction using geometric reasoning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5837–5843. IEEE, 2019.

[133] Patrick Nalepka, Rachel W Kallen, Anthony Chemero, Elliot Saltzman, and Michael J Richardson. Herd those sheep: Emergent multiagent coordination and behavioral-mode switching. *Psychological Science*, 28(5):630–650, 2017.

[134] Rahul Narain, Armin Samii, and James F O'brien. Adaptive anisotropic remeshing for cloth simulation. *ACM transactions on graphics (TOG)*, 31(6):1–10, 2012.

[135] Ashutosh Nayyar, Aditya Mahajan, and Demosthenis Teneketzis. Decentralized stochastic control with partial history sharing: A common information approach. *IEEE Transactions on Automatic Control*, 58(7):1644–1658, 2013.

[136] Truong-Huy Dinh Nguyen, David Hsu, Wee-Sun Lee, Tze-Yun Leong, Leslie Pack Kaelbling, Tomas Lozano-Perez, and Andrew Haydn Grant. Capir: Collaborative action planning with intention recognition. *arXiv preprint arXiv:1206.5928*, 2012.

[137] Jesús R Nieto and Antonio Susín. Cage based deformations: a survey. In *Deformation models*, pages 75–99. Springer, 2013.

[138] Stefanos Nikolaidis, Anton Kuznetsov, David Hsu, and Siddharta Srinivasa. Formalizing human-robot mutual adaptation: A bounded memory model. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, pages 75–82. IEEE Press, 2016.

[139] Stefanos Nikolaidis, Swaprava Nath, Ariel D Procaccia, and Siddhartha Srinivasa. Game-theoretic modeling of human adaptation in human-robot collaboration. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 323–331. ACM, 2017.

[140] Stefanos Nikolaidis and Julie Shah. Human-robot cross-training: computational formulation, modeling and evaluation of a human team training strategy. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, pages 33–40. IEEE Press, 2013.

[141] Marnix Nuttin, Dirk Vanhooydonck, Eric Demeester, and Hendrik Van Brussel. Selection of suitable human-robot interaction techniques for intelligent wheelchairs. In *Proceedings. 11th*

IEEE International Workshop on Robot and Human Interactive Communication, pages 146–151. IEEE, 2002.

[142] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, and Jonathan P How. Decentralized control of partially observable markov decision processes using belief space macro-actions. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5962–5969. IEEE, 2015.

[143] Lisa Ordonez and Lehman Benson III. Decisions under time pressure: How time constraint affects risky decision making. *Organizational Behavior and Human Decision Processes*, 71(2):121–140, 1997.

[144] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 7(1-2):1–179, 2018.

[145] Takayuki Osogami and Makoto Otsuka. Restricted boltzmann machines modeling human choice. In *Advances in Neural Information Processing Systems*, pages 73–81, 2014.

[146] Malayandi Palan, Nicholas C. Landolfi, Gleb Shevchuk, and Dorsa Sadigh. Learning reward functions by integrating human demonstrations and preferences. In *Proceedings of Robotics: Science and Systems (RSS)*, June 2019.

[147] Xinlei Pan, Animesh Garg, Animashree Anandkumar, and Yuke Zhu. Emergent hand morphology and control from optimizing robust grasps of diverse objects. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7540–7547. IEEE, 2021.

[148] James Paulos, Steven W Chen, Daigo Shishika, and Vijay Kumar. Decentralization of multiagent policies by learning what to communicate. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[149] Robert Platt Jr, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. 2010.

[150] Amanda Prorok, Alexander Bahr, and Alcherio Martinoli. Low-cost collaborative localization for large-scale multi-robot systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4236–4241, 2012.

[151] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

[152] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736, 2006.

[153] Joao Rebelo, Thomas Sednaoui, Emiel Boudewijn den Exter, Thomas Krueger, and Andre Schiele. Bilateral robot teleoperation: A wearable arm exoskeleton featuring an intuitive user interface. *IEEE Robotics & Automation Magazine*, 21(4):62–69, 2014.

[154] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[155] Kyle B Reed and Michael A Peshkin. Physical collaboration of human-human and human-robot teams. *IEEE Transactions on Haptics*, 1(2):108–120, 2008.

[156] Roberto Ribeiro, João Ramos, David Safadinho, and António Manuel de Jesus Pereira. Uav for everyone: An intuitive control alternative for drone racing competitions. In *2018 2nd International Conference on Technology and Innovation in Sports, Health and Wellbeing (TISHW)*, pages 1–8. IEEE, 2018.

[157] Ben Robins, Kerstin Dautenhahn, Rene Te Boekhorst, and Aude Billard. Effects of repeated exposure to a humanoid robot on children with autism. *Designing a more inclusive world*, pages 225–236, 2004.

[158] David W Robinson, Thomas R Nixon, Michael Hanuschik, Randal P Goldberg, Jason Hemphill, David Q Larkin, and Paul Millman. Adaptable integrated energy control system for electrosurgical tools in robotic surgical systems, June 28 2016. US Patent 9,375,288.

[159] Nicolas Rojas, Raymond R Ma, and Aaron M Dollar. The gr2 gripper: An underactuated hand for open-loop in-hand planar manipulation. *IEEE Transactions on Robotics*, 32(3):763–770, 2016.

[160] Sara Brin Rosenthal, Colin R Twomey, Andrew T Hartnett, Hai Shan Wu, and Iain D Couzin. Revealing the hidden networks of interaction in mobile animal groups allows prediction of complex behavioral contagion. *Proceedings of the National Academy of Sciences*, 112(15):4690–4695, 2015.

[161] Michael Rubenstein, Adrian Cabrera, Justin Werfel, Golnaz Habibi, James McLurkin, and Radhika Nagpal. Collective transport of complex objects by simple robots: theory and experiments. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 47–54. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[162] Dorsa Sadigh. *Safe and Interactive Autonomy: Control, Learning, and Verification.* PhD thesis, University of California, Berkeley, 2017.

[163] Dorsa Sadigh, Nick Landolfi, Shankar S Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for cars that coordinate with people: leveraging effects on human actions for planning and active information gathering over human internal state. *Autonomous Robots*, 42(7):1405–1426, 2018.

[164] Dorsa Sadigh, S Shankar Sastry, Sanjit A Seshia, and Anca Dragan. Information gathering actions over human internal state. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 66–73. IEEE, 2016.

[165] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems*, volume 2. Ann Arbor, MI, USA, 2016.

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

[167] R Saravanan, S Ramabalan, N Godwin Raja Ebenezer, and C Dharmaraja. Evolutionary multi criteria design optimization of robot grippers. *Applied Soft Computing*, 9(1):159–172, 2009.

[168] Lukas Christoffer Malte Wiuf Schwartz, Adam Wolniakowski, Andrzej Werner, Lars-Peter Ellekilde, and Norbert Krüger. Designing fingers in simulation based on imprints. In *SIMULTECH*, 2017.

[169] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557. PMLR, 2018.

[170] Georg S Seyboth, Dimos V Dimarogonas, and Karl H Johansson. Event-based broadcasting for multi-agent average consensus. *Automatica*, 49(1):245–252, 2013.

[171] Lin Shao, Toki Migimatsu, and Jeannette Bohg. Learning to scaffold the development of robotic manipulation skills. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5671–5677. IEEE, 2020.

[172] Herbert A Simon. Theories of bounded rationality. *Decision and organization*, 1(1):161–176, 1972.

[173] Peng Song and Vijay Kumar. A potential field based approach to multi-robot manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1217–1222, 2002.

[174] Koushil Sreenath and Vijay Kumar. Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots. In *Robotics: Science and Systems (RSS)*, 2013.

[175] Peter Stone, Gal A Kaminka, Sarit Kraus, Jeffrey S Rosenschein, and Noa Agmon. Teaching and leading an ad hoc teammate: Collaboration without pre-coordination. *Artificial Intelligence*, 203:35–65, 2013.

[176] Peter Stone, Gal A Kaminka, Sarit Kraus, Jeffrey S Rosenschein, et al. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI*, page 6, 2010.

[177] Peter Stone and Sarit Kraus. To teach or not to teach?: decision making under uncertainty in ad hoc teams. In *AAMAS*, pages 117–124, 2010.

[178] Ariana Strandburg-Peshkin, Colin R Twomey, Nikolai WF Bode, Albert B Kao, Yael Katz, Christos C Ioannou, Sara B Rosenthal, Colin J Torney, Hai Shan Wu, Simon A Levin, et al. Visual sensory networks and effective information transfer in animal groups. *Current Biology*, 23(17):R709–R711, 2013.

[179] Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.

[180] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2244–2252, 2016.

[181] Priya Sundaresan, Suneel Belkhale, and Dorsa Sadigh. Learning visuo-haptic skewering strategies for robot-assisted feeding. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[182] John Swigart and Sanjay Lall. Optimal controller synthesis for a decentralized two-player system with partial output feedback. In *American Control Conference (ACC)*, pages 317–323, 2011.

[183] Rohan Tiwari, Puneet Jain, Sachit Butail, Sujit P Baliyarasimhuni, and Michael A Goodrich. Effect of leader placement on robotic swarm control. In *AAMAS*, pages 1387–1394, 2017.

[184] Marc A Toussaint, Kelsey Rebecca Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. 2018.

[185] Katherine Tsui, Holly Yanco, David Kontak, and Linda Beliveau. Development and evaluation of a flexible interface for a wheelchair mounted robotic arm. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, pages 105–112. ACM, 2008.

[186] Toshiaki Tsuji, Jun Ohkuma, and Sho Sakaino. Dynamic object manipulation considering contact condition of robot with tool. *IEEE Transactions on Industrial Electronics*, 63(3):1972–1980, 2015.

[187] Amos Tversky and Daniel Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and uncertainty*, 5(4):297–323, 1992.

[188] VB Velasco and Wyatt S Newman. Computer-assisted gripper and fixture customization using rapid-prototyping technology. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, volume 4, pages 3658–3664. IEEE, 1998.

[189] Zijian Wang and Mac Schwager. Force-amplifying n-robot transport system (force-ants) for cooperative planar manipulation without communication. *The International Journal of Robotics Research*, 35(13):1564–1586, 2016.

[190] Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. *arXiv preprint arXiv:2103.16021*, 2021.

[191] Adam Wolniakowski, Jimmy A Jorgensen, Konstantsin Miatliuk, Henrik Gordon Petersen, and Norbert Kruger. Task and context sensitive optimization of gripper design using dynamic grasp simulation. In *2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 29–34. IEEE, 2015.

[192] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[193] Annie Xie, Dylan Losey, Ryan Tolsma, Chelsea Finn, and Dorsa Sadigh. Learning latent representations to influence multi-agent interaction. In *Proceedings of the 4th Conference on Robot Learning (CoRL)*, November 2020.

[194] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An end-to-end differentiable framework for contact-aware robot design. *arXiv preprint arXiv:2107.07501*, 2021.

[195] Kimitoshi Yamazaki, Yoshiaki Watanabe, Kotaro Nagahama, Kei Okada, and Masayuki Inaba. Recognition and manipulation integration for a daily assistive robot working on kitchen environments. In *2010 IEEE International Conference on Robotics and Biomimetics*, pages 196–201. IEEE, 2010.

[196] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10(12):399, 2013.

[197] Chih-Han Yu, Justin K Werfel, and Radhika Nagpal. Collective decision-making in multi-agent systems by implicit leadership. 2010.

[198] Hao Zhang, Pinxin Long, Dandan Zhou, Zhongfeng Qian, Zheng Wang, Weiwei Wan, Dinesh Manocha, Chonhyon Park, Tommy Hu, Chao Cao, et al. Dorapicker: An autonomous picking system for general objects. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 721–726. IEEE, 2016.

[199] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE, 2020.

[200] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438, 2008.

[201] Brian D Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, J Andrew Bagnell, Martial Hebert, Anind K Dey, and Siddhartha Srinivasa. Planning-based prediction for pedestrians. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3931–3936. IEEE, 2009.