Efficient Human-Robot Collaboration through Inverse Reinforcement Learning

by

Prasanth Sengadu Suresh

(Under the Direction of Prashant Doshi)

Abstract

Human-robot collaboration (HRC) stands at the forefront of scientific innovation and technological advancements. By combining human agents' cognitive acuity and dexterity with robotic agents' endurance and efficiency, HRC systems can seamlessly tackle complex tasks. To realize this ambitious goal, developing sophisticated models that capture the intricacies of humans and robots navigating shared workspaces is crucial. Additionally, intelligent algorithms that leverage these models to learn behavioral policies are essential for guiding agents appropriately during task execution. To this end, we develop novel multiagent models and inverse reinforcement learning (IRL) algorithms to solve various HRC problems. We begin by addressing a key challenge in robotics: learning a near-optimal policy from imperfect and incomplete input data. We achieve this by generalizing the well-known maximum-a-posteriori Bayesian IRL technique to sum out the occluded portions of the trajectory and then extending it with an observation model to account for perception noise. Subsequently, in HRC, we tackle three key challenges: 1. Decentralized collaboration with sparse interactions: we develop a decentralized adversarial IRL technique (Dec-AIRL). 2. Open HRC to allow agents to enter and exit the task as needed: we propose a new open HRC model (oDec-MDP) and formulate a novel IRL method (oDec-AIRL) to enable open collaboration. 3. Type-based collaboration by modeling dynamic agent types: we propose a new type-based mixed-observability model (TB-Dec-MDP) and a corresponding IRL technique (TB-Dec-AIRL) to learn a type-specific reward function and its corresponding policies (one for each agent). In this dissertation, we present these challenges through realistic HRC domains and demonstrate how our methods significantly improve upon existing art.

Index words: IRL, decision-making under uncertainty, human-robot collaboration, agent openness, learning from observations.

Efficient Human-Robot Collaboration through Inverse Reinforcement Learning

by

Prasanth Sengadu Suresh

M.S., Colorado School of Mines, Golden, CO, 2018

A Dissertation Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Doctor of Philosophy

Athens, Georgia

2024

Efficient Human-Robot Collaboration through Inverse Reinforcement Learning

by

Prasanth Sengadu Suresh

Major Professor:	Prashant Doshi
Committee:	Ramviyas Nattamai Parasuraman, Jin Sun Vaibhav Unhelkar

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
December 2024

Dedication

"I have approximate answers, possible beliefs, and different degrees of certainty about different things, but I'm not absolutely sure of anything. There are many things that I don't know anything about, such as if it means anything to ask why we are here." — Richard Feynman

Dedicated to my parents and my sister, who have been my pillars of strength throughout.

Acknowledgments

I would like to wholeheartedly thank my advisor Dr. Prashant Doshi, who has guided, supported, and groomed me into the researcher I am today. His mentorship has not only enhanced my research skills but also instilled in me a profound appreciation for the rigor and curiosity that define the scientific process. Through his guidance, I have learned the importance of perseverance and the value of approaching each challenge with an open mind. Dr. Doshi's dedication to my growth has been unwavering, shaping not only my professional path but also my personal development. I am deeply grateful for his unwavering support, insightful feedback, and countless hours of investment in my success. His influence will undoubtedly continue to guide and motivate me throughout my career.

I would also like to thank my committee members Dr. Ramviyas Nattamai Parasuraman, Dr. Jin Sun, and Dr. Vaibhav Unhelkar, for their invaluable feedback over the years. I would like to thank the School of Computing and the Graduate School for their financial support of my travels to research conferences. I also extend my gratitude to the staff members of the School of Computing, both current and former, for their kindness and welcoming disposition, which helped me feel at home.

Finally, I thank my fellow researchers at the THINC Lab and my collaborators for the insightful discussions and valuable contributions. I would like to specially thank Ehsan Asali, who has been like an elder brother to me throughout my doctoral journey. He has not only helped me out professionally, but has also stood by me when I was disheartened, upset, or frustrated by life's adversities. I am deeply grateful for all the experiences this journey has brought, both the highs and the lows, as they have shaped me into the person I am today.

Publication List

1. Sengadu Suresh, Prasanth, Bikramjit Banerjee, and Prashant Doshi. "Adaptive Human-Robot Collaboration using Type-Based IRL." Submitted to 2024 International Conference on Autonomous Agents and Multiagent Systems., 2024.

2. Sengadu Suresh, Prasanth, Diego Romeres, Prashant Doshi, and Siddarth Jain. "Open Human-Robot Collaboration Systems (OHRCS): A Research Perspective." in The Sixth IEEE International Conference on Cognitive Machine Intelligence 2024.

3. Sengadu Suresh, Prasanth, Siddarth Jain, Prashant Doshi, and Diego Romeres. "Open human-robot collaboration using decentralized inverse reinforcement learning." in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2024.

4. Sengadu Suresh, Prasanth, Yikang Gui, and Prashant Doshi. "Dec-AIRL: Decentralized Adversarial IRL for Human-Robot Teaming." in the 2023 International Conference on Autonomous Agents and Multiagent Systems. 2023.

5. Suresh, Prasanth Sengadu, and Prashant Doshi. "Marginal MAP estimation for inverse RL under occlusion with observer noise." in Uncertainty in Artificial Intelligence. PMLR, 2022.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots, equipped with sensory perception and advanced decision-making capabilities, excel at handling simple and repetitive tasks. In contrast, humans are proficient in dexterous manipulation and complex cognitive tasks. The overarching goal of human-robot collaboration (HRC) is to develop systems that harness the complementary strengths of both human and robotic agents, driving progress and innovation. Consequently, achieving seamless collaboration between these agents becomes essential. Collaborative robots (cobots) — robotic manipulators with built-in safety features — emerge as the ideal candidates for this endeavor (Villani et al., 2018).

(a) (b) (c) (d)

Figure 1.1: Figure shows collaborative robots that I have worked with during my doctoral journey. (a) Sawyer by Rethink Robotics (now Hahn Robotics) a 7 DOF cobot with Robotiq 2F-85 gripper holding a faux onion (b) Universal Robots UR3e, a 6 DOF cobot with OnRobot sg-b soft gripper and a faux onion placed on a conveyor belt (c) Franka Research-3 7 DOF cobot with default gripper assembling a wooden table (d) KUKA lbr iisy 3 r760 6 DOF cobot with Ubiros Gentle Pro soft gripper.

For a cobot to work seamlessly alongside a human in a shared workspace, its actions at every step must be guided by an optimal behavioral policy learned through an accurate decision-making model, using feedback from the environment. Preprogramming cobots with responses to all possible situations is both inefficient and impractical. Motivated by potential real-world HRC domains such as human-

robot collaborative produce sorting and a collaborative furniture assembly domain, this thesis describes common problems in human-robot collaboration (HRC), formulates appropriate models, and proposes novel learning-from-observations (LfO) techniques to solve them.

1.1 Answering some commonly posed questions

Over the years, many relevant questions have been raised regarding some of the research and implementation choices I have made. Since some of these answers require a deeper understanding of background concepts, I provide a high-level justification in this section and elaborate further in the upcoming sections.

Q: Why choose a robotics-based solution over an automation-based solution?

A: We opt for a robotics-based solution over an automation-based approach because:

1. A cobot is safe to deploy in an environment shared by humans due to its built-in safety features. An automation-based solution is generally blind to dynamic obstacles like humans and may lead to undesirable consequences.

2. A cobot (along with its sensors) has a smaller form factor, greater economy, and turnkey compatibility with most industrial setups.

3. If multiple cobots are deployed along a conveyor line, the task is distributed among them, and a single cobot failure would not significantly impact productivity. In contrast, the failure of an automation system may be a single-point failure leading to substantial losses.

Q: How do you define human-robot collaboration?

A: Human-robot collaboration can be formally defined as a scenario where human and robotic agents work together to efficiently achieve a common goal. We focus on collaboration within shared workspaces and classify HRC into two scenarios:

1. Non-interactive collaboration: Where agents work independently towards a common goal while traversing a shared workspace.

2. Interactive collaboration: Where all agents try to manipulate the same object or aim to reach the same goal location simultaneously. In the latter case, agents need to adjust their behavior to avoid collisions, while in the former case, they work in concert to manipulate a common object.

Q: Why use a multiagent model to represent the human-robot team instead of just modeling the robot using a single-agent model?

A: We choose a multiagent model because it allows us to accurately capture the behavioral interdependencies and interaction intricacies from a third-person perspective. Thus, we model both the human and robot, as a team. We learn a team reward function and decentralized policies for each agent, enabling each agent to work independently while considering the other agents' actions in their decision-making.

Q: Why use IRL instead of imitation learning techniques like behavior cloning?
A: A key difference between imitation learning techniques like behavior cloning and inverse RL is that the former focuses on minimizing the classification error between the expert's behavior and the learned behavior, while the latter focuses on learning a behavior with equivalent value as the expert, which is a more relevant objective. Consequently, the IRL learner learns to optimally solve the task, instead of blindly mimicking the behavior observed in the expert demonstrations. This leads to better generalization.
Q: How does IRL help in the context of HRC?
A: The problem of HRC involves learning intricate interactions necessary to navigate a shared workspace while optimally collaborating to solve the task. Designing a reward function for this can be quite challenging and often intractable. By observing an expert human-human team perform the task efficiently, the IRL learner can deduce the underlying preferences through a reward function that can effectively guide the human and robotic agents during execution.
Q: Why not learn a history-dependent policy?
A: A history-dependent policy typically conditions its actions on the action-observation history up to the current timestep. This may not be necessary in all HRC scenarios. For instance, in cases where the order of task execution plays a vital role in determining the optimality of the solution, a history-dependent policy is appropriate since it considers the tasks completed thus far before suggesting an action. However, in scenarios like the onion-sorting domain, where the order of onions sorted is irrelevant, a history-dependent policy may be an excessive measure that delays convergence.
Q: How does the policy handle the asynchronicity of the human and robot during execution?
A: Currently, it does not. This is an inherent problem with Markovian models. Markovian models assume that each action lasts one timestep and all agents transition to the next timestep simultaneously. Hence, most simulators are built this way (including OpenAI Gym). During execution, it usually does not matter much since the agents are working in a decentralized fashion (note that agents collaboratively manipulating the same object are always synced). However, a previous method—Mac-Dec-POMDPs (Amato et al., 2019)—breaks each action into "macro actions" and attaches a stochastic termination condition to them. This alleviates the problem by allowing agents to stop in the middle of a larger action containing multiple macro actions. By allowing temporally extended actions, this method uses a Semi-MDP-based (D. Bertsekas, 2012; Mahadevan et al., 1997; Puterman, 1994) solution to address asynchronicity. However, this model is still untested in HRC scenarios and hence it is hard to ascertain its effectiveness.

1.2 Cobots in real-world environments

Cobots, much like industrial robots, come in various sizes and capabilities, such as degrees of freedom and payload capacity, and can be used in a wide range of contexts, from educational research to industrial applications (Vicentini, 2021). However, unlike industrial robots, cobots have built-in safety features that allow them to safely coexist in shared workspaces. This provides the opportunity to develop intelligent algorithms that enable cobots to sense, adapt, and work synergistically with humans.

Figure 1.2: Stakeholders witnessing cobot UR3e deployed at Vidalia Onion and Vegetable Research Center - a real produce processing shed.

The seamless functioning of cobots in shared workspaces involves solving a wide array of research and implementation challenges that can be summarized into the following overarching categories:

1. A priori knowledge of the static obstacles within the cobot's reachable workspace.
2. Input data processing from feedback devices using trained ML models (Redmon et al., 2016; Soans et al., 2020; C.-Y. Wang et al., 2023; Wojke et al., 2017) to recognize desiderata.
3. Intelligent learning paradigms like inverse reinforcement learning (IRL) to learn behavioral policies from expert demonstrations.
4. A pipeline architecture stitching together all these pieces to guide the cobot's actions.

Since points 1, 2, and 4 are technical challenges, we discuss them in Chapter A. The remainder of this document focuses on the research challenges surrounding humans and cobots working in shared workspaces while following learned policies. We briefly explain the challenges associated with processing data obtained from sensor recordings, as it is directly relevant to the first challenge we address. We strongly recommend the reader consult Chapter A to gain a comprehensive understanding of the complexities of real-world HRC.

1.3 Processing sensor feedback

One of the main challenges of IRL is that, to learn a meaningful and well-defined reward function from expert demonstrations, the data provided must sufficiently describe all aspects needed to solve the task optimally. Subsequently, during execution, proper feedback must be obtained from sensors for the learned policy to guide the cobot appropriately. However, clearly capturing all the vital components of the task through raw data is non-trivial. To describe these challenges systematically, we split the sensor data processing into two main parts:

1. Processing expert task demonstrations to obtain state-action trajectories needed for IRL training.
2. Processing live RGB-D frames during execution to identify desiderata, including the human state.

1.3.1 Processing expert demonstrations

To obtain expert task demonstrations from raw RGB images, we use a popular deep-learning-based technique called SA-Net (Soans et al., 2020), which takes in RGB frames of the demonstrations and returns the state-action labels corresponding to each input frame. To address different research goals, we trained two versions of SA-Net. The first version was trained to detect a single expert performing the task, which yielded incomplete and imperfect state-action trajectories due to immovable environmental obstacles and sensor noise. The second version was trained to generate joint state-action labels from human-human team demonstrations, which are needed to obtain behavioral policies for the human and robot using IRL.

(a) (b)

Figure 1.3: (a) (above) SA-Net detection of a single expert inspecting an unblemished onion, (below) A person walking by occludes part of the camera view, causing an occluded observation (b) SA-Net detections of time-synchronized frames of a human-human team demonstrating collaborative task execution. (above) human-1 inspecting an unblemished onion. (below) human-2 picking an onion from the conveyor.

1.3.2 Processing live RGB-D frames during execution

In order to update attributes of the cobot's local states, it needs to observe aspects of the task, and in the case of HRC – certain aspects of the human teammate. This updated current state is then used to obtain the policy action for the cobot to execute.

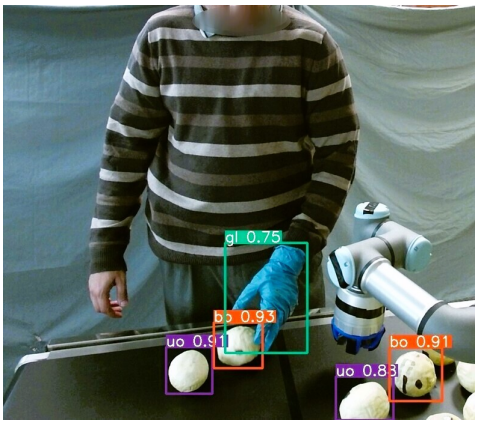
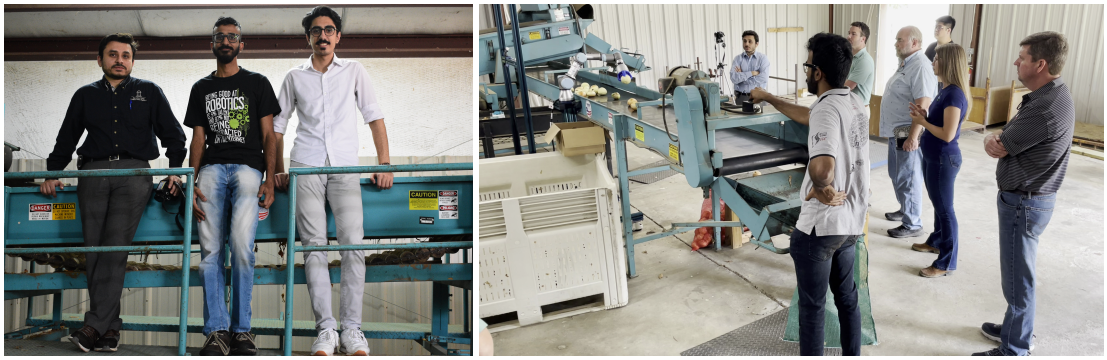


Figure 1.4: Cobot UR3e's live camera frame showing YOLOv7-Deepsort detections of desiderata.

To recognize desiderata from RGB frames of a live camera feed, we use a popular object detection network – YOLOv7 (C.-Y. Wang et al., 2023) with a tracking network called DeepSort (Wojke et al., 2017).

1.4 UR3e deployed and tested at VOVRC

As part of our HRC research and development, we conducted some field trials that involved talking to stakeholders like farmers, industrialists, and workers at produce processing sheds. We also deployed and tested our setup at VOVRC on three separate occasions (2019, 2021, 2022), iteratively improving the cobot's performance every time, based on the feedback we received.



(a) (b)

Figure 1.5: (a) Our first visit to VOVRC in 2019 and (b) Our most recent UR3e sorting demo in 2022 where several stakeholders were present to witness our sorting demo and provided invaluable feedback.

As can be seen from Figures 1.6d and 1.6e, human workers stand side-by-side and opposite each other to sort onions on the conveyor line.

(a) (b) (c)

(d) (e) (f)

Figure 1.6: (a) Collecting measurements of the conveyor system at VOVRC to build the cobot environment appropriately. (b) Brainstorming ideas on how the cobot could be deployed as a turnkey solution that leaves the existing conveyor system undisturbed. (c) Setting up our cobot and its camera feed for demo and testing. (d – f) Field research conducted at multiple sorting sheds and farms to gather information about human workers and their existing sorting procedures.

During our 2022 visit, we successfully demonstrated our system to many stakeholders, who were impressed by the cobot's performance (Figure 1.5). Until this time, we had not included the aspect of the cobot working alongside the human as a team and had only focused on deploying the cobot as a stand-alone agent.

1.5 Thesis structure

In this thesis, we handle a variety of research challenges in the context of HRC and solve them using IRL-based solutions. The rest of the thesis is organized as follows:

1. In Chapter 2, we introduce the most common decision-making framework used in RL and inverse RL settings - Markov decision processes (MDP). We explain why this is insufficient to model multiagent domains, why a decentralized variant of the MDP is the most appropriate for our case

and formally define this framework. Subsequently, we describe the foundational methods in inverse RL.

2. In Chapter 3, we describe some fundamental concepts used in pragmatic RL and IRL such as generalized advantage estimate, KL divergence, etc. We conclude by comparing popular RL methods and referencing some popular open-source RL libraries that provide stable implementations.

3. In Chapter 4, we compare and contrast existing works that address similar challenges using IRL or related techniques and explain how our methods improve upon them.

4. In Chapter 5, we tackle a challenge ubiquitous to real-world domains like robotics — noise and occlusions in training data. We generalize the well-known MAP Bayesian IRL technique to obtain a marginal MAP inference by summing out the missing portions of the input trajectories; we extend this using a sensor model to handle noisy input observations.

5. In Chapter 6, we delve into human-robot collaboration by ascribing a Dec-MDP to a dyadic human-robot team and generalizing a popular deep IRL method - adversarial IRL. We propose our novel IRL technique Dec-AIRL that learns a common team reward function and a decentralized vector of policies (one for each agent), from expert human-human team demonstrations.

6. In Chapter 7, we tackle a completely unexplored challenge in HRC - open collaboration. By developing a novel multiagent model - oDec-MDP and a corresponding IRL technique to solve it - oDec-AIRL, we allow agents to enter and exit the collaboration as needed. This allows the policy to learn when the human is required for the task and only summon them when needed; consequently saving the human's time and energy.

7. In Chapter 8, we allow the agents to model each other by maintaining a belief over each others' dynamic types within our novel multiagent model - TB-Dec-MDP. Enabling the robot to detect human fatigue through behavior changes and learn type-contingent behavior that optimizes task completion using our novel TB-Dec-AIRL.

8. In Chapter 9, we describe some potential future directions for this research in HRC using IRL and some potential ways to address them. Finally, In Chapter 10, we summarize the accomplishments of this thesis and conclude by highlighting the importance of seamless HRC in future industrial and research applications.

Chapter 2

Technical background - I

2.1 Modelling the environment

An accurate model of the environment is crucial to make decisions under uncertainty. This model must be sophisticated enough to capture the task, the agent(s) attributes, and environmental uncertainties accurately. Using such a model, an intelligent algorithm can be designed to learn a behavioral policy that guides the agent appropriately. Note that the reader is assumed to have a basic understanding of foundational concepts like i.i.d, Boltzmann energy function, Lagrangian optimization, etc. Please refer to Chapter 3 for a quick introduction to some of these fundamental concepts.

2.1.1 Markov decision processes

A common modeling framework adopted throughout reinforcement learning literature is the Markov decision processes (MDP) (Puterman, 1994). Formally, the MDP of an expert is defined as a quadruple $\langle S, A, T, R \rangle$, where S is the set of states defining the environment, A is the expert's set of possible actions, $T : S \times A \times S \to [0, 1]$ gives the transition probabilities from any given state to a next state for each action, and $R : S \times A \to \mathbb{R}$ is the reward function modeling the expert's preferences, rewards, or costs of performing an action from a state. Typically, the learner is aware of the expert's S, A, and T, but not R. Model-free techniques don't assume access to T either.

An important assumption needed to correctly apply a MDP to an agent is that the current state subsumes all relevant information needed for the agent's decision, i.e. the agent's history has no bearing on the policy or transition function. This is known as the Markov assumption and is formally described as: $T(S_{1:t}, A_{1:t}, S_{t+1}) = T(S_t, A_t, S_{t+1})$. Note that we drop subscripts in common usage [1]. A (stationary) policy for a MDP is a mapping from states to actions $\pi : S \to A$ and the discounted, infinite-horizon value of a policy π for a given reward function R at some state $s \in S$, with t denoting time steps is given

[1] It is important to note that some domains (especially partially observable ones) necessitate the need to condition the policy on the action-observation history and hence require a more general variant of the MDP - POMDP (Kaelbling et al., 1998). We do not consider those cases here.

by:

$$E_s\Big[V^{\pi}(s)\Big] = E\Big[\sum_{t=0}^{\infty}\gamma^t R(s^t,\pi(s^t))|s,\pi\Big].$$

In this work, we assume that the expert is a rational agent that follows the optimal policy while executing actions at every state. A rational agent is one that monotonically prefers policies that produce higher expected rewards. The forward reinforcement learning problem entails solving the aforementioned MDP by interacting with the environment to obtain feedback in the form of rewards. These rewards are used within the Bellman value function (Eqn (2.1)) to arrive at the optimal policy.

$$V^0(s) \;=\; \max_a R_{\boldsymbol{\theta}}(s,a); \quad V^t(s) \;=\; \max_a\Big(R_{\boldsymbol{\theta}}(s,a) + \gamma\sum_{s'}T(s,a,s')\,V^{t-1}(s')\Big). \tag{2.1}$$

Once the policy is learned, the agent may now refer to the policy to obtain the optimal action to perform at any given state. This produces a trajectory: $X \;=\; \Big(< s,a,r >, < s,a,r >, ...\Big)$ describing the sequence of states arrived at, actions chosen, and rewards received by the agent over time. Note that expert trajectories used in IRL only contain the state-action pairs and not the step-wise reward as shown in Eq. (2.2).

2.1.2 Decentralized Markov decision processes

A single-agent MDP as defined in Section 2.1.1, is insufficient to represent scenarios where multiple agents are involved. Such a system is formally known as a *multiagent system* (Shoham & Leyton-Brown, 2008). The goal(s) of the agents in a multiagent system could be cooperative, competitive, of self-interest, or a mixture of the three. Each agent typically uses all available information about the other agents to make a decision at every step.

Depending on the task, the nature of the agents, and the amount of information they have access to (Xuan et al., 2000), a multiagent system can be modeled in a number of different ways:

1. **Markov game** - where each agent works to maximize its own reward until an equilibrium condition is met. Here each agent gets an individual reward for their actions and their policy maps the global state to their local actions.

2. **Multi-agent Markov decision process (MMDP)** - where every agent is aware of the global state, including attributes specific to themselves, the other agents, and the task. All agents receive a shared reward as a result of their collective actions. Agents follow a joint policy that maps global states to global actions.

3. **Decentralized Markov decision process (Dec-MDP)** - specifically a locally fully-observable variant (Goldman & Zilberstein, 2003) is a model where each agent is only aware of their own local state

- containing attributes specific to themselves and any directly observable task-related attributes. Like a MMDP, agents receive a common system reward as a result of their collective actions, however, each agent learns an independent policy, mapping their local state to a local action.

Note that all of the above models have a partially observable variant that we do not discuss in this thesis. Since our focus is on HRC, which is by definition decentralized and collaborative, we choose the Dec-MDP to model our collaborative scenarios. A two-agent Dec-MDP is defined using the following tuple:

$$\mathcal{DM} \triangleq \langle S, A, T, R \rangle$$

where the global state, $S = S_i \times S_j$. Here, S_i and S_j are the locally observed states of the two agents i and j, which when combined yield the complete global state of the system; $A = A_i \times A_j$ is the set of joint actions of the two agents; $T : S \times A \times S \rightarrow [0, 1]$ is the transition function of the multi-agent system; and $R : S \times A \rightarrow \mathbb{R}$ is the shared reward function. Note that the latter is unknown, whereas the rest of the elements of the model are usually known. As such, the agents know their own local state only, any observable parameters relevant to the interaction, and can act independently while optimizing a task-centric, shared reward (Melo & Veloso, 2011). Thus, our Dec-MDP is a locally fully observable model whose local states when combined yield the fully observable global state, per Goldman and Zilberstein, 2003's categorization of such decentralized models. The solution to a Dec-MDP is a vector of policies, $\boldsymbol{\pi}^* = \langle \pi_i^*, \pi_j^* \rangle$, where $\pi_i^* : S_i \rightarrow A_i$ and analogously for π_j^*. If the interactions between the two agents are sparse and can occur at some joint states only, we may leverage this domain structure to simplify the model. Let $S_I \in S$ be the set of states where an interaction may occur, and $S_{NI} = S/S_I$ be the remaining states. Then, we may specify the transition and reward functions as:

$$T(s, \mathbf{a}, s') = \begin{cases} Pr(s'|s, \mathbf{a}) & \text{if } s \in S_{NI} \\ Pr(s'_i|s_i, a_i) \cdot Pr(s'_j|s_j, a_j) & \text{if } s \in S_I \end{cases};$$

$$R(s, \mathbf{a}, s') = \begin{cases} R(s, \mathbf{a}, s') & \text{if } s \in S_{NI} \\ R_i(s_i, a_i, s'_i) + R_j(s_j, a_j, s'_j) & \text{if } s \in S_I \end{cases}$$

where $s \in S$, $s' \in S$, and $\boldsymbol{a} \in A$. Thus, we may exploit the transition and reward independence in the non-interaction states.

2.2 Inverse Reinforcement Learning

While reinforcement learning (RL) is a valid method to tackle the scenarios mentioned in Section 1.2, designing an appropriate reward function that adequately captures the intricacies of real-world collaborative behaviors is non-trivial (Arora & Doshi, 2021a). Therefore, we move to the paradigm of learning from observations (LfO), which entails observing an expert performing a task and learning to perform it similarly. LfO has several branches like inverse RL (IRL), imitation learning (IL), kinesthetic teaching, etc. Branches like IL often use techniques like behavior cloning (BC) (Torabi et al., 2018) which is essentially a supervised learning approach to learn a direct mapping from the states to the actions allow-

ing the learner to mimic the expert. However, these techniques perform poorly on tasks with a lot of uncertainty (Ghasemipour et al., 2020b).

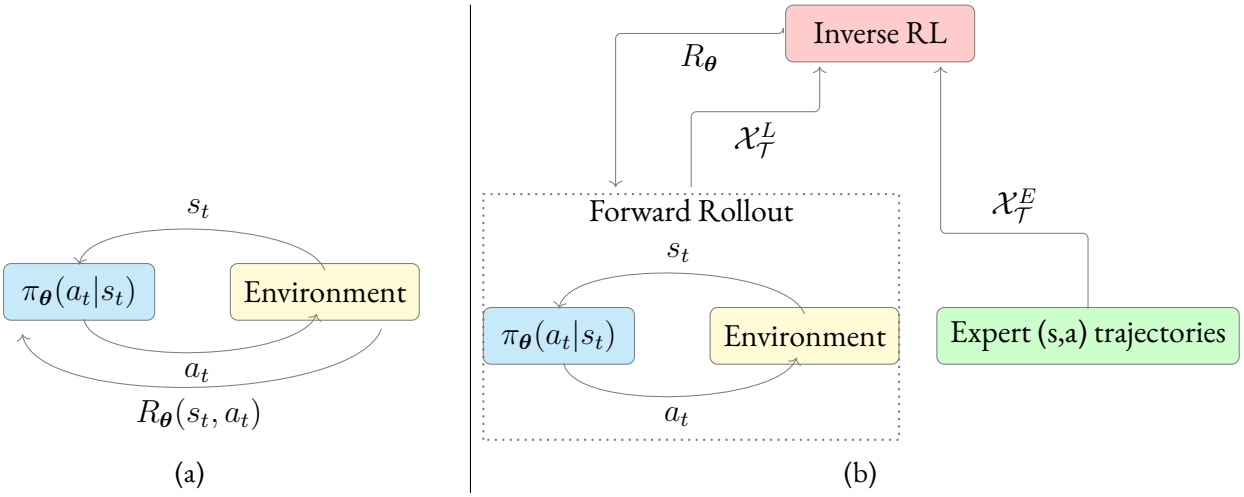


Figure 2.1: Figure shows a comparison between the forward and inverse reinforcement learning paradigms. (a) Shows the forward reinforcement learning architecture where the agent interacts with the environment by performing an action (a_t) in the current state (s_t), receives a reward ($R_{\boldsymbol{\theta}}(s_t, a_t)$) and reaches the next state. Using this experience, it learns a policy ($\pi_{\boldsymbol{\theta}}(a_t|s_t)$). (b) Shows the inverse reinforcement learning paradigm where the observer uses expert demonstrations ($\mathcal{X}_{\mathcal{T}}^{E}$) and sampled state-action pairs from the currently learned policy ($\mathcal{X}_{\mathcal{T}}^{L}$) to learn a reward function ($R_{\boldsymbol{\theta}}$) that can be used in forward-rollout to update the learned policy.

Conversely, inverse reinforcement learning (Russell, 1998) refers to the problem of inferring the expert's preferences through a learned reward function using observations of the expert performing the task. This learned reward function can then be used in forward RL to solve the task optimally. **A key difference between imitation learning and inverse RL is that the former focuses on minimizing the classification error between the expert's behavior and the learned behavior, while the latter focuses on learning a behavior with equivalent value as the expert, which is a more relevant objective** (Abbeel & Ng, 2004b). IRL literature calls the former objective - learned behavior accuracy (LBA) and the latter objective - inverse learning error (ILE). One of the problems with IRL is that it is an ill-posed problem. Meaning that the agent could learn an uncountable set of different reward functions that accurately explain the observed behavior. This is known as the *degeneracy problem* in IRL literature (Arora & Doshi, 2021a). Several approaches have been proposed to solve this problem adequately, including max-margin IRL, maximum entropy IRL, and Bayesian IRL.

Ng, Russell, et al., 2000 proposed one of the earliest solutions to the IRL problem that constructs a linear program to maximize the margin of value between the action chosen by the expert in each state and all other actions. They further use a penalty term that encourages the use of smaller magnitude rewards. This can then be solved as a linear program to obtain the reward value for each state. They formulate the

reward function as a linearly-weighted sum of K basis functions:

$$R_{\boldsymbol{\theta}}(s,a) \triangleq \sum_{k=1}^{K} \boldsymbol{\theta}_k \phi_k(s,a)$$

where K is finite and non-zero, $\boldsymbol{\theta_k}$ are the weights, and $\phi : (S, A) \rightarrow (0, 1)$ is a *feature function*. A binary feature function maps a state from the set of states S and an action from the set of actions A to 0 (false) or 1 (true).

Notice that this approach assumes access to a single optimum policy from the expert, and that this representation requires pre-defining these features. An alternative is to learn these feature functions (Levine et al., 2010) or use a neural network representation (Wulfmeier & Posner, 2015), which automatically identifies the features but typically requires far more demonstrations to converge. The learner's task now becomes finding a vector of weights $\boldsymbol{\theta}$ that complete the reward function. One way to narrow down the search space is by calculating an expectation of features that any given policy would obtain during execution:

$$\phi_\pi = \sum_s \mu_\pi(s)\, \phi(s, \pi(s)).$$

where $\mu_\pi(s)$ is the state visitation frequency of state s, as explained in Eq. (2.3) (μ_π^0 defines the initial state distribution). Let $\mathcal{X}^E$ be the set of expert demonstrations and let X denote a complete trajectory, where $X \in \mathcal{X}^E$ is given by [2],

$$X = (s^1, a^1, s^2, a^2, s^3, ..., s^\mathcal{T}, a^\mathcal{T}). \tag{2.2}$$

Let the set of input trajectories of finite length $\mathcal{T}$ generated by an MDP attributed to the expert be, $\mathcal{X}_\mathcal{T}^E = \{X | X = Y \bigcup Z\}$.

Here, Y is the observable portion and Z is the occluded part of a trajectory X. A complete trajectory X is a sequence, $X = (s^1, a^1, s^2, a^2, s^3, ..., s^\mathcal{T}, a^\mathcal{T})$; some of these may be occluded (For the time being, we consider all expert trajectories to be occlusion-free. Occlusions in the context of robotics will be introduced in Section 5.1). Now we can calculate the empirical feature expectations of the expert from an observed trajectory as:

$$\hat{\phi}_E = \frac{1}{|X|} \sum_{(s,a) \in X} \phi_k(s,a).$$

Once we reach a set of weights where the corresponding reward function generates a policy satisfying the constraint: ($\phi_\pi = \hat{\phi}_E$) the IRL problem is considered to be solved. At this point, we can calculate the

[2] Note the important distinction that expert trajectories in IRL do not include the immediate reward obtained vis-a-vis the definition of a trajectory defined at the end of 2.1.1.

gradient of the weight vector as

$$\nabla \boldsymbol{\theta} = \phi_\pi - \hat{\phi}_E.$$

wherein if the learned policy results in a feature expectation that is higher than the expert's, the corresponding weight must be reduced, and vice versa.

$$\mu_\pi(s) = \mu_\pi^0(s) + \gamma \sum_{s'} T(s, \pi(s), s')\mu_\pi(s') \tag{2.3}$$

Since the set of weight vectors that produce a policy matching this criterion is infinite, Abbeel and Ng, 2004a formulate a max-margin approach to maximize the value difference between the expert policy and all previously found policies to arrive at the following quadratic program:

$$\max_{m,\theta} m \quad \text{subject to} \quad \begin{array}{l} \boldsymbol{\theta}^T \hat{\Phi}_E \geq \boldsymbol{\theta}^T \Phi^{(i)} + m, \;\; i = 0, \ldots, N-1; \\ \| \boldsymbol{\theta} \|_2 \leq 1. \end{array} \tag{2.4}$$

Where $\boldsymbol{\theta}$ is a vector of weights, $\hat{\Phi}_E$ is the expert's empirical feature expectations, and $\Phi^{(i)}$ is the feature expectations from the policy found at iteration i. Equation 2.4 tries to solve for a reward function on which the expert does better, by a "margin" of m, than any of the N policies previously obtained. This program can be solved with any quadratic solver, such as SVM, to obtain the candidate feature weights. A convex combination of all produced policies is then generated, weighted such that the feature expectation of the distribution over these policies matches exactly to the expert's empirical feature expectations.

2.3 Bayesian IRL and MAP inference

One of the most popular methods in IRL is Bayesian IRL (BIRL) (Ramachandran & Amir, 2007a). Before we dive into BIRL, let's take a quick look at Bayes' theorem. The Bayesian probability theory, named after Thomas Bayes, describes the probability of a hypothesis (H) being true, given some evidence (E). In order to do so it uses, the probability of seeing the evidence E, given that the hypothesis H is true - this is called the *likelihood*, and the prior knowledge about the probability of the hypothesis H being true - called the *prior*. This is then divided by the total probability of seeing the evidence E, to arrive at the *posterior* inference. Formally, this can be represented as:

$$\textit{Posterior} \; \Pr(H|E) \; = \; \frac{\overbrace{\Pr(E|H)}^{Likelihood} \times \overbrace{\Pr(H)}^{Prior}}{\underbrace{\Pr(E)}_{\text{Partition function}}}. \tag{2.5}$$

To present IRL in this formulation, BIRL postulates the reward function as the hypothesis explaining the observed behavior of the expert (which is the evidence). Therefore, the posterior estimates the

probability that the learned reward function is true, given the expert's trajectories. BIRL treats the reward function as a random variable and utilizes a prior distribution over the reward function, given as

$$\Pr(R_{\boldsymbol{\theta}}) \;=\; \prod\nolimits_{s\in S, a\epsilon A} \Pr(R_{\boldsymbol{\theta}}(s,a)). \tag{2.6}$$

Notice that the reward values for the state-action pairs are i.i.d. Ramchandran and Amir [2007b] discuss some example prior distributions including the Gaussian [3]. We may derive the likelihood function for the demonstrated set of trajectories $\mathcal{X}$ as:

$$\Pr(\mathcal{X}|R_{\boldsymbol{\theta}}) = \prod_{X=1}^{|\mathcal{X}|}\prod_{t=1}^{\mathcal{T}} \Pr(s_X^t, a_X^t; R_{\boldsymbol{\theta}}) = \prod_{X=1}^{|\mathcal{X}|} \Pr(s_X^1)\Pr(a_X^1|s_X^1; R_{\boldsymbol{\theta}}) \prod_{t=1}^{\mathcal{T}-1} \Pr(s_X^{t+1}|s_X^t, a_X^t)\times \Pr(a_X^{t+1}|s_X^{t+1}; R_{\boldsymbol{\theta}}).$$

We may rewrite this as,

$$\Pr(\mathcal{X}|R_{\boldsymbol{\theta}}) \;=\; \prod\nolimits_{X=1}^{|\mathcal{X}|} \Pr(s_X^1)\,\pi(a_X^1|s_X^1; R_{\boldsymbol{\theta}}) \times \prod\nolimits_{t=1}^{\mathcal{T}-1}\; T(s_X^t, a_X^t, s_X^{t+1})\,\pi(a_X^{t+1}|s_X^{t+1}; R_{\boldsymbol{\theta}}). \tag{2.7}$$

The policy is commonly modeled in BIRL as a Boltzmann energy function (Ramachandran & Amir, 2007b; Vroman, 2014) of the form:

$$\pi_{\boldsymbol{\theta}}(a|s) \;=\; \frac{e^{\beta Q_{\boldsymbol{\theta}}(s,a)}}{\sum_{a'\in A} e^{\beta Q_{\boldsymbol{\theta}}(s,a')}} = \frac{e^{\beta Q_{\boldsymbol{\theta}}(s,a)}}{\Xi(s)}. \tag{2.8}$$

where $\Xi(s)$ is the partition function. As the Boltzmann temperature parameter β becomes large, exploration assigns increasing probability to the action(s) with the largest Q-value(s). One possible assignment to β could be between $0-1$ with 0 being fully exploratory and 1 being fully greedy.
The Q function is given as:

$$Q_{\boldsymbol{\theta}}(s,a) = R_{\boldsymbol{\theta}}(s,a) + \gamma \sum_{s'\in S} T(s,a,s') \times \sum_{a'\in A} Q_{\boldsymbol{\theta}}(s',a')\pi_{\boldsymbol{\theta}}(s',a'). \tag{2.9}$$

Methods for both maximum likelihood (Jain et al., 19; Vroman, 2014) and maximum-a-posteriori (Choi & Kim, 2011a) inferences of the reward function exist, which use the likelihood function of Eq. 2.7 and, in case of MAP inference, the prior as well [4]. MAP inference for IRL has been shown to be more accurate, benefiting from its use of the prior (Choi & Kim, 2011a). Formally, we may write MAP inference in

[3] Note that Gaussian distribution is one of the most commonly used priors in Bayesian probability theory because its conjugate prior is also a Gaussian distribution, which makes it easy to estimate.

[4] We consider the prior to be a Gaussian distribution in our case.

unnormalized log form [5] as:

$$R^*_{\boldsymbol{\theta}} = \arg\max_{\mathcal{R}} \underbrace{L_{\boldsymbol{\theta}}}_{\text{log-posterior}} = \arg\max_{\mathcal{R}} \underbrace{L^{lh}_{\boldsymbol{\theta}}}_{\text{log-likelihood}} + \underbrace{L^{pr}_{\boldsymbol{\theta}}}_{\text{log-prior}}. \tag{2.10}$$

where $\mathcal{R}$ is the continuous space of reward functions, and

$$L_{\boldsymbol{\theta}} = \log \Pr(R_{\boldsymbol{\theta}}|\mathcal{X}); \quad L^{lh}_{\boldsymbol{\theta}} = \log \Pr(\mathcal{X}|R_{\boldsymbol{\theta}}); \quad L^{pr}_{\boldsymbol{\theta}} = \log \Pr(R_{\boldsymbol{\theta}}). \tag{2.11}$$

Consequently, the partial differential of Eqn 2.10 becomes:

$$\frac{\partial L_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = \frac{\partial L^{lh}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + \frac{\partial L^{pr}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}.$$

The partial differential of the log-likelihood and log-prior are given as:

$$\frac{\partial L^{lh}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = \sum_{x \in \mathcal{X}} \sum_{t=1}^{\mathcal{T}} \frac{1}{\pi_{\boldsymbol{\theta}}(s^t, a^t)} \frac{\partial \pi_{\boldsymbol{\theta}}(s^t, a^t)}{\partial \boldsymbol{\theta}}; \quad \frac{\partial L^{pr}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = \Big(\frac{-(\boldsymbol{\theta} - \mu_{\boldsymbol{\theta}})}{\sigma^2_{\boldsymbol{\theta}}} \Big).$$

Choi and Kim, 2011a presents a gradient-based approach to obtain R^*, which searches the reward optimality region H^π only. Given the expert's policy, Ng and Russell [2000] show that this region can be obtained as:

$$H^\pi \triangleq I - (I^A - \gamma T)(I - \gamma T^\pi)^{-1} E^\pi. \tag{2.12}$$

where I is the identity matrix, T is the transition matrix, E^π is an $|S| \times |S||A|$ matrix with the $(s, (s', a'))$ element being 1 if s = s' and $\pi(s') = a'$. I^A is an $|S||A| \times |S|$ matrix constructed by stacking the $|S| \times |S|$ identity matrices $|A|$ times. The reward update rule in the gradient ascent is given as,

$$R^{new}_{\boldsymbol{\theta}} \leftarrow R_{\boldsymbol{\theta}} + \delta_t \nabla_{\boldsymbol{\theta}} \Pr(R_{\boldsymbol{\theta}}|\mathcal{X}). \tag{2.13}$$

where δ_t is an appropriate step size (or the learning rate). As computing $\nabla_{\boldsymbol{\theta}} \Pr(R_{\boldsymbol{\theta}}|\mathcal{X})$ involves calculating an optimal policy, this may slow down the computations. By checking if the gradient lies within the new reward optimality region using the condition: $H^\pi \cdot R^{new}_{\boldsymbol{\theta}} \leq 0$, we can reuse the same gradient and reduce the computational time.

[5] Log form is preferred because it preserves function monotonicity and simplifies product terms into sum terms. This is a handy feature in optimization since computing differentials of product terms can get hairy.

2.4 Maximum Entropy IRL

The principle of maximum entropy originates from information theory and was proposed by E.T.Jaynes in 1957 when he put forth the idea that entropy in information theory and entropy in statistical mechanics is based on the same principle. The theory of maximum entropy argues that while there may be an infinite number of probability distributions that may satisfy a set of constraints, the one with the maximum entropy is the one that makes the least assumptions about the data, apart from what is required to satisfy the constraints. This makes the learned distribution maximally uncertain about the parts of data that it hasn't encountered before and hence is an encoding of all the provided constraint information and nothing else. As described by B. D. Ziebart et al., 2008, MaxEntIRL builds a probability distribution over all possible expert trajectories as:

$$\max_{\Delta}\Big(\underbrace{-\sum_{X\in\mathcal{X}}\Pr(X)\,log\ \Pr(X)}_{\textit{H(A|S) - Shannon entropy}}\Big)\ \text{ subject to }\ \begin{aligned}&\sum_{X\in\mathcal{X}}\Pr(X)=1,\\&\sum_{X\in\mathcal{X}}\Pr(X)\sum_{\langle s,a\rangle\in X}\phi_k(s,a)=\hat{\phi}_k\quad\forall k.\end{aligned} \tag{2.14}$$

Here, $\mathcal{X}$ is the set of expert trajectories, $H(A|S)$ denotes the conditional entropy distribution of the policy, Δ is the space of all distributions $\Pr(X)$. This is also known as maximizing Shannon entropy (Zellner & Highfield, 1988). In order to simplify this, we apply Lagrangian relaxation to Eq 2.14 which is the *primal-problem*, to form an objective function that subsumes both the constraints, and then we solve the Lagrangian dual [6]. The new objective function becomes:

$$\begin{aligned}\mathcal{L}(Pr,\boldsymbol{\theta},\eta)\ \ =&-\sum_{X\in\mathcal{X}}\Pr(X)\,log\Pr(X)+\sum_k\theta_k\Big(\sum_{X\in\mathcal{X}}\Pr(X)\sum_{\langle s,a\rangle\in X}\phi_k(s,a)-\hat{\phi}_k\Big)\\&+\eta\Big(\sum_{X\in\mathcal{X}}\Pr(X)-1\Big).\end{aligned} \tag{2.15}$$

Where $\boldsymbol{\theta}$ and η are the Lagrange multipliers and since Eq. 2.15 is convex, taking the partial derivative of $\mathcal{L}$ with respect to $\Pr(X)$ and setting it to zero gives us the optimum:

$$\begin{aligned}&\frac{\partial\mathcal{L}}{\partial\Pr(X)}=-log\ \Pr(X)-1+\sum_k\theta_k\sum_{\langle s,a\rangle\in X}\phi_k(s,a)+\eta=0;\\&\text{where,}\quad\Pr(X)=\frac{e^{\sum_k\theta_k\sum_{\langle s,a\rangle\in X}\phi_k(s,a)}}{\Xi(\boldsymbol{\theta})}.\end{aligned} \tag{2.16}$$

$\Xi(\boldsymbol{\theta})$ is the normalization constant $e^{\eta}\cdot e^{-1}$; where η may be obtained using the Karush-Kuhn-Tucker (KKT) conditions (Gordon & Tibshirani, 2012). Substituting $\Pr(X)$ from Eq. 2.16 into the Lagrangian Eq. 2.15), we arrive at the dual $\mathcal{L}^{\text{dual}}(\boldsymbol{\theta})$ which is concave. We now have:

$$\mathcal{L}^{\text{dual}}(\boldsymbol{\theta})=log\,\Xi(\boldsymbol{\theta})-\sum_k\theta_k\hat{\phi}_k.$$

[6] For a background on Lagrangian Constrained Optimization, refer to Section 3.1.4.

With its gradient being,

$$\nabla \mathcal{L}^{\text{dual}}(\boldsymbol{\theta}) \;=\; \sum\nolimits_{X \in \mathcal{X}} \Pr(X) \sum_{\langle s,a \rangle \in X} \phi_k(s,a) - \hat{\phi}_k.$$

This method is built on the hypothesis that the expert follows a particular trajectory with a probability proportional to the reward accrued along it. While this is valid in a deterministic setting, in a stochastic MDP, an approximation is considered (Bogert, 2016).

$$\Pr(Y) \approx \frac{\prod_{<s,a,s'> \in Y} T(s,a,s') e^{\sum_k \theta_k \sum_{\langle s,a \rangle \in Y} \phi_k(s,a)}}{\Xi(\boldsymbol{\theta})}.$$

2.4.1 Maximum Causal Entropy

In order to avoid the bias that maximizing Shannon entropy introduces towards actions with uncertain (and possibly) risky outcomes, maximum causal entropy must be adopted in stochastic MDPs. Causal entropy is the sum of the entropies of the policy action selected conditioned on the state at that timestep, $H(A_{0:T-1}||S_{0:T-1}) = \sum_{t=0}^{T-1} \gamma^t H(A_t|S_t)$ (Gleave & Toyer, 2022). This has the useful property that it conditions only on the information available to the agent until the current timestep, namely, the current state, as well as the prior states and actions. Conversely, conventional Shannon entropy computes the entropy over the entire trajectory distribution, introducing an unwanted dependency on the transition dynamics (B. D. Ziebart et al., 2010). Formally, we can derive this as:

$$\begin{aligned}
H(S_{0:T-1}, A_{0:T-1}) &= \sum_{t=0}^{T-1} \gamma^t H(S_t, A_t | S_{0:t-1}, A_{0:t-1}) && \textit{chain rule} \\
&= \sum_{t=0}^{T-1} \gamma^t H(S_t | S_{0:t-1}, A_{0:t-1}) + H(A_t | S_{0:t-1}, A_{0:t-1}) && \textit{chain rule} \\
&= \sum_{t=0}^{T-1} \gamma^t H(S_t | S_{t-1}, A_{t-1}) + H(A_t | S_t) && \textit{independence} \\
&= \sum_{t=0}^{T-1} \underbrace{\gamma^t H(S_t | S_{t-1}, A_{t-1})}_{\textit{state transition entropy}} + \underbrace{H(A_{0:T-1}||S_{0:T-1})}_{\textit{causal entropy}} && \textit{causal ent.}
\end{aligned}$$

2.5 Adversarial Inverse Reinforcement Learning

The idea of adversarial training using two competing neural networks was first popularized by Goodfellow et.al, in 2014. This was given the moniker generative adversarial networks (GANs) and describes two neural networks (often deep networks) called the generator and the discriminator that compete against each other during training; upon convergence, the optimal generator accurately depicts the probability distribution

of the training data that it received. The generator network learns to generate samples by attempting to fool the discriminator network into believing its samples are real data. Both estimation procedures use the same function to drive learning, and sans a delicate balance between them, the learning is thrown off course. Drawing inspiration from GANs, generative adversarial imitation learning (GAIL) architecture was proposed by Ho and Ermon, 2016.

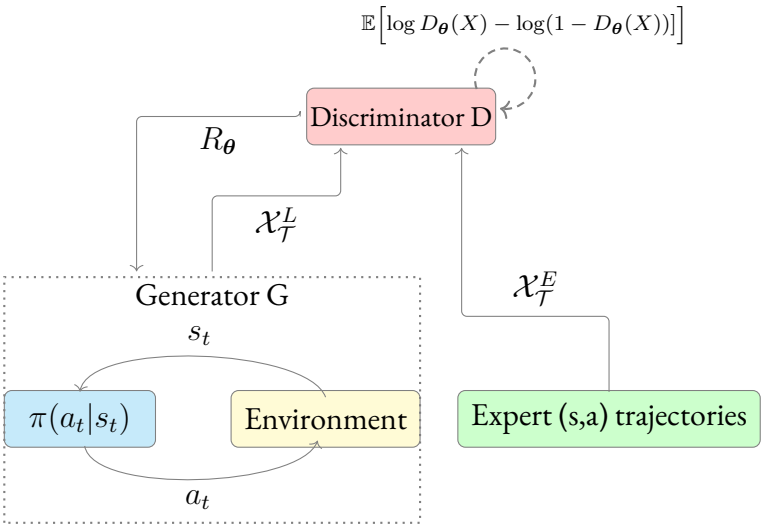


Figure 2.2: Architecture of adversarial inverse reinforcement learning (AIRL). Discriminator D receives expert demos and sampled trajectories, using which it minimizes the reverse KL divergence between expert and learned state-action marginal distributions to learn a function $f_{\boldsymbol{\theta}}$. G learns policy using $f_{\boldsymbol{\theta}} - \log(\pi)$ as the reward. We use the dotted arrow at D to denote the training loop with the objective function denoted over the dotted arrow. The definitions of $\mathcal{X}_{\mathcal{T}}^{E}$ and $\mathcal{X}_{\mathcal{T}}^{L}$ are the same as Fig. 2.1b.

GAIL by definition being an imitation learning technique learns to mimic the expert by arriving at a policy distribution that accurately represents the distribution of the input expert demonstrations. As discussed in Section 2.2, blindly mimicking the expert's behavior doesn't work well in environments with high uncertainty. Therefore, Fu et.al formulated Adversarial Inverse Reinforcement Learning (AIRL) in 2018 by extending Chelsea Finn's work - Guided Cost Learning (Finn, Levine, & Abbeel, 2016). AIRL builds on the maximum causal entropy IRL framework (B. Ziebart, 2010), based on an entropy-regularized MDP. While AIRL shares similarities with GAIL (Ho & Ermon, 2016), GAIL does not analytically solve for the reward function and instead uses $-log(1 - D_{\boldsymbol{\theta}})$ (where $D_{\boldsymbol{\theta}}$ is the discriminator at the current epoch), as the reward function (Orsini et al., 2021).

In most imitation learning and inverse RL methods, the stochastic policy is represented using a Boltzmann energy distribution, where the parameters of the distribution are optimized to maximize the likelihood of the observed data points, thus arriving at a policy distribution identical to the expert's. Alternatively, a simple approach can also be taken to this problem by minimizing the classification error of the output policy from the generator and the observed data, without learning a discriminator or energy function. However, when the generative model does not have the capacity to represent the entire data distribution sufficiently, maximizing likelihood directly without an energy function, will lead to a

moment-matching distribution that tries to "cover" all of the modes, leading to a solution that puts much of its mass in parts of the space that have negligible probability under the true distribution.

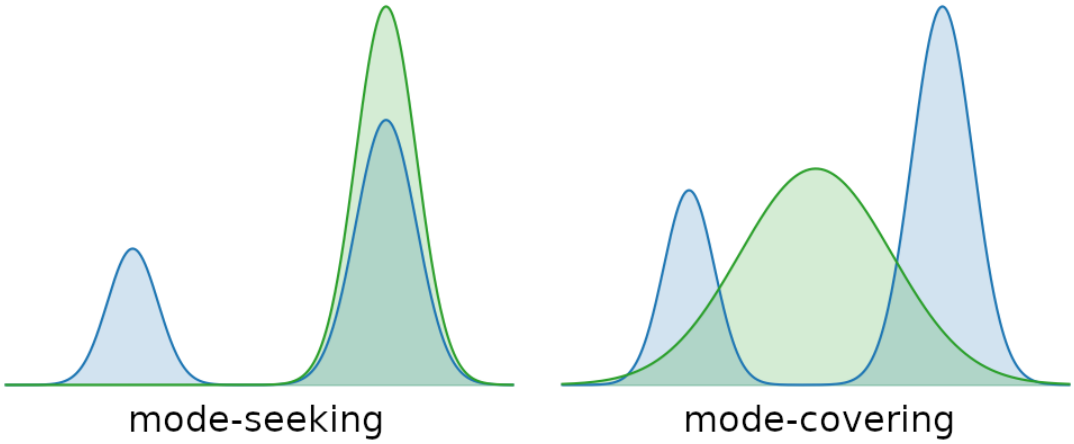


Figure 2.3: Comparison between the distributions learned by minimizing reverse KL divergence verses Jensen-Shannon divergence (Menéndez et al., 1997) or forward KL divergence. Notice that the latter induces a mean-seeking or mode-covering behavior, whereas the former induces a mode-seeking behavior. In many scenarios, it is preferable to produce only realistic, highly probable samples, by "filling in" as many modes as possible, at the trade-off of reduced variance (Finn, Christiano, et al., 2016). Image credit: Sander AI

Consequently, even if the generator trained through maximum likelihood has the same capacity as the one with an energy distribution, the latter exhibits mode-seeking behavior as long as the energy function is more flexible than the generator. Evidently, this phenomenon is often achieved at the cost of tractability, since generating samples from an energy function requires training a generator, which in the case of IRL, is forward policy optimization (Finn, Christiano, et al., 2016). While methods like behavior cloning (BC) that employ such maximum likelihood techniques to learn a behavior matching that of the demonstrating agent, these techniques prove particularly ineffective in complex scenarios due to compounding errors. In other words, when the policy erroneously deviates from the learned behavior, it encounters states other than what it was exposed to during training, making it more likely to continue making mistakes. This phenomenon is referred to as the covariate shift (Ghasemipour et al., 2020a).

Forward reinforcement learning aims to find the optimal policy π^* that maximizes the expected entropy-regularized discounted reward, under π, T, and ρ_0:

$$\pi^* = \arg\max_{\pi} E_{X\sim\pi}\Big[\sum_{t=0}^{\mathcal{T}} \gamma^t \Big(r(s_t, a_t) + H(\pi(.|s_t))\Big)\Big].$$

The distribution of the trajectories derived from the optimal policy $\pi^*(a|s)$ has been shown to take the form $\pi^*(a|s) \propto \exp Q^*_{soft}(s_t, a_t)$ in the MaxEntIRL formulation (Haarnoja et al., 2017; B. Ziebart,

2010), where:

$$Q^*_{soft}(s_t, a_t) = r_t(s, a) + E_{(s_{t+1},\ldots)\sim\pi}\underbrace{\Big[\sum_{t'=t}^{\mathcal{T}} \gamma^t \Big(r(s'_t, a'_t) + H(\pi(.|s'_t))\Big)\Big]}_{\text{Entropy regularized discounted reward}}.$$

In the context of AIRL, $\pi(a|s)$ is the probability of the adaptive sampler, and the goal is to minimize the reverse KL divergence between the trajectory distribution generated by the generator and that induced by the reward function (Alsaleh & Sayed, 2021). The structure of the discriminator is adapted from GCL (Finn, Levine, & Abbeel, 2016) where the discriminator $D_{\boldsymbol{\theta}}(X)$ learns a function $f_{\boldsymbol{\theta}}(X)$ (J. Fu et al., 2018) which at optimality is the expert policy's advantage function. The discriminator structure derived is very similar to a typical model for binary classification, with a sigmoid as the final layer. By subtracting $\log(\pi(X))$ from the input to the sigmoid $f_{\boldsymbol{\theta}}$, the discriminator can be made completely independent of the generator. This change is very simple to implement and is applicable in any setting where the density $\pi(X)$ can be cheaply evaluated (Finn, Christiano, et al., 2016).

$$\begin{aligned}
\mathcal{S}(p) &= \frac{1}{1+\exp\{-p\}} \quad \rightarrow \quad \text{Regular sigmoid function with input } p \\
\mathcal{S}(f_{\boldsymbol{\theta}} - \log(\pi(X))) &= \frac{1}{1+\exp\{-(f_{\boldsymbol{\theta}} - \log(\pi(X)))\}} \\
&= \frac{\exp\{f_{\boldsymbol{\theta}} - \log\pi(X)\}}{\exp\{f_{\boldsymbol{\theta}} - \log(\pi(X))\}+1} = \frac{\frac{\exp\{f_{\boldsymbol{\theta}}\}}{\exp\{\log\pi(X)\}}}{\frac{\exp\{f_{\boldsymbol{\theta}}\}}{\exp\{\log\pi(X)\}}+1} \\
&= \frac{\frac{\exp\{f_{\boldsymbol{\theta}}\}}{\cancel{\exp\{\log\pi(X)\}}}}{\frac{\exp\{f_{\boldsymbol{\theta}}\}+\exp\{\log\pi(X)\}}{\cancel{\exp\{\log\pi(X)\}}}} = \frac{\exp\{f_{\boldsymbol{\theta}}\}}{\exp\{f_{\boldsymbol{\theta}}\}+\pi(X)}.
\end{aligned}$$

The learned softmax policy is given as,

$$\pi(X) = exp\Big(A^{soft}_{f_{\boldsymbol{\theta}}}(s,a)\Big) = exp\Big(Q^{soft}_{f_{\boldsymbol{\theta}}}(s,a) - V^{soft}_{f_{\boldsymbol{\theta}}}(s)\Big) \tag{2.17}$$

The soft advantage function in Eqn 2.17 (Gleave & Toyer, 2022) is computed through the forward rollout before the next discriminator update and used as:

$$D_{\boldsymbol{\theta}}(X) = \frac{\exp\{f_{\boldsymbol{\theta}}(X)\}}{\exp\{f_{\boldsymbol{\theta}}(X)\}+\pi(X)} \tag{2.18}$$

The goal of the discriminator is to minimize the binary cross-entropy loss (Mao et al., 2023) between the expert and learned distributions as:

$$L(\theta) = \sum_{t=0}^{T}\left(-\mathop{\mathbb{E}}_{(s_t,a_t)\sim X^t_E}[\log D_\theta(s_t,a_t)] - \mathop{\mathbb{E}}_{(s_t,a_t)\sim X^t_L}[\log(1-D_\theta(s_t,a_t))]\right).$$

Therefore the reward function can be denoted as (we drop the subscripts here to avoid clutter),

$$R_{\boldsymbol{\theta}}(X) \leftarrow \log D_{\boldsymbol{\theta}}(X) - \log(1 - D_{\boldsymbol{\theta}}(X)). \tag{2.19}$$

Substituting the value of $D_{\boldsymbol{\theta}}$ from Eqn 2.18 into Eqn 2.19, we get,

$$\begin{aligned}
R_{\boldsymbol{\theta}}(X) &= \log\Big(\frac{\exp\{f_{\boldsymbol{\theta}}(X)\}}{\exp\{f_{\boldsymbol{\theta}}(X)\} + \pi(X)}\Big) - \log\Big(1 - \frac{\exp\{f_{\boldsymbol{\theta}}(X)\}}{\exp\{f_{\boldsymbol{\theta}}(X)\} + \pi(X)}\Big) \\
&= \Big(\log\exp\{f_{\boldsymbol{\theta}}(X)\} - \log\Big(\exp\{f_{\boldsymbol{\theta}}(X)\} + \pi(X)\Big)\Big) \\
&\qquad\qquad - \log\Big(\frac{\cancel{\exp\{f_{\boldsymbol{\theta}}(X)\}} + \pi(X) - \cancel{\exp\{f_{\boldsymbol{\theta}}(X)\}}}{\exp\{f_{\boldsymbol{\theta}}(X)\} + \pi(X)}\Big) \\
&= \Big(f_{\boldsymbol{\theta}}(X) - \log(\exp\{f_{\boldsymbol{\theta}}(X)\} + \pi(X))\Big) - \log\Big(\frac{\pi(X)}{\exp\{f_{\boldsymbol{\theta}}(X)\} + \pi(X)}\Big) \\
&= f_{\boldsymbol{\theta}}(X) - \cancel{\log\Big(\exp\{f_{\boldsymbol{\theta}}(X)\} + \pi(X)\Big)} - \log(\pi(X)) + \cancel{\log\Big(\exp\{f_{\boldsymbol{\theta}}(X)\} - \log(\pi(X))\Big)} \\
&= f_{\boldsymbol{\theta}}(X) - \log(\pi(X)). \qquad \rightarrow \qquad \text{Entropy regularized reward.}
\end{aligned} \tag{2.20}$$

In contrast to GCL's formulation, AIRL's discriminator does not use a partition function ($1/Z$) for the $\exp\{f_{\boldsymbol{\theta}}(X)\}$ term because for a given state, $Z(s)$ remains constant and is assumed to be implicitly learned as part of $f_{\boldsymbol{\theta}}(X)$, although it cannot be extracted [7]. AIRL aims to minimize the *reverse KL divergence* between the learner's and expert's marginal state-action distribution $KL(\rho_\pi(s,a)||\rho_{exp}(s,a))$ since this results in mode-seeking behavior in contrast to BC methods that use forward KL divergence with a mode-covering behavior (Ghasemipour et al., 2020b).

$$KL(\rho_\pi(s,a)||\rho_{exp}(s,a)) = \sum_{(s,a)\in X}\sum_{t=0}^{T-1} \Pr_{\pi}(s^t,a^t)\Big[\log(\Pr_{\pi}(s^t,a^t)) - \log(\Pr_{exp}(s^t,a^t))\Big]. \tag{2.21}$$

One key difference between the original formulation by J. Fu et al., 2018 and our approach is that we model the reward function as a function of both states and actions, since the disentangled reward learning comes with the following strong assumptions:

1. The MDP modelling the environment is ergodic. In other words, every state gets visited eventually.
2. The MDP has deterministic environment transitions (Geng et al., 2020; Venuto, 2020).
3. The deterministic transition dynamics satisfies a strong requirement known as the decomposability condition (Gleave & Toyer, 2022). In other words, all pairs of states in the MDP are linked.

[7] (J. Fu et al., 2018) AIRL OpenReview comment.

2.6 Proximal Policy Optimization

IRL consists within it, a step of forward rollout which in tabular methods is typically a straightforward value iteration or policy iteration that converges to a global optimum. However, in model-free techniques, we use RL methods like Q-learning, policy-gradient, soft actor-critic (SAC), trust region policy optimization (TRPO), etc to obtain the updated policy and the value-function. In this list of popular RL methods is proximal policy optimization (PPO), which is the forward rollout method used in the original AIRL (J. Fu et al., 2018) framework. PPO was first proposed by Schulman et al., 2017, wherein they introduced a new policy-gradient method that alternates between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. This technique improves upon trust-region policy optimization (TRPO) by using a lower-bound on the unclipped objective. More specifically, TRPO maximizes the policy objective:

$$L = E_t[\lambda^t A^t_{\pi_\omega}]; \quad \text{where} \quad \lambda^t = \frac{\pi_\omega(a^t|s^t)}{\pi^{old}_\omega(a^t|s^t)}.$$

Where A^t is the advantage function at the current timestep and π^{old}_ω is the policy from the previous iteration and ω are the weights for the policy. However, since this leads to an excessively large policy update (Schulman et al., 2017), PPO proposes the following objective function:

$$L^{CLIP} = E_t\Big[min\Big(\lambda^t\, A^t_{\pi_\omega}, clip(\lambda^t, 1-\epsilon, 1+\epsilon)A^t_{\pi_\omega}\Big)\Big].$$

where L^{CLIP} is the clipped surrogate objective that takes the minimum of the clipped and unclipped objective to maintain a pessimistic bound over the final objective. Clipping the importance sampling ratio removes the incentive for moving λ^t outside of the interval $[1-\epsilon, 1+\epsilon]$. $A^t_{\pi_\omega}$ is calculated with respect to the reward estimates from Eqn 2.20. Refer to Section 3.2 for a comparative analysis of PPO with other commonly used RL techniques and some popular libraries that implement them in a stable manner.

Chapter 3

Technical background - 2

So far in Chapter 2 we discussed the most popular IRL techniques, some common decision-making models adopted for the same, and AIRL's forward rollout technique - PPO. In this chapter, we look at some pivotal concepts that play a crucial role in realizing these RL and IRL techniques in the real world. We will also compare some of the popular RL techniques briefly and talk about why we chose PPO over the other RL methods. Overall, this chapter aims to be a succinct summary of the knowledge required to get AIRL and PPO implemented in pragmatic scenarios.

3.1 Key concepts in pragmatic RL and IRL

In this section, we explain some fundamental concepts that play into almost all recent RL and IRL algorithms. These techniques play an integral role in the quality of learning and are worth understanding in detail before we advance to our specific research.

3.1.1 Monte-Carlo vs Temporal Difference learning

In this section we briefly compare the most common techniques used to update the value function in reinforcement learning settings.

Monte-Carlo Learning:

The goal of forward RL is to learn a policy π that maximizes the expectation of discounted, long-term reward (also known as the state value function).

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s, \pi\right] = \mathbb{E}\left[G_t \mid s_0 = s, \pi\right].$$

However, when the dynamics of the environment are unknown, one most common technique to estimate the value function, is by sampling the environment repeatedly and using the returned rewards

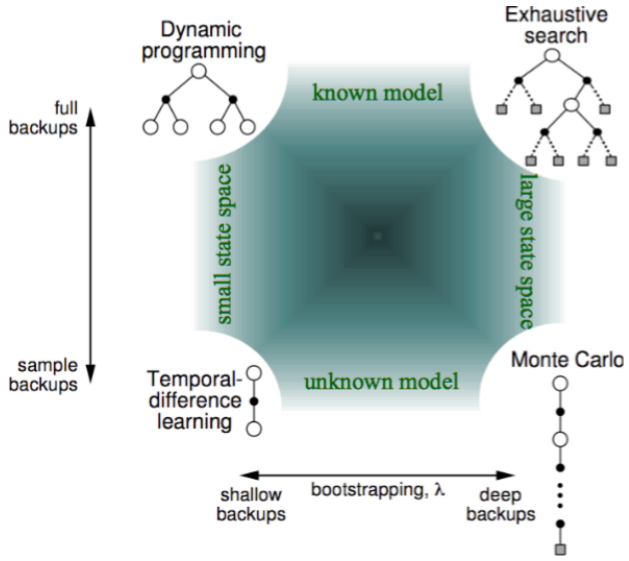


Figure 3.1: Figure showing the spectrum of classical RL approaches based on the width and depth of bootstrap backups. Image credit: Sutton and Barto, 2018.

to estimate the value.

$$\underbrace{V(S_t)}_{\text{Updated value of state } S_t} \leftarrow \underbrace{V(S_t)}_{\text{Former estimate of state } s_t} + \underbrace{\alpha}_{\text{Learning rate}} \Big[\underbrace{G_t}_{\text{Return at timestep } t} - \underbrace{V(S_t)}_{\text{Current value of state} S_t} \Big].$$

Monte Carlo learning techniques use the simple idea of equating empirical mean return to value (Face, n.d.). However, this requires complete sampled trajectories and their returns. MC methods usually have high variance but are unbiased estimators of the true value function. Also, since they wait until the end of the episode to update, they are slow to converge.

Temporal Difference Learning:

Temporal Difference (TD) learning, is the concept of using the difference in expected returns between timesteps to update the value function. The most basic version TD-0, only waits one time step before making an update (Face, n.d.).

TD-0: updates after every step, and hence doesn't have the expected return G_t. Instead, it uses R_{t+1} and the discounted value of the next state to update the value of the current state as shown below:

$$\underbrace{V(S_t)}_{\text{Updated value of state } S_t} \leftarrow \underbrace{V(S_t)}_{\text{Former estimate of state } S_t} + \underbrace{\alpha}_{\text{Learning rate}} \underbrace{\Big[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD target}} - \underbrace{V(S_t)}_{\text{Current value of state } S_t} \Big]}_{\text{TD Error}}.$$

This is called *bootstrapping*. It's called this because TD bases its update in part on an existing estimate $V(S_{t+1})$ and not a complete sample G_t. By extension, the n-step TD update can be given as:

$$\underbrace{V(S_t)}_{\text{Updated value of state } S_t} \leftarrow \underbrace{V(S_t)}_{\text{Former estimate of state } s_t} + \underbrace{\alpha}_{\text{Learning rate}} \Big[\underbrace{G_t^{(n)}}_{\text{Return at timestep } t} - \underbrace{V(S_t)}_{\text{Current value of state} S_t} \Big].$$

However, picking the right n is difficult, therefore, we use an intuitive technique to circumvent the

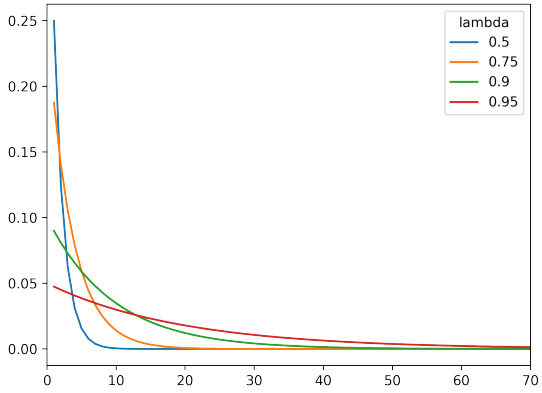


Figure 3.2: Graph showing the impact of different λ values on the initial return and how it decays over time. Image credit: Reis, n.d.

problem of picking an n value by picking all possible n values at once. This can be done by weighting the n-step return G_t^n by a weight $\lambda \in [0, 1]$ that decays exponentially over time. The nth step is weighted by λ^{n-1} (Reis, n.d.). λ in the context of TD learning is called the "trace decay parameter" and it controls how much past traces contribute to the current trace. A value close to 0 makes the traces short-term, focusing on recent events, while a value close to 1 makes the traces long-term, considering events further back in time. In other words, λ determines the trade-off between bootstrapping and sampling.

Since we want all of these weights to sum to one (to have a weighted average), we need to normalize them. The normalization constant is easy to derive:

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \sum_{n=0}^{\infty} \lambda^n = \frac{1}{1-\lambda}.$$

Therefore, multiplying each term by $(1 - \lambda)$ gives us the normalized weights. Now, the λ-return can be written as:

$$G_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}.$$

The problem with this is the same as that with the Monte Carlo update. We need to wait for a trajectory to complete to compute every possible n-step return and combine them to obtain the λ-weighted average. This flavor of TD-λ is called the *forward view* (Silver, n.d.).

Eligibility Traces: In the backward view of TD-λ, we use the concept of eligibility traces $e_t(s, a)$ to remember what happened in the past and use current information to update the state-values for every state encountered so far. This is especially useful in cases where the reward is sparse or delayed. Eligibility traces help in assigning credit or blame to states and actions based on their temporal proximity to rewards. They enable the algorithm to remember and track which states and actions occurred recently and adjust their associated values accordingly.

$$\mathbf{e}_t(s,a) = \begin{cases} \gamma\lambda \mathbf{e}_{t-1}(s,a) + 1 & \text{if } (s,a) = (s_t, a_t) \\ \gamma\lambda \mathbf{e}_{t-1}(s,a) & \text{otherwise.} \end{cases}$$

This type of eligibility trace is known as the "accumulating eligibility trace". Alternatively, if we reset a state's eligibility value to 1 every time it is visited instead of adding 1 to it, it is known as the "replacing eligibility trace".

TD-λ Backward View: uses a λ-return target for bootstrapping. A λ-return target takes all n-step targets, and weights each step by λ^{n-1}.

$$\underbrace{V(S_t)}_{\text{Updated value of state } S_t} \leftarrow \underbrace{V(S_t)}_{\text{Former estimate of } S_t} + \underbrace{\alpha}_{\text{Learning rate}} \cdot \underbrace{\delta_t}_{\text{TD Error}} \cdot \underbrace{e_t}_{\text{Eligibility trace}} \tag{3.1}$$

where $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$.

We now use the eligibility value as a scaling factor for a TD(0) error. However, notice that every state is updated at once during every update. By propagating the current error information into the states visited in the past, we combine the n-step returns in an online manner.

3.1.2 Generalized Advantage Estimate

Before we delve into generalized advantage estimate (GAE) (Schulman, Moritz, et al., 2015), let us get some background knowledge to understand how and why GAE improves learning performance.

Advantage Function

The advantage function ($A(s, a)$) measures how much better an action a is compared to others in a given state s (viz., the relative advantage of that action). Mathematically, it is defined as the difference between the state value function ($V(s)$) and the action-value function ($Q(s, a)$):

$$A(s,a) = Q(s,a) - V(s).$$

Bias-variance tradeoff: in machine learning refers to the balance between two types of errors: bias and variance. Bias manifests in oversimplified models that fail to capture the underlying complexities of the data, leading to underfitting. Variance is seen in overly complex models that capture even the noise in the training data as a meaningful signal, resulting in overfitting. Achieving low bias and low variance simultaneously is challenging since reducing one often increases the other.

Generalized Advantage Estimate

While the advantage function is valuable, accurately estimating it can be challenging, especially in environments with long time horizons or high variance in rewards. GAE aims to address these challenges by providing a more stable and efficient estimation of the advantage function. GAE introduces a parameter (λ^{GAE}) that balances the trade-off between bias and variance in estimating the advantage function. This parameter controls how much credit to assign to future states when computing advantages.

$$A_t^{\text{GAE}} = \sum_{\tau=0}^{\infty} (\gamma\lambda^{GAE})^{\tau} \delta_{t+\tau}.$$

where $\delta_{t+\tau}$ represents the advantage at timestep $t+\tau$, t is the timestep at which A is being computed and τ denotes the number of timesteps. Usually λ^{GAE} is chosen based on domain knowledge, empirical experience, heuristics or through theoretical analysis. A common value chosen for λ^{GAE} is between $0.95 - 0.98$. Note that both TD-λ-backward-view and GAE leverage λ to adjust the temporal credit assignment, influencing how credit or blame is propagated backward in time. However, λ in the latter balances generalization versus accurate advantage estimation, while λ in the former affects the trade-off between bootstrapping and sampling in value function updates.

3.1.3 Importance Sampling

Sampling is widely used in statistics and machine learning to estimate probability distributions that cannot be perfectly captured. Sampling techniques typically draw samples from a target distribution and use approximation techniques to create nearly-identical distributions. While several popular sampling techniques exist like Monte-Carlo, Rejection Sampling, Markov-Chain Monte Carlo, etc exist, one of the most commonly used sampling techniques in RL and IRL is Importance Sampling (Glynn & Iglehart, 1989).

The core idea behind Importance Sampling is to use samples drawn from a known distribution (often termed the "importance distribution" or "proposal distribution") to estimate properties of the target distribution. Instead of directly drawing samples from the target distribution, which might be infeasible, Importance Sampling reweights these samples based on the ratio of the target distribution to the importance distribution. Let us denote the target distribution as P, the importance distribution as Q, and any ith sample drawn from the importance distribution as x_i. The key is that the support of Q should cover the support of P, meaning that Q should assign non-zero probability to regions where P also has non-zero probability.

For each sample drawn from the importance distribution, we compute its weight, which is the ratio of the target distribution's probability density function (pdf) to the importance distribution's pdf at that point. Mathematically, the weight for a sample x_i is given by:

$$w_i = \frac{P(x_i)}{Q(x_i)}.$$

Now, we use these weights to compute estimates of the properties of interest for the target distribution. For example, to estimate the expected value (mean) of a function $f(x)$ with respect to the target distribution P, we compute:

$$\mathbb{E}_P[f(x)] \approx \frac{\sum_{i=1}^{N} w_i \, f(x_i)}{\sum_{i=1}^{N} w_i}.$$

where N is the number of samples drawn from the importance distribution. Estimating the mean of $f(x)$ with respect to P provides a summary statistic that describes the central tendency of the distribution or function $f(x)$ under consideration (Wikipedia, 2024b).

Note that $f(x)$ could be any arbitrary function of x, such as: the pdf of the target distribution P. Estimating the mean of this function with respect to P essentially means estimating the expected value of the distribution itself. It could be a function of a random variable whose expectation you are interested in calculating. For example, the average height of individuals in a population (where x represents height), $f(x)$ could be the height of an individual, and estimating the mean of $f(x)$ with respect to P would give you the average height. $f(x)$ could also represent some function of the variable x that you are interested in estimating, such as a cost function, utility function, or any other measure of interest.

3.1.4 Lagrangian Method for Constrained Optimization:

Lagrangian method or the Lagrangian Multiplier method (Lagrange, 1853), is a popular technique used to solve constrained optimization problems. Error correction through optimization is ubiquitous in Science and Engineering. Optimization usually entails finding the (global) maxima or minima of a function by iteratively moving towards the point where the gradient of the function with respect to the optimizing variable becomes zero. A constrained optimization problem is one that may have two (common) types of constraints - regional and functional (Courcoubetis & Weber, 2003). For example, consider this generic objective function:

$$L : \underset{x \in X}{maximize} \;\; f(x) \text{ subject to } g(x) = b.$$

Here, $x \in X$ is a regional constraint that says the value of x must lie within the space of X. For example, X may be $\mathbb{N}$ - the space of natural numbers. The second constraint $g(x) = b$ is a functional constraint that says the max value obtained for $f(x)$ must also satisfy a secondary condition $g(x) = b$. Let us pick

another simple example (D. P. Bertsekas, 2014):

$$maximize \;\; z = f(x, y) \text{ subject to } g : x + y \leq 100.$$

Using a Lagrange multiplier λ, this equation can be rewritten as:

$$L = f(x, y) + \lambda(100 - x - y)$$

Then we can follow the same steps as any other regular maximization problem:

$$\frac{\partial L}{\partial x} = f_x - \lambda = 0$$
$$\frac{\partial L}{\partial y} = f_y - \lambda = 0$$
$$\frac{\partial L}{\partial \lambda} = 100 - x - y = 0$$

Now we have three equations and three unknowns, which is straightforward to solve. Similarly, the more constraints a problem has, the more number of Lagrangian variables can be used to rewrite it, such that the essence of the optimization is retained. Intuitively, we are trying to find the value for λ such that $\nabla f = \lambda \nabla g$.

3.1.5 Kullback-Leibler (KL) Divergence:

Kullback-Leibler (KL) divergence (Kullback & Leibler, 1951), proposed by Solomon Kullback and Richard Liebler in 1951, defines a measure of how one probability distribution (P) is different from another, expected probability distribution (Q). It is closely related to relative entropy and is an intuitive way to effectively measure the average likelihood of observing (infinite) data with the distribution P if the particular model Q actually generated the data (Shlens, 2014).

The KL divergence between two probability distributions P and Q is defined as:

$$D_{KL}(P\|Q) = \sum_{x \in \mathcal{X}} P(x) \log_2 \left(\frac{P(x)}{Q(x)} \right).$$

where $\mathcal{X}$ is the set of possible outcomes, $P(x)$ is the probability of observing outcome x according to the distribution P, and $Q(x)$ is the probability of observing outcome x according to the distribution Q.

One popular application of KL divergence is to compute mutual-information. For example, to estimate the similarity of a joint distribution $P(x, y)$ to the product of its marginals $P(x)P(y)$ — this is called mutual information, a general measure of statistical dependence between two random variables (Cover,

Thomas, et al., 1991).

$$I(X;Y) = \sum_{x,y} P(x,y) \log_2 \frac{P(x,y)}{P(x)\,P(y)}.$$

Some noteworthy points about KL divergence are:

- KL divergence is not symmetric, meaning $D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$.
- It's non-negative, meaning $D_{KL}(P\|Q) \geq 0$, with equality only when $P(x) = Q(x)$ for all x.
- KL divergence quantifies the "information loss" when using a probability distribution Q to approximate P.
- KL divergence is different from Jenson-Shannon (JS) distance (Menéndez et al., 1997) since JS is symmetric. JS distance is half the sum of the KL divergences between the two distributions and their average:

$$D_{JS}(P\|Q) = \frac{1}{2}\left(D_{KL}\left(P\|\frac{P+Q}{2}\right) + D_{KL}\left(Q\|\frac{P+Q}{2}\right)\right).$$

3.1.6 Wasserstein Metric:

Wasserstein metric (Vaserstein, 1969), also known as Earth Mover's Distance (EMD) or Kantorovich-Rubinstein distance, measures the minimum amount of work required to transform one probability distribution into another.

$$W_p(\mu,\nu) = \left(\inf_{\gamma\in\Gamma(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} d(x,y)^p \, d\gamma(x,y)\right)^{1/p}.$$

where μ and ν are probability measures, $\Gamma(\mu,\nu)$ is the set of all joint distributions with marginals μ and ν, $d(x,y)$ is a distance metric between points x and y, p is a parameter determining the order of the distance.

Applications: One of the most popular techniques that use this metric is called WGAN or Wasserstein GAN (Arjovsky et al., 2017). This metric has also been used in a recent RL concept called *Distributional RL* (Bellemare et al., 2017), where instead of learning the expected, discounted, long-term returns (or) the value function, the agent learns the entire distribution of returns. The Distributional Bellman equation for a given state-action pair (s,a) is:

$$Z(s,a) = R(s,a) + \gamma Z'(s',a').$$

where $Z(s,a)$ is the distribution of returns for state-action pair (s,a), $R(s,a)$ is the immediate reward received after taking action a in state s, γ is the discount factor, $Z'(s',a')$ is the distribution of returns

for the next state-action pair (s', a'). The distributional Bellman update, which updates the distribution of returns based on the Bellman equation, leads to a contraction in the Wasserstein metric space. This contraction property is desirable because it implies that repeated application of the Bellman operator converges the distribution of returns towards the true distribution, which improves the accuracy of the value estimates.

3.1.7 Regularization

Regularization (Poggio et al., 1987) is a technique used in machine learning to prevent overfitting and improve the generalization of learned models. In the context of inverse learning, regularization involves penalizing the objective function, to discourage large parameter updates that often lead to overly complex models. This penalty term helps balance the bias-variance trade-off between fitting the training data well and maintaining simplicity in the model.

The goal of regularization is to reduce the model's sensitivity to small fluctuations or noise in the training data, thereby improving its ability to generalize to unseen data. By constraining the model's complexity, regularization can prevent it from memorizing the training data and instead encourage it to capture underlying patterns and relationships that are more likely to hold true across different datasets (Wikipedia, 2024c).

Parameter Regularization

Parameter regularization techniques penalize the complexity of the policy or value function by adding a regularization term to the objective function. Common regularization techniques include L^1 and L^2 regularization, which penalize the magnitude of the parameters.

L^1 Regularization: adds a penalty term proportional to the absolute value of the parameters:

$$\text{Regularized Objective} = \text{Original Objective} + \lambda \sum_i |w_i|$$

where w_i are the parameters of the policy or value function, and λ is the regularization strength hyperparameter.

L^2 Regularization: adds a penalty term proportional to the square of the parameters:

$$\text{Regularized Objective} = \text{Original Objective} + \lambda \sum_i w_i^2$$

where w_i and λ have the same meaning as in L^1 regularization.

Entropy Regularization

Entropy regularization encourages exploration by adding the entropy of the policy to the objective function. This is especially helpful in cases of large state-action spaces, such as continuous domains.

$$\text{Regularized Objective} = \text{Original Objective} - \lambda \mathbb{E}[\text{Entropy}(\pi)]$$

where π is the policy, and λ is the regularization strength hyperparameter.

KL Divergence Regularization

KL divergence regularization encourages the learned policy to stay close to a old policy used as reference. It penalizes deviations from the reference policy by adding the KL divergence between the learned policy and the reference policy to the error function. This is the type of regularization used in both TRPO (Schulman, Levine, et al., 2015) and PPO (Schulman et al., 2017).

$$\text{Regularized Objective} = \text{Original Objective} + \lambda \, \text{KL}[\pi_{\text{new}} \parallel \pi_{\text{old}}].$$

where λ is the regularization strength hyperparameter.

3.1.8 Backpropagation and a common conundrum

Backpropagation (Rumelhart et al., 1986), arguably, the backbone of modern deep-learning methods, is a paradigm to efficiently compute gradients of the loss function with respect to the weights of the network. These gradients are then used to update the weights in order to minimize the loss function, which is the objective of learning. Backpropagation involves a forward pass - that uses the input data fed into the network to extract features, layer by layer, while being transformed by activation functions and weighted sums; loss computation - At the last layer, the network makes a prediction based on the weights assigned to the features in that layer. By comparing this prediction to the ground-truth, a loss value is computed; backpropagation - Using chain rule from calculus, starting from the output layer and moving backwards, the gradient of the loss function with respect to the output of each neuron is computed and multiplied with the gradient of the previous layer until the first layer is reached.

Now, assume our objective function L is a function of two variables θ and ϕ and we are interested in optimizing its expected value with respect to both parameters θ and ϕ:

$$L(\theta, \phi) = \mathbb{E}_{X \sim p_\phi(x)} \Big[f_\theta(X) \Big].$$

Since both the objective and its gradients are intractable in general, we estimate them using samples from $p_\phi(x)$. The gradient with respect to θ is straightforward:

$$\nabla_\theta L(\theta, \phi) = \nabla_\theta \mathbb{E}_{X \sim p_\phi(x)} \Big[f_\theta(X) \Big].$$

We can take the gradient ∇_θ into the expectation using the Leibniz integral rule (Amazigo & Rubenfeld, 1980) as follows:

$$\nabla_\theta L(\theta, \phi) = \mathbb{E}_{X \sim p_\phi(x)} \Big[\nabla_\theta f_\theta(X) \Big].$$

Now, L can be estimated using Monte Carlo sampling:

$$\nabla_\theta L(\theta, \phi) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta f_\theta(X^n) \;\; \text{where} \;\; X^n \sim p_\phi(x) \text{ i.i.d.}$$

However, when it comes to computing the gradient with respect to ϕ, the problem gets complicated because ∇_ϕ cannot be taken into the expectation since X sampled from $p_\phi(x)$ is also parameterized by ϕ:

$$\nabla_\phi L(\theta, \phi) = \nabla_\phi \mathbb{E}_{X \sim p_\phi(x)} \Big[f_\theta(X) \Big] \neq \mathbb{E}_{X \sim p_\phi(x)} \Big[\nabla_\phi f_\theta(X) \Big].$$

This is a common issue encountered in posterior computation in variational inference, value function and policy learning in reinforcement learning, derivative pricing in computational finance, and inventory control in operations research, amongst many others.

Score Function Estimators

This is one of the methods that can help sidestep the aforementioned predicament (M. C. Fu, 2006; Glynn, 1990; Williams, 1992). This uses a method called "log-derivative trick" (which says: $\nabla_\phi p_\phi(x) = p_\phi(x) \nabla_\phi \log p_\phi(x)$), to express the gradient as:

$$\nabla_\phi L(\theta, \phi) = \mathbb{E}_{X \sim p_\phi(x)} \Big[f_\theta(X) \nabla_\phi \log p_\phi(X) \Big].$$

Now, we can compute the expectation of the gradient using Monte Carlo estimate as:

$$\nabla_\phi L(\theta, \phi) \approx \frac{1}{N} \sum_{n=1}^{N} f_\theta(X^n) \nabla_\phi \log p_\phi(X^n) \;\; \text{where} \;\; X^n \sim p_\phi(x) \text{ i.i.d.}$$

The advantage of this method is that it does not require $f_\theta(x)$ to be differentiable or even continuous as a function of x, this method can be used with both discrete and continuous variables. The downside is that this basic version can suffer from high variance which would then require variance reduction techniques to obtain a usable estimate.

Reparameterization Trick

This method is the low variance alternative that simply samples from a fixed distribution $q(z)$ and transforms it using a function $g_\phi(z)$ to represent a sample obtained from $p_\phi(x)$. For example, by sampling from the standard Normal distribution $\mathcal{N}(0, 1)$ and transforming it using the location-scale transformation technique: $g_{\mu,\sigma} = \mu + \sigma Z$, we can get a sample from $\mathcal{N}(\mu, \sigma^2)$. This two-stage reformulation of the sampling process, called the reprameterization trick (Maddison et al., 2016), allows us to transfer the dependence on ϕ from p into f by writing $f_\theta(x) = f_\theta(g_\phi(z))$ for $x = g_\phi(z)$.

$$L(\theta, \phi) = \mathbb{E}_{X\sim p_\phi(x)}\Big[f_\theta(X)\Big] = \mathbb{E}_{Z\sim q_z(x)}\Big[f_\theta(g_\phi(Z))\Big].$$

Since $q(z)$ does not depend on ϕ, the gradient can now be estimated similar to the gradient with respect to θ.

$$\nabla_\phi L(\theta, \phi) = \mathbb{E}_{Z\sim q_z(x)}\Big[\nabla_\phi f_\theta(g_\phi(Z))\Big] = \mathbb{E}_{Z\sim q_z(x)}\Big[f'_\theta(g_\phi(Z))\nabla_\phi g_\phi(Z)\Big].$$

In the context of RL, specifically policy gradient techniques, this trick is used to sample from a known, deterministic distribution that can be transformed to obtain sampled actions to compute the gradient during backpropagation (Schulman, Heess, et al., 2015).

3.1.9 Target Networks

The difference between Q-learning and DQN (Mnih et al., 2015) is that the latter replaces an exact value function with a function approximator - a neural network. While Q-learning updates exactly one state/action value at each timestep, DQN updates many. This causes the action values for the very next state to drastically change from the previous timestep, causing drastic instability in learning. The effect is typically referred to as "catastrophic forgetting" in neural network literature (French, 1999). Using a stable, secondary target network as the error measure is one way of combating this effect. Conceptually, it is equivalent to waiting for a significant change in values before updating the network as opposed to updating it after every time step. By allowing the network more time to consider many recent actions, learning stability increases manifold.

3.2 Comparison of forward RL methods

For our application, we need a technique that can learn a decentralized policy using a centralized reward function. We also need a method that can work both for continuous and discrete action spaces and can converge well in both. As can be seen from Table 3.1 and Table 3.2, TRPO and PPO are the only two options among the most popular RL techniques that have all these features.

Algorithm	Objective Function
Q-learning	$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) - Q(s,a)]$
SARSA	$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
DQN (Deep Q Network)	$L(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2]$
A3C (Asynchronous Advantage Actor-Critic)	$L(\theta) = \sum_t \big(-\log(\pi(a_t\|s_t;\theta))(R_t - V(s_t;\theta_v)) + \beta H(\pi(\cdot\|s_t;\theta))\big)$
DDPG (Deep Deterministic Policy Gradient)	$L(\theta^\mu) = -\mathbb{E}_{s_t\sim\rho^\beta}[\nabla_{\theta^\mu} Q(s,a\|\theta^Q)\|_{s=s_t,a=\mu(s_t\|\theta^\mu)}]$
SAC (Soft Actor Critic)	$J(\theta) = \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[\alpha \log \pi_\theta(a_t\|s_t) - Q_\phi(s_t,a_t)\right]$
PPO (Proximal Policy Optimization)	$L(\theta) = \mathbb{E}_t\left[\min\left(\frac{\pi_\theta(a_t\|s_t)}{\pi_{\theta_{\text{old}}}(a_t\|s_t)} A^{\text{clip}}(\theta), 1+\epsilon, 1-\epsilon\right) A_t - \beta H(\pi(\cdot\|s_t;\theta))\right]$
TRPO (Trust Region Policy Optimization)	$\max_\theta \mathbb{E}_t\left[\frac{\pi_\theta(a_t\|s_t)}{\pi_{\theta_{\text{old}}}(a_t\|s_t)} A^{\text{clip}}(\theta)\right]$
DQN + Prioritized Experience Replay	$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'\sim D_i}[(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2 \cdot \mathbb{I}(i \in B)]$

Table 3.1: Objective functions of key RL algorithms (Achiam, 2018; Larsen et al., 2021)

(Schulman et al., 2017) shows that the clipped surrogate objective used by PPO allows it to use a pessimistic bound over the objective function and hence converge to a better optima. C. Yu et al., 2022 also show that PPO scales better when extended to cooperative multiagent scenarios. This is why PPO is one of the most sought-after RL algorithms currently.

Algorithm	Obsv Space	Action Space	Type	Policy
Q-learning	Discrete	Discrete	Value-based	Off-policy
SARSA	Discrete	Discrete	Value-based	On-policy
DQN (Deep Q Network)	Continuous	Discrete	Value-based	Off-policy
A3C (Asynchronous Advantage Actor-Critic)	Continuous	Discrete	Actor-critic	On-policy
DDPG (Deep Deterministic Policy Gradient)	Continuous	Continuous	Actor-critic	Off-policy
SAC (Soft Actor Critic)	Continuous	Continuous	Actor-critic	Off-policy
PPO (Proximal Policy Optimization)	Continuous	Continuous or Discrete	Actor-critic	On-policy
TRPO (Trust Region Policy Optimization)	Continuous	Continuous or Discrete	Actor-critic	On-policy
DQN + Prioritized Experience Replay	Continuous	Discrete	Non-stationary targets	

Table 3.2: Comparison of key RL algorithms (Achiam, 2018; Larsen et al., 2021; Wikipedia, 2024a).

We used the implementation of StableBaselines3 to design Dec-PPO for the forward-rollout phase of Dec-AIRL, however, there are many other popular RL libraries that provide elegant implementations of PPO like Ray RLLib, Dopamine, ACME, Mushroom-RL, Tianshou, etc. While Stable-Baselines3 provides almost all the features required for most RL reseach applications, it also has some experimental techniques like Recurrent-PPO, on its contrib branch of the library.

Chapter 4

Related work

In this chapter, we list (to the best of our knowledge) the works related to the research problems we handle in this dissertation. We briefly discuss these prior works and highlight why they are insufficient to solve the problems we address. We organize the related works in the order of our contributions for easy perusal.

4.1 Noisy and missing data

One of the first approaches to consider observer noise (Shahryari and Doshi, 2017) expands the well-known maximum entropy IRL (B. D. Ziebart et al., 2008) to maximize the entropy of the joint distribution of the hidden state-action trajectories and observation sequences. A Lagrangian relaxation of this non-linear program yields gradients that can be used in the optimization. However, these gradients are hard to compute and hence make the approach computationally unwieldy and challenging to scale. On the other hand, techniques like Robust-BIRL (Zheng et al., 2014), D-REX (Brown et al., 2020), and the more recent SSRR (L. Chen et al., 2021) target noisy trajectories due to the expert's failures during task performance. The former is based on the premise that noisy execution may cause the expert to sometimes follow off-policy actions. A latent variable characterizing the reliability of the action is introduced and an expectation-maximization schema in the framework of BIRL manages this noise. The latter technique (D-REX) solves the problem of automatically ranking demonstrated trajectories based on Luce-Shepard rule while SSRR uses Adversarial IRL and assumes that the demonstrator is suboptimal and that pairwise preferences over trajectories are additionally needed for IRL. However, none of these methods introduce an observation model or account for partially occluded trajectories. As the expert in our setting fully and perfectly observes their state while the learner experiences noise due to imperfect sensors, IRL methods that model the expert as a partially observable MDP (POMDP) (Choi and Kim, 2011b) are not relevant. A Bayesian nonparametric IRL technique (Michini and How, 2012) allows partitioning the task into smaller parts such that a reward function can be learned for each part using the Chinese restaurant process. This can be used to model different types of noise caused by different sources of noise in the environment when the number of sources is not known a priori.

Over the past few years, there has been a steady stream of methods for inverse learning in the context of occlusion. Beginning with a method that ignores the occlusions (uses just the available data) for maximum entropy IRL (Bogert and Doshi, 2015), to the HiddenDataEM, which inferred the hidden variables in actions using the expectation-maximization schema (Bogert et al., 2016), followed by ways of improving the computational tractability of the approach (Bogert and Doshi, 2017). Mai et al. [2019] show that forward solving the expert's MDP using a system of linear equations also allows for inferring the missing portions of the input data. But, it requires the underlying Markov chain to be non-cyclic – an assumption difficult to satisfy in practice. There has been some work in online IRL, that also includes occlusions (Arora et al., 2021), however, since the focus of this document is on offline IRL methods, we will not delve deeply into online IRL. Kim et al. 2016, use MAP-BIRL to learn socially adaptive path planning and mention that they use a binary feature vector representation with regularization to alleviate the problem of occlusions and sensor noise but don't go into any details about the specifics of this method.

4.2 Multi-agent coordination/cooperation

Recently, a few IRL techniques have targeted the problem of multi-agent coordination/cooperation. They model the problem either as a game-theoretic equilibrium, solve an MMDP with the assumption that all agents have access to the global state, or just consider the other agent(s) as part of the transition noise in the environment (S. Nikolaidis, Ramakrishnan, et al., 2015).

Multi-agent imitation learning. Imitation learning techniques that deal with multi-agent tasks include MAGAIL (Song et al., 2018), CoDAIL (Liu et al., 2020), MA-DAAC (Jeon et al., 2020), and Bayesian MA-DAAC (Yang et al., 2020). These techniques extend GAIL to a multi-agent setup. MAGAIL and CoDAIL model the problem as a Markov game with Nash equilibrium as the convergence criteria. MA-DAAC focuses on the scalability of multi-agent GAIL by representing the generator with an actor-attention-critic instead of the regular actor-critic and using a centralized or decentralized discriminator. All of these methods evaluate their approach on multi-agent particle environments [1] in both cooperative and competitive setups. The most recent work Co-GAIL (C. Wang et al., 2022) extends InfoGAIL (Lin et al., 2017) to model human-robot interaction scenarios with multiple human strategies. They model it as an MMDP, where all agents have access to the global state, and assume that the latent feature, the human strategy, is shared between agents. Co-GAIL is implemented on iGibson's simulated human-robot domains and the expert trajectories are recorded using Amazon turk. BTIL (Seo & Unhelkar, 2022) uses a task-model and an agent-model and infers the time-varying mental states of team members, and learns decentralized team policies from demonstrations of suboptimal teamwork. It also jointly learns their transition models. BTIL has been tested only on a movers and packers domain which is a relatively small, simulated platform and although the paper claims to learn a decentralized policy, their policy is a mapping of agent action to the global task state, which is not a fully decentralized representation. *None of these techniques utilize a decentralized MDP as the underlying model of the task*, which is better suited

[1] This is a simulated platform by OpenAI, which has small multi-agent domains that do not involve adverse interactions.

for modeling collaborative domains where agents only have access to their local attributes. They also do not model necessary or adverse interactions between agents and do not evaluate the method on real-world domains that require a human actively interact with a robot.

Multi-agent inverse RL. In the IRL context, MA-AIRL (L. Yu et al., 2019), ME-AIRL (Wei et al., 2019), and MA-AIRL-Latent (Gruver et al., 2020) extend AIRL to a multi-agent setup. MA-AIRL models the problem under logistic stochastic best response equilibrium (also known as the quantal response) and tests the method on multi-agent particle environments. MA-AIRL-Latent aims to improve the sample efficiency of MA-AIRL when the agents have shared latent variables, and evaluates on highway trajectories from the highD data set (Krajewski et al., 2018) and trajectories of aircraft in terminal airspace from non-public Federal Aviation Administration (FAA) data (also simulated domains). ME-AIRL uses an analytical method to connect the actor-critic model with AIRL and GAIL and measures the performance of the method on a simple single-state continuous game for 2 agents with one action dimension per agent. Older techniques such as Joint-Action-IRL (S. Nikolaidis, Ramakrishnan, et al., 2015) BA-IRL (Trivedi & Doshi, 2018) consider the other agent to be part of the transition noise and model the IRL problem as a single agent scenario. Joint-Action-IRL implements the method on a hand-finishing task where the human's role is to refinish the surface of a box attached to an ABB industrial robot and the robot moves to aid the process. BA-IRL implements their method on a ROS-Gazebo simulation of two Turtlebots trying to sort balls on a table using attached manipulators. An early technique GSSG-IRL (Reddy et al., 2012) models decentralized non-cooperative multi-agents using IRL, but formulates the problem as a Nash equilibrium based general-sum stochastic game between the agents and tests the technique on a simple gridworld domain only. The key difference between these techniques and Dec-AIRL is that in our case, *the agents are cooperative and maximize a common system reward instead of working in self-interest*. Additionally, we test our method on both a simulated continuous domain and a real-world discrete domain with human-robot collaboration scenarios that model adverse and necessary interactions.

4.3 Human-robot collaboration as a multiagent system

The primary aim of HRC is to develop systems that efficiently avail the combined strengths of human and robotic agents. Robots are assigned tasks that are repetitive or physically demanding, enabling humans to focus on activities requiring cognitive and dexterous skills. Achieving effective collaboration in shared workspaces necessitates both human and robotic agents to respond appropriately to each other's actions and environmental changes.

The selection of a suitable multiagent decision-making model depends on the agents' natures and objectives. For instance, in scenarios where agents are competitive, the task can be framed as a Markov game, with an equilibrium condition ensuring the best strategy profile (Littman, 1994). Conversely, if agents are collaborative and possess complete knowledge of one another, a fully centralized framework like multiagent MDP (MMDP) can be employed. Alternatively, one might consider the human agent(s)' actions as impacting only the transition dynamics, allowing them to be marginalized; in such cases, a single-agent MDP suffices to learn just the robot's behavior.

In their exploration of HRC tasks, S. Nikolaidis and Shah, 2012 and M. Chen et al., 2020 model their HRC tasks as a single-agent partially observable MDP (POMDP) (Monahan, 1982). While S. Nikolaidis, Ramakrishnan, et al., 2015 and S. Nikolaidis, Zhu, et al., 2017 consider slightly simpler cases where only portions of the world state are partially observable and adopt a mixed-observability MDP (MOMDP) (Ong et al., 2010) to model the task. Conversely, S. Nikolaidis, Nath, et al., 2017 model their task as a two-player Markov game, where the human adapts their behavior with evolving expectations of the robot's capabilities. Alternatively, Unhelkar et al. 2020 and Seo and Unhelkar, 2022 split their HRI scenario into an agent model and a task model, where the task model - MMDP, captures the task attributes and the agent Markov model, the mental states of the other agent.

A recent approach by C. Wang et al., 2022 and van der Spaa et al., 2024 model a simulated HRC handover and combined manipulation task using an MMDP and use inverse learning methods (Ho & Ermon, 2016; B. D. Ziebart et al., 2008) to learn collaborative joint policies. Along the same lines, recent works by Yuan et al., 2022 and Jiang et al., 2024 use a variational inference to discover a latent strategy of the human teammate for enhanced collaboration, and represent their task as a decentralized scenario. Another recent work by Sengadu Suresh et al., 2023 models their HRC scenarios as a Dec-MDP (Goldman & Zilberstein, 2004) where the human and robot are each aware of their own local state and some task attributes only. A vector of policies (one for each agent) is learned and the robot policy is tested on a simulated patient assistance task (Erickson et al., 2020) and a realistic collaborative produce sorting task (P. S. Suresh & Doshi, 2022). While all these previous works solve key problems in HRC, notice that they all use *closed* models to represent their HRC scenarios and hence are unsuitable for OHRCS.

4.4 Ad Hoc teamwork and open collaboration

4.4.1 Ad Hoc Teamwork

A line of work that aims to address ad hoc teamwork (Stone et al., 2010) makes related contributions. Mirsky et al., 2022 defines ad hoc teamwork as *"To create an autonomous agent that is able to efficiently and robustly collaborate with previously unknown teammates on tasks to which they are all individually capable of contributing as team members"*. While research into ad hoc teamwork has made great strides in the past decade, one key difference between such teamwork and OHRC is that the latter focuses on agents learning to collaborate within an open, dynamically changing team that allows both human and robotic agents to enter and exit the task as needed. For example, the task may begin as a dyadic human-robot team but evolve to engage more humans and vice versa. Therefore, OHRC expands the scope of ad hoc teamwork to settings where other agents can join or leave the system while the task is ongoing. As such, the decision-making complexity of OHRC is considerably higher than ad-hoc teamwork; the details of which we will discuss in the upcoming sections.

Paper	AO	TaO	TO	Experiment type	Task nature	Key contribution
(Simão & Demazeau, 2001)	✓	✓	✗	N/A	Planned-CoOp	Social reasoning framework
(Jumadinova et al., 2013)	✓	✓	✗	Sim	Ad Hoc	Teacher-learner framework
(B. Chen et al., 2015)	✓	✓	✗	Sim	Ad Hoc	Flexible ad hoc framework
Open-DecPOMDP (Cohen et al., 2017))	✓	✗	✗	Sim	Planned-CoOp	Open agent collab model
I-PBVI (Chandrasekaran et al., 2016))	✓	✗	✓	Sim	Self-Interest	Posthoc & predictive planning
I-POMCP (Eck et al., 2020))	✓	✓	✗	Sim	Self-Interest	Scalable planning framework
CI-POMCP-PF (Kakarlapudi et al., 2022))	✓	✓	✗	Sim	Self-Interest	Planning with communication
LIA2C (He et al., 2023))	✓	✓	✗	Sim	Self-Interest	DTDE-MARL for open systems
GPL-SPI (Rahman et al., 2023))	✓	✗	✗	Sim	Ad Hoc	Graph RL for open systems
Fastap (Zhang et al., 2023))	✗	✗	✓	Sim	Planned-CoOp	MARL for fast policy adaptation

Table 4.1: Prior works in Open Agent Systems. Ad Hoc - working with previously unseen teammates; Planned-CoOp - collaborating with known teammates towards a common goal; Self-interest - optimizing agent-specific cumulative return; AO- Agent Openness, TaO - Task Openness, TO - Type Openness.

4.4.2 Open Agent Systems

Multiagent systems with the additional relaxation of openness introduces several additional challenges since the agents must now also take into account: the agents that have left or joined in - agent openness (AO), type of agents that exited or entered - type openness (TO), and any changes in the goals - task openness (TaO). While prior research in multiagent systems has achieved broad success within closed systems, most real-world domains tend to be *open* (Eck et al., 2023). Hence, developing sophisticated methods to model openness becomes crucial, especially when considering realistic domains such as HRC.

Each agent may only have limited information at every timestep regarding goals, active agents, their type, abilities, and level of openness. Therefore, the first challenge becomes modeling the task appropri-

ately to capture such uncertainties. One of the earliest works in the field, Simão and Demazeau, 2001, addresses the concept of agent openness from a social reasoning perspective that enables an agent to reason about others when the agents' organization is not available a priori. They hypothesize that such a mechanism can be used both as a basis for coalition and decision-making to adapt to dynamic changes. On the other hand, Jumadinova et al., 2013 consider distributed collaboration among multiple agents with both TaO and AO. They use a teacher–learner framework to decide what capabilities to learn from the other agents and validate their model using a simulated agent-based modeling tool ("RePast Framework", 2023). They conclude that the agents that learn all capabilities, regardless of their usefulness, outperform the other agents.

The work by Chen et al. 2015 considers two types of openness, AO and TaO, in an ad hoc setting to ascertain how they contribute to learning. They assume that all agents know the level of AO and TaO and perform simulated experiments where agents can bid on tasks based on their knowledge level. The authors conclude that while TaO makes it harder for the agents to solve the tasks, AO improves the performance, as newer agents could bring additional capabilities to help solve the tasks. Cohen et al., 2017 provide a modified Dec-POMDP framework (Goldman & Zilberstein, 2004) that can be applied to open systems by factoring in the coalition model during decision-making. They use an offline best-response algorithm and Monte Carlo tree search (MCTS) (Browne et al., 2012) to plan appropriate actions under different coalitions. They validate their model using a simulated urban firefighting domain. However, in their coalition transition model, agents make the decision to transition to a new coalition solely based on the previous coalition, which is unrealistic. Additionally, the methods used for planning such as the best-response algorithm and MCTS do not scale well to real-world domains.

Using an IPOMDP-Lite framework, (Hoang & Low, 2013), Chandrasekaran et al., 2016 model an open system and use an agent interaction graph for post hoc reasoning. They validate their model on the simulated wildfire-suppression domain and compare their learned policies with heuristic-based and random policies. Eck et al., 2020 extend this method to be more scalable by selectively modeling the neighboring agents and providing theoretical bounds for the same using regret analysis. They consider more complex scenarios of the wildfire-suppression domain and compare their learned policy with heuristics-based policies.

A novel model called CI-POMDP was proposed by Kakarlapudi et al., 2022 that improves upon the IPOMDP-Lite framework by introducing communication between agents. They posit that by using the ability to communicate, agents can better navigate the challenges of an open system. They use the wildfire-suppression domain to validate their claims, compare various levels of communication, and analyze its corresponding costs. A recent work He et al., 2023 uses a reinforcement learning (RL) technique called decentralized training with decentralized execution policy gradient method with an underlying I-POMDP model to solve a simulated open-organization problem where employees can join or quit the organization at any point. They propose a novel method to factor partial-observability in open agent systems. Finally, the most recent works Rahman et al., 2023 and Zhang et al., 2023 use multiagent RL to solve openness problems. The former proposes a partially observable open stochastic Bayesian game model to model AO and use a graph-based policy learning approach to learn RL policies. The latter studies decision-making

through multiagent RL when other agents' policies abruptly change during the task; which could be considered TO. They both evaluate their method using simulated toy domains like Level-based Foraging, WorldPack, and PredatorPrey.

Some common denominators (as summarized in Section 4.4.2) that can be considered limitations of the aforementioned prior works are that they evaluate their methods on small, simulated, or toy domains like the urban firefighting or the open-organization domain, and largely do not provide theoretical or empirical analysis of convergence, regret, scalability, or sample complexity (when applicable). Methods using IPOMDPs or Game Theory approaches do not apply to HRC since they typically model competitive or self-interested scenarios. Finally, it is essential to demonstrate the effectiveness of the proposed methods on a physical system such as a robot in real settings, to ascertain their applicability to HRC. Considering these factors, existing techniques may not be directly usable in HRC, although they address similar concepts.

4.5 Modelling other agents

Modeling teammates to enhance decision-making in multiagent systems is common in fields like Game Theory, multiagent planning, and reinforcement learning. Previous work has had success in simple environments like SMAC (Samvelyan et al., 2019) and Level-Based Foraging, but research in this area is limited. Kamar et al., 2009 extends an AND-OR tree to model agents, their context, and the subset performing a particular action, determining the usefulness of a helping action for the team. However, probabilistic trees do not scale to large, complex, real-world domains.

Albrecht and Ramamoorthy, 2015 introduced a Harsanyi-Bellman Ad Hoc Coordination strategy for Bayesian Games (Shoham & Leyton-Brown, 2008), optimizing actions based on teammates' strategies. Another approach, interactive POMDPs (I-POMDPs) (Gmytrasiewicz & Doshi, 2005; Panella & Gmytrasiewicz, 2016), recursively updates beliefs about other agents' models while solving for their policies, interleaving learning and planning in multiagent settings. Unlike our collaborative focus, I-POMDPs and Game Theory-based models are often tailored for competitive contexts.

Unhelkar et al. 2020b proposed an agent Markov model (AMM) to capture an agent's mental states, allowing inference through variational Bayesian methods. However, this approach does not develop behavioral policies for efficient collaboration at every step. Recent work by Seo and Unhelkar, 2024 learns the expert's policy and hidden intent using expectation-maximization and IQ-Learn (Garg et al., 2021), modeling the human as a single-agent MDP, unlike our multiagent scenario.

Other work (Gopalan & Tellex, 2015; Sadigh et al., 2016; Whitney et al., 2017) uses POMDPs to represent tasks as partially observable, with robots performing probing actions to gather information and update beliefs about agents' hidden states. Nikolaidis et al. 2015 used expectation-maximization to cluster demonstrations and derive different reward functions for each expert type, assuming constant agent types, unlike our scenario which accounts for changing agent types mid-task.

Previous works using bounded-memory models combined with MOMDPs estimate human adaptability (S. Nikolaidis, Forlizzi, et al., 2017; S. Nikolaidis, Lasota, et al., 2015; S. Nikolaidis, Zhu, et al., 2017),

enabling robots to adjust actions. Similar methods use MOMDPs with verbal clarifications or communication to update robots' beliefs over humans' mental states (S. Nikolaidis et al., 2018, Unhelkar et al. 2020a). Our TB-Dec-MDP maintains beliefs over agent types, computing policies based on states and types, capturing collaboration nuances for maximizing team rewards.

Orlov-Savko et al., 2022 uses a factored AMM to represent behavior of agents performing sequential tasks, employing variational inference with execution traces and domain-specific priors to learn agent models semi-supervisedly. Orlov Savko et al., 2024 addresses the lack of data on human mental states, providing an extensive dataset for inverse learning. Peternel et al. 2018 adapted robot behavior online to account for human motor fatigue using Dynamical Movement Primitives and Adaptive Frequency Oscillators.

Trust-POMDP (Chen et al. 2020) maintains a belief over trust as a latent variable, allowing the robot to adapt its policy based on human interruptions and trust. This work aligns more closely with active learning (Settles, 2009) than with traditional HRC. Yuan et al. 2022 and Jiang et al. 2024 applied variational inference with mutual information maximization in decentralized settings, learning a variable z informing each agent about others, assuming a constant latent strategy space, unlike our TB-Dec-MDP allowing diverse and evolving agent types.

Recent work (van der Spaa et al., 2024) focuses on learning human hidden intent and task preferences for effective human-robot collaboration using maximum entropy IRL, employing an MMDP model for a centralized framework, limiting applicability to decentralized scenarios. Prasad et al. 2024 modeled human-robot interactions with a mixture of Gaussians, learning the latent space of robot actions, but their policy is conditioned solely on human movements. Qian et al., 2024 uses the help of after-action reviews and human subject studies to quantify and establish the impact of latent factors such as trust and intent on human-robot collaboration.

Ultimately, HRC aims to create human-centric solutions prioritizing human time and energy. Techniques unvalidated on physical systems may not be applicable in real-world HRC contexts, even if they address related concepts.

Chapter 5

Occlusions and sensor noise

The goal of data-driven AI techniques is to capture an accurate representation of the underlying features of the data and their correlations. This is usually done by learning a mathematical representation of the data using a probability distribution that sufficiently captures the intricacies. However, noisy or missing data may obscure certain underlying features, key to learning a meaningful representation. Typically in IRL, the expert trajectory is assumed to be noiseless and unoccluded (in other words, the input data needs no preprocessing). However, such an assumption is hard to satisfy in most real-world domains, where the trajectories are recorded using sensors such as RGB-D cameras. These sensors often tend to be imperfect and usually have a limited field of view, leading to occluded and noisy expert trajectories.

5.1 Occlusions and its effects on IRL

In the IRL context, occlusions usually refer to one or more timesteps of the expert trajectory being inaccessible for learning. While the learner has the MDP model of the expert (sans the reward function) and can make an inference just using the unoccluded data in the trajectories, we posit that it is necessary to process the missing data to learn a well-rounded reward function that sufficiently captures the expert's underlying preferences.

This is because the missing portions of the trajectory may contain key features [1] relevant to safety, goal states, or in the case of HRC, interaction between agents. In such cases, the missing data makes a considerable difference in reward learning since each timestep is causally dependent on its previous timestep. In keeping with previously established notation from Section 2.2, let the set of input trajectories of finite length $\mathcal{T}$ generated by an MDP attributed to the expert be, $\mathcal{X}^{\mathcal{T}} = \{X | X = Y \bigcup Z\}$. Here, Y is the observable portion and Z is the occluded part of a trajectory X. A complete trajectory X is a sequence, $X = (s^1, a^1, s^2, a^2, s^3, ..., s^{\mathcal{T}}, a^{\mathcal{T}})$; some of these may be occluded.

[1] Recall the definition of feature expectations ϕ_π discussed in Section 2.2.

5.2 Sensor noise and its effects on IRL

Sensors are arguably the most widely used tools in robotics to obtain live feedback about the environment. Usually, sensor data is passed through smoothing filters (Kassam & Poor, 1985) before being used to interpret desiderata. However, always relying on the existence of such preprocessing modules is unpragmatic. Therefore, IRL techniques must adapt to use noisy input to learn the expert's true underlying preferences. Noise, however, makes it challenging to ascertain the underlying state and action of the expert at each step. Therefore, instead of ground-truth state-action pairs, we receive observations, with multiple state-action pairs mapped to each observation with a certain probability. To provide an example of an *observation*, let's briefly look at the tiger problem:

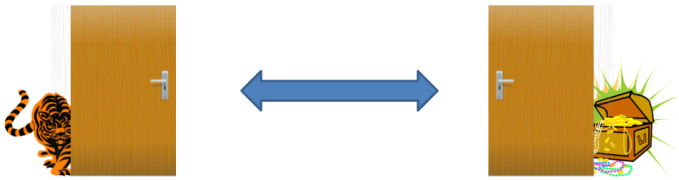


Figure 5.1: A popular example used to explain a partially observable scenario. The agent has to open one of two doors based on the observation it receives at every step. One door has a chest of gold, while the other has a tiger behind it. The agent can open the left door, open the right door, or listen for the tiger's growl with a small penalty. The tiger's growl is considered the observation in this context, since hearing the growl behind one door does not conclusively establish its position.

In this tiger problem shown in Fig 5.1, the agent has 2 states, 2 actions, and 2 observations. The observations are growl-left, growl-right; the states are tiger-left, tiger-right, and the actions are open-left, open-right, and listen (which incurs a small penalty). The agent has to build a probability distribution over the observations, use that to decide which action to perform, and avoid the door with the tiger.

The probability distribution associated with observations and their corresponding state-action pairs is known as an observation model or a sensor model. In our IRL context, let $O_l(s^1, a^1, o_l^1)$ represent $Pr(o_l^1|s^1, a^1)$, which is the learner's stochastic mapping from the expert's state and performed action at timestep 1: (s^1, a^1), to the learner's observation of it, o_l^1. O_l informs the learner about the expert's state and action, albeit noisily. We assume that the learner has access to the observation model as part of the MDP, however, a recent work attempts to learn the observation model alongside the reward function (Arora et al., n.d.).

5.3 MMAP-BIRL to handle occlusions and sensor noise

We generalize MAP-BIRL (Section 2.3) to learn in the context of both occlusions and noisy input data. Let $X = (o_l^1, o_l^2, o_l^3, ..., o_l^{\mathcal{T}})$ where each element o_l^t is the learner's observation of the expert at a time step t; some of these observations may be occluded. We begin by using the parameterized linear sum of reward features, $R_{\boldsymbol{\theta}}$, as the representation of the reward function. Subsequently, the prior over the

reward function (Eq. 2.6) is now a distribution over the feature weights, decomposed into an independent distribution over each weight:

$$Pr(R_{\boldsymbol{\theta}}) = \prod_{\theta_k \in \Theta} Pr(\theta_k). \tag{5.1}$$

The likelihood of the visible portions of the trajectories can be written as the marginal of the complete trajectory X by summing out the corresponding hidden portion Z:

$$Pr(\mathcal{Y}|R_{\boldsymbol{\theta}}) = \prod_{Y \in \mathcal{Y}} Pr(Y|R_{\boldsymbol{\theta}}) = \prod_{Y \in \mathcal{Y}} \sum_{Z \in \mathcal{Z}} Pr(Y, Z|R_{\boldsymbol{\theta}}) = \prod_{Y \in \mathcal{Y}} \sum_{Z \in \mathcal{Z}} Pr(X|R_{\boldsymbol{\theta}}).$$

Here, the parameters $\boldsymbol{\theta}$ are the maximization variables, and the occluded portion Z of a trajectory comprises the summation variables of the marginal MAP inference. Using the above likelihood function, the MMAP-BIRL problem is fully formulated as:

$$R^*_{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \prod_{Y \in \mathcal{Y}} \sum_{Z \in \mathcal{Z}} Pr(Y, Z|R_{\boldsymbol{\theta}})\, Pr(R_{\boldsymbol{\theta}}). \tag{5.2}$$

Let Z be the collection of the observations in the occluded time steps of X, and $Y = X/Z$. Then,

$$R^*_{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \prod_{Y \in \mathcal{Y}} \sum_{Z \in \mathcal{Z}} Pr(o_l^1, o_l^2, o_l^3, ..., o_l^{\mathcal{T}}|R_{\boldsymbol{\theta}}) \times Pr(R_{\boldsymbol{\theta}}).$$

The learner's observation o_l^t is a noisy perception of the expert's state and action at time step t, and the observations are conditionally independent of each other given the expert's state and action. Therefore, we introduce the state-action pairs in the likelihood function above.

$$Pr(o_l^1, o_l^2, o_l^3, \ldots, o_l^{\mathcal{T}}|R_{\boldsymbol{\theta}}) = \sum_{s^1, a^1, ..., s^{\mathcal{T}}, a^{\mathcal{T}}} Pr(o_l^1, o_l^2, o_l^3, ..., o_l^{\mathcal{T}}, s^1, a^1, s^2, a^2, ..., s^{\mathcal{T}}, a^{\mathcal{T}}|R_{\boldsymbol{\theta}}).$$

For convenience, let τ denote the underlying (hidden) trajectory of the actual state-action pairs of the expert, $\tau = (s^1, a^1, s^2, a^2, ..., s^{\mathcal{T}}, a^{\mathcal{T}})$. Then, we may reformulate the MMAP-BIRL problem as:

$$R^*_{\boldsymbol{\theta}} = \arg\max_{R_{\boldsymbol{\theta}}} \prod_{Y \in \mathcal{Y}} \sum_{Z \in \mathcal{Z}} \sum_{\tau \in (|S||A|)^{\mathcal{T}}} Pr(o_l^1, o_l^2, o_l^3, \ldots, o_l^{\mathcal{T}}, \tau|R_{\boldsymbol{\theta}})\, Pr(R_{\boldsymbol{\theta}}). \tag{5.3}$$

5.3.1 MMAP-BIRL inference

The MMAP inference problem is hard. Previous approaches, mostly in the context of Bayesian network inference, have utilized AND-OR graph structures to perform the inference (Marinescu et al., 2014). But, the maximization variables in these techniques are discrete, which allows the use of a discrete data structure such as a graph to model the inference. As our maximization variables are continuous, we seek to solve the hard MMAP inference problem using continuous-variable optimization such as gradient ascent.

The log forms of the prior and the likelihood function in (5.3) are represented respectively as:

$$L^{pr}_{\boldsymbol{\theta}} = \log Pr(R_{\boldsymbol{\theta}}); \quad L^{lh}_{\boldsymbol{\theta}} = \sum\nolimits_{Y \in \mathcal{Y}} \log \sum\nolimits_{Z \in \mathcal{Z}} \sum\nolimits_{\tau \in (|S||A|)^{\mathcal{T}}} Pr(o^1_l, o^2_l, o^3_l, \ldots, o^{\mathcal{T}}_l, \tau | R_{\boldsymbol{\theta}}).$$

If we let the prior $Pr(\theta_k)$ in Eq. 5.1 be Gaussian (some values of each feature weight are more likely than others), i.e.,

$$Pr(\theta_k; \mu_\theta, \sigma_\theta) = \frac{1}{\sqrt{2\pi}\sigma_\theta} e^{-\frac{(\theta_k - \mu_\theta)^2}{2\sigma^2_\theta}} \tag{5.4}$$

where the mean μ_θ and standard deviation σ_θ may differ between the feature weights. Then, the gradient of the log prior is obtained as [2],

$$\frac{\partial L^{pr}_{\boldsymbol{\theta}}}{\partial \theta} = \frac{-(\theta - \mu_\theta)}{2\sigma^2_\theta}.$$

Next, to obtain the gradient of $L^{lh}_{\boldsymbol{\theta}}$, we may expand the term $Pr(o^1_l, o^2_l, o^3_l, \ldots, o^{\mathcal{T}}_l, \tau | R_{\boldsymbol{\theta}})$ as shown below.

$$\begin{aligned} Pr(o^1_l, o^2_l, o^3_l, \ldots, o^{\mathcal{T}}_l, \tau | R_{\boldsymbol{\theta}}) &= Pr(o^1_l, o^2_l, o^3_l, \ldots, o^{\mathcal{T}}_l | \tau, R_{\boldsymbol{\theta}})\, Pr(\tau | R_{\boldsymbol{\theta}}) \qquad \longrightarrow \text{ chain-rule} \\ &= Pr(o^1_l | o^2_l, o^3_l, ..., o^{\mathcal{T}}_l, \tau, R_{\boldsymbol{\theta}})\, Pr(o^2_l, o^3_l, ..., o^{\mathcal{T}}_l, \tau | R_{\boldsymbol{\theta}})\, Pr(\tau | R_{\boldsymbol{\theta}}) \\ &= \underbrace{Pr(o^1_l | s^1, a^1)}_{\text{conditional independence}}\, Pr(o^2_l, o^3_l, \ldots, o^{\mathcal{T}}_l, \tau' | R_{\boldsymbol{\theta}})\, Pr(\tau | R_{\boldsymbol{\theta}}). \end{aligned}$$

The last step is obtained by noting that the learner's current observation is conditionally independent of its future observations given the expert's true state and action in the same time step. Using the above observation model, we may continue expanding $Pr(o^2_l, o^3_l, \ldots, o^{\mathcal{T}}_l, \tau' | R_{\boldsymbol{\theta}})$ as,

$$\begin{aligned} Pr(o^1_l, o^2_l, o^3, \ldots, o^{\mathcal{T}}_l, \tau | R_{\boldsymbol{\theta}}) &= O_l(s^1, a^1, o^1_l) Pr(o^2_l, o^3_l, \ldots, o^{\mathcal{T}}_l, \tau' | R_{\boldsymbol{\theta}})\, Pr(\tau | R_{\boldsymbol{\theta}}) \\ &= \prod_{t=1}^{\mathcal{T}} O_l(s^t, a^t, o^t_l)\, Pr(\tau | R_{\boldsymbol{\theta}}). \end{aligned}$$

Observe that the last term $Pr(\tau | R_{\boldsymbol{\theta}})$ corresponds exactly to $Pr(X | R_{\boldsymbol{\theta}})$ in Eq. 2.7 for a trajectory of state-action pairs $X \in \mathcal{X}$. Hence, we use the terms inside the outer product of Eq. 2.7 to substitute

[2] The gradients for the MMAP-BIRL prior and likelihood have been derived in Section 5.3.2

$Pr(\tau|R_{\boldsymbol{\theta}})$ above as follows,

$$Pr(o_l^1, o_l^2, o_l^3, \ldots, o_l^{\mathcal{T}}, \tau|R_{\boldsymbol{\theta}}) = \prod_{t=1}^{\mathcal{T}} O_l(s^t, a^t, o_l^t) Pr(s^1)\, \pi(a^1|s^1;\boldsymbol{\theta}) \prod_{t'=1}^{\mathcal{T}-1} T(s^{t'}, a^{t'}, s^{t'+1}) \times \pi(a^{t'+1}|s^{t'+1};\boldsymbol{\theta})$$
$$= Pr(s^1)\, \pi(a^1|s^1;\boldsymbol{\theta}) \left(\prod_{t=1}^{\mathcal{T}-1} O_l(s^t, a^t, o_l^t) T(s^t, a^t, s^{t+1})\, \pi(a^{t+1}|s^{t+1};\boldsymbol{\theta}) \right) \times O_l(s^{\mathcal{T}}, a^{\mathcal{T}}, o_l^{\mathcal{T}}).$$

We may now rewrite the log likelihood $L_{\boldsymbol{\theta}}^{lh}$ more fully as,

$$L_{\boldsymbol{\theta}}^{lh} = \sum_{Y\in\mathcal{Y}} \log \sum_{Z\in\mathcal{Z}} \sum_{\tau\in(|S||A|)^{\mathcal{T}}} Pr(s^1)\, \pi(a^1|s^1;\boldsymbol{\theta}) \left(\prod_{t=1}^{\mathcal{T}-1} O_l(s^t, a^t, o_l^t) T(s^t, a^t, s^{t+1}) \pi(a^{t+1}|s^{t+1};\boldsymbol{\theta}) \right) \times O_l(s^{\mathcal{T}}, a^{\mathcal{T}}, o_l^{\mathcal{T}}). \quad (5.5)$$

5.3.2 Deriving the MMAP-BIRL gradients

Derivative of Log-Prior

If we choose the prior $Pr(\boldsymbol{\theta}; \mu_{\boldsymbol{\theta}}, \sigma_{\boldsymbol{\theta}})$ to be Gaussian, then the distribution is given as:

$$Pr(\boldsymbol{\theta}; \mu_{\boldsymbol{\theta}}, \sigma_{\boldsymbol{\theta}}) = \frac{1}{\sqrt{2\pi}\sigma_{\boldsymbol{\theta}}} e^{-\frac{(\boldsymbol{\theta}-\mu_{\boldsymbol{\theta}})^2}{2\sigma_{\boldsymbol{\theta}}^2}}.$$

where the mean $\mu_{\boldsymbol{\theta}}$ and standard deviation $\sigma_{\boldsymbol{\theta}}$ may differ between the feature weights. Then, log prior becomes:

$$L_{\boldsymbol{\theta}}^{pr} = \log\left(\frac{1}{\sqrt{2\pi}\sigma_{\boldsymbol{\theta}}} e^{-\frac{(\boldsymbol{\theta}-\mu_{\boldsymbol{\theta}})^2}{2\sigma_{\boldsymbol{\theta}}^2}} \right) = \log\left(\frac{1}{\sqrt{2\pi}\sigma_{\boldsymbol{\theta}}} \right) + \log\left(e^{-\frac{(\boldsymbol{\theta}-\mu_{\boldsymbol{\theta}})^2}{2\sigma_{\boldsymbol{\theta}}^2}} \right)$$
$$= -\log\left(\sqrt{2\pi}\sigma_{\boldsymbol{\theta}}\right) + \left(\frac{-(\boldsymbol{\theta}-\mu_{\boldsymbol{\theta}})^2}{2\sigma_{\boldsymbol{\theta}}^2} \right).$$

Therefore, partial differential of $L_{\boldsymbol{\theta}}^{pr}$ becomes: $\dfrac{\partial L_{\boldsymbol{\theta}}^{pr}}{\partial \boldsymbol{\theta}} = \left(\dfrac{-(\boldsymbol{\theta}-\mu_{\boldsymbol{\theta}})}{\sigma_{\boldsymbol{\theta}}^2} \right).$

Derivative of log-likelihood

For convenience, let's represent everything within the log term in (5.5) as:

$$h_\theta = \sum_{Z\in\mathcal{Z}} \sum_{\tau\in(|S||A|)^{\mathcal{T}}} Pr(s^1)\,\pi(a^1|s^1;\boldsymbol{\theta}) \left(\prod_{t=1}^{\mathcal{T}-1} O_l(s^t,a^t,o_l^t)\,T(s^t,a^t,s^{t+1})\,\pi(a^{t+1}|s^{t+1};\boldsymbol{\theta})\right) \times O_l(s^{\mathcal{T}},a^{\mathcal{T}},o_l^{\mathcal{T}}).$$

$$L_{\boldsymbol{\theta}}^{lh} = \sum_{Y\in\mathcal{Y}} \log h_\theta \implies \frac{\partial L_{\boldsymbol{\theta}}^{lh}}{\partial\boldsymbol{\theta}} = \sum_{Y\in\mathcal{Y}} \frac{1}{h_\theta}\frac{\partial h_\theta}{\partial\theta}.$$

$$\frac{\partial h_\theta}{\partial\theta} = \sum_{Z\in\mathcal{Z}} \sum_{\tau\in(|S||A|)^{\mathcal{T}}} Pr(s^1)\pi(a^1|s^1;\boldsymbol{\theta}) \left(\prod_{t=1}^{\mathcal{T}-1} O_l(s^t,a^t,o_l^t)\,T(s^t,a^t,s^{t+1})\frac{\partial}{\partial\theta}\left(\prod_{t=1}^{\mathcal{T}-1}\pi(a^{t+1}|s^{t+1};\boldsymbol{\theta})\right)\right) \times O_l(s^{\mathcal{T}},a^{\mathcal{T}},o_l^{\mathcal{T}}).$$

Now let's say for convenience P_θ^π holds $\prod_{t=1}^{\mathcal{T}-1}\pi(a^{t+1}|s^{t+1};\boldsymbol{\theta})$ term from the above equation:

$$P_\theta^\pi = \prod_{t=1}^{\mathcal{T}-1}\pi(a^{t+1}|s^{t+1};\boldsymbol{\theta}) = \pi(a^2|s^2;\boldsymbol{\theta}) \times \pi(a^3|s^3;\boldsymbol{\theta}) \times \pi(a^4|s^4;\boldsymbol{\theta})...\pi(a^{\mathcal{T}-1}|s^{\mathcal{T}-1};\boldsymbol{\theta})$$

$$\begin{aligned}\frac{\partial P_\theta^\pi}{\partial\theta} =& \left(\pi(a^3|s^3;\boldsymbol{\theta}) \times \pi(a^4|s^4;\boldsymbol{\theta})...\pi(a^{\mathcal{T}-1}|s^{\mathcal{T}-1};\boldsymbol{\theta})\right)\frac{\partial\pi(a^2|s^2;\boldsymbol{\theta})}{\partial\theta} + \\ & \left(\pi(a^2|s^2;\boldsymbol{\theta}) \times \pi(a^4|s^4;\boldsymbol{\theta})...\pi(a^{\mathcal{T}-1}|s^{\mathcal{T}-1};\boldsymbol{\theta})\right)\frac{\partial\pi(a^3|s^3;\boldsymbol{\theta})}{\partial\theta} + \\ & \left(\pi(a^2|s^2;\boldsymbol{\theta}) \times \pi(a^3|s^3;\boldsymbol{\theta})...\pi(a^{\mathcal{T}-1}|s^{\mathcal{T}-1};\boldsymbol{\theta})\right)\frac{\partial\pi(a^4|s^4;\boldsymbol{\theta})}{\partial\theta} + ... \\ & \left(\pi(a^2|s^2;\boldsymbol{\theta}) \times \pi(a^3|s^3;\boldsymbol{\theta})...\pi(a^{\mathcal{T}-2}|s^{\mathcal{T}-2};\boldsymbol{\theta})\right)\frac{\partial\pi(a^{\mathcal{T}-1}|s^{\mathcal{T}-1};\boldsymbol{\theta})}{\partial\theta}\end{aligned}$$

$$\frac{\partial P_\theta^\pi}{\partial\theta} = \left(\sum_{t=1}^{\mathcal{T}-1}\frac{\partial\pi(a^{t+1}|s^{t+1};\boldsymbol{\theta})}{\partial\theta}\prod_{k\neq t}^{\mathcal{T}-1}\pi(a^k|s^k;\boldsymbol{\theta})\right).$$

Partial derivative of the policy $\pi(a^{t+1}|s^{t+1};\boldsymbol{\theta})$ is given as,

$$\frac{\partial\pi(a^{t+1}|s^{t+1};\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} = \pi(a^{t+1}|s^{t+1};\boldsymbol{\theta})\left(\frac{\beta\,\partial Q^*(s^{t+1},a^{t+1};\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} - \sum_{a'\in A}\pi(a'|s^{t+1};\boldsymbol{\theta})\frac{\beta\,\partial Q^*(s^{t+1},a';\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right).$$

where the partial derviative of the Q-function can be obtained as:

$$\frac{\partial Q^*(s^{t+1},a^{t+1};\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} = \frac{\partial R_\theta(s^{t+1},a^{t+1})}{\partial\boldsymbol{\theta}} + \gamma\sum_{s'\in S}T(s^{t+1},a^{t+1},s')\sum_{a'\in A}\pi(a'|s^{t+1};\boldsymbol{\theta})\frac{\partial Q^*(s',a';\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}).$$

5.4 Experiments

We use two domains to evaluate MMAP-BIRL. We compare the results from both domains with a previous state-of-the-art baseline - HiddenDataEM (Bogert et al., 2016). However, since HiddenDataEM does not account for observer noise, we generalize it by introducing the observation model O_l into the method. The formative domain is a simulated toy domain derived from Bogert et al., 2016 called Forest world. The summative domain is a realistic use-inspired line sorting domain. We use a popular metric called inverse learning error (ILE) that has been previously used in IRL literature (Arora & Doshi, 2021b). ILE can be defined as $ILE = \sum_{s \in S} \|V^{\pi_E}(s) - V^{\pi_L}(s)\|$, where V^{π_E} is the value of the expert's policy π_E and V^{π_L} is the value of the learned policy π_L using the true MDP. We also compare the time taken for learning for both domains as compared to the baseline method. For both these domains, we use the Boltzmann temperature $\beta = 0.3$, step size $\delta_n = 0.01$, discount factor $\gamma = 0.99$, and a decay rate of 0.95.

5.4.1 Forest world

Forest world is a 4x4 gridworld traversed by a fugitive. A UAV is tasked with learning the goal location and the undesirable locations for the fugitive through reconnaissance, but the latter's movement is not always visible due to forest cover in some sectors, as shown in Fig. 5.2. We model the fugitive's navigation through the grid as a MDP where the states of the MDP are the sector coordinates (x, y) and the actions correspond to the fugitive's movement in the 4 cardinal directions. The start state of the fugitive may vary.

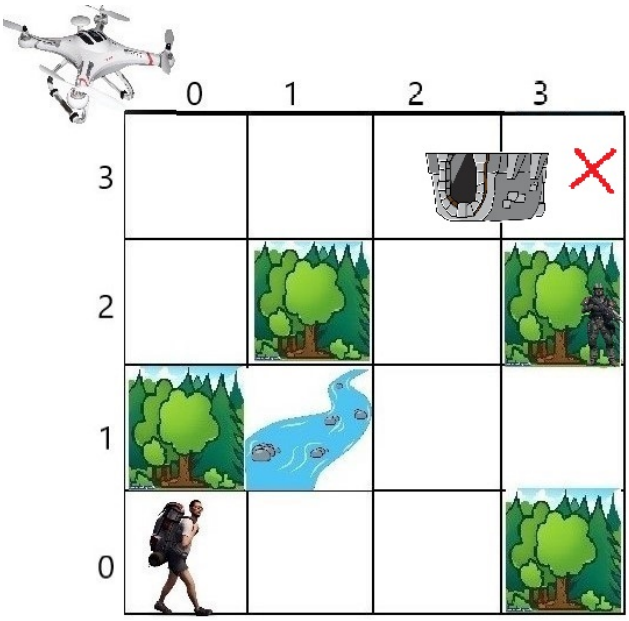


Figure 5.2: A fugitive intends to reach the safe sector $(3, 3)$ while avoiding the river in $(1, 1)$ and the army personnel in $(3, 2)$. These sector preferences are not known to the UAV flying overhead.

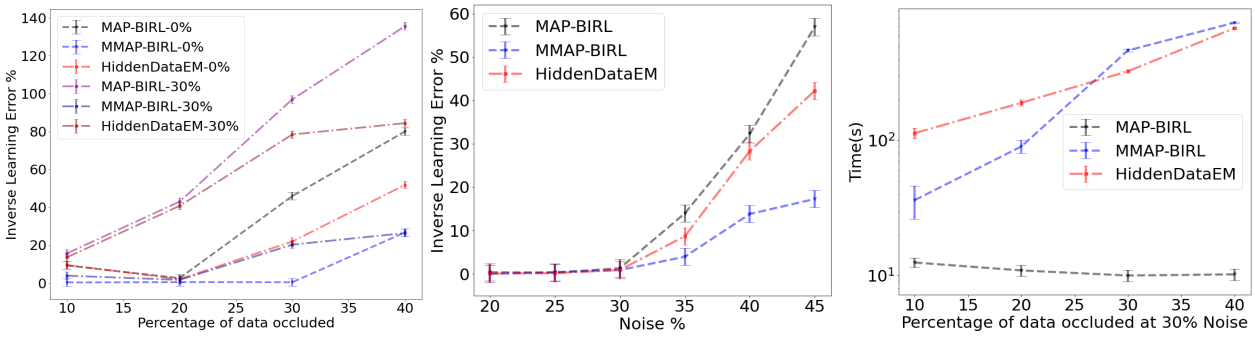
However, the movement within the grid is not deterministic, meaning, there is a 10% chance that the fugitive may end up in any of the three sectors other than the intended one. A hidden tunnel from $(2, 3)$ to $(3, 3)$ makes it hard to ascertain the exact location of the fugitive, thus introducing ambiguity as to whether the fugitive has reached the goal location of $(3, 3)$ or not. With a 30% chance, the UAV's sensors may incorrectly place the fugitive back at $(2, 3)$. This forms the UAV camera's observation model. The UAV models the fugitive's reward function as a weighted linear sum of the following two feature functions:

- $\boldsymbol{Avoidable\_state(x, y)}$ is activated if the fugitive eschews (x, y),
- $\boldsymbol{Goal\_state(x, y)}$ is activated if (x, y) is the fugitive's goal location.

For Fig. 5.2, the fugitive's own reward function places a high negative weight on $\boldsymbol{Avoidable\_state(1, 1)}$ and $\boldsymbol{Avoidable\_state(3, 2)}$, whereas a high positive weight for $\boldsymbol{Goal\_state(3, 3)}$. The trajectory of the fugitive is defined by the steps taken by the fugitive from their start state to the goal state of $(3, 3)$.

Results

We compare the learned weights to the expert's true reward weights since we have access to them in a toy domain. We compare along the 3 features, in a setting of 30% noise and 4 occlusions per trajectory for a total of 10 trajectories with 15 steps each. MMAP-BIRL produces learned reward weights $[-0.7181, -0.8397, 0.6902]$ as compared to the expert's weights of $[-0.5, -1, 0.1]$. However, in order to compare them on a common scale, we apply softmax to both rewards and compare the differences. This shows a difference of 0.2131 along feature 1, 0.0065 along feature 2, and -0.2196 along feature 3. Thus we can infer that the learned weights induce a slightly lower reward for the first two features and a slightly higher one for the goal feature, nonetheless, maintaining the general trend of the rewards.



(a) (b) (c)

Figure 5.3: (a) ILE increases with increasing occlusions on noise-free data and data with 30% noise, but less so for MMAP-BIRL. (b) ILE changes with increasing noise on occlusion-free data. (c) Average clock times for increasing occlusion at 30% noise. These were measured on a Ubuntu PC with quad-core Xeon CPU @ 3.2GHz and 76GB RAM. The vertical error bars denote one standard deviation.

We evaluate the performance of the methods under varying levels of occlusion (from 10% to 40%) while keeping the observation noise fixed at 0% and 30% (Fig. 5.3a), and for varying levels of noise (from 20% to 45%) without occlusion (Fig. 5.3b). Each data point is the mean of 10 batches with 10 trajectories each, exhibiting the corresponding level of occlusion and noise. A Gaussian prior ($\mu = -1$, $\sigma^2 = .5$) is used for MAP- and MMAP-BIRL. As we may expect, the ILE increases as learning becomes more challenging. This increase is worst for MAP-BIRL, since it does not model the noise and simply ignores the occlusions. Between the two methods that model both, MMAP-BIRL exhibits a much lower ILE, and does not increase as dramatically as HiddenDataEM, especially for the 30%-noise level. However, HiddenDataEM does run marginally faster than MMAP-BIRL in this toy problem, and both methods show run times that generally increase *linearly* as the occlusions increase. As we may expect, MAP-BIRL's run times remain mostly consistent as it does not reason about the uncertainty.

5.4.2 Robotic line sorting

The summative domain we consider is a realistic use-inspired robotic line sorting scenario where the 7 DOF cobot Sawyer (from Rethink Robotics, now Hahn Robotics) is tasked with sorting onions on a conveyor belt after observing a human perform the sort. Sawyer observes the human using a Kinect v2 RGB-D camera, and a trained YOLO v5 deep neural network model (Redmon et al., 2016) is used to detect and classify the onions as blemished or unblemished. The depth-camera frames are quantized into appropriate state variables by SA-Net (Soans et al., 2020). These state recognitions are shown in red text on the frames in Fig. 5.4.

(a) (b) (c) (d) (e)

Figure 5.4: (a–d) These frames show a human picking an onion, inspecting it, placing it after making a decision, and choosing the next onion in sequence. The red text appearing on the images is the state predicted by SA-Net. (e) An example occluded frame where SA-Net is unable to make a prediction. At this point, the expert could be placing the onion back on the conveyor or in the bin.

We model the onion-sorting domain as a discrete MDP as follows. The factored state is captured by 3 key variables yielding a total of 48 states: *Onion_location*: {on conveyor, hover location, in front of face, in bin}; *EndEffector_location*: {on conveyor, hover location, in front of face, in bin}; and *Prediction*:{good, bad, unknown}. The value "hover location" is a region of space just on top of the conveyor and "in bin" indicates just inside a bin that holds discarded onions. An example state where the onion is on the conveyor, the end-effector is in the hover location, and the prediction is blemished would be represented as (on conveyor, hover location, bad). Prior to inspection, every onion's condition is unknown. The sorter

performs one of five abstract actions: *claim new onion*, which shifts the sorter's focus to a new onion, *Pick up the onion*, which denotes the sorter grasping the onion; *Inspect after picking* the onion by rotating it and checking for blemishes; *Place onion on the conveyor* after it is picked; and *Place onion in the bin* after it is picked.
We utilize the following six Boolean feature functions to represent the human sorter's preferences:

- **Good onion placed on conveyor**(s, a) is 1 when a good onion is placed back on the conveyor,
- **Bad onion placed on conveyor**(s, a) is 1 when a bad onion is placed back on the conveyor,
- **Good onion placed in bin**(s, a) is 1 when a good onion is placed in the bin,
- **Bad onion placed in bin**(s, a) is 1 when a bad onion is placed in the bin,
- **Claim new onion**(s, a) is activated when a new onion is chosen if no onion is currently in focus,
- **Pick if unknown**(s, a) is 1 if the considered onion, whose classification is unknown, is picked.

The observer noise in this domain comes from YOLO sometimes misclassifying onions due to changing lighting conditions leading to SA-Net incorrectly identifying the state; we estimated this noise empirically to be approximately 30%. This makes the state estimation uncertain and is recorded as an *observation*. This forms the probabilistic observation model of the camera. Occlusions occur when another person inadvertently passes by in front of the camera during the recording and blocks a frame either partially or fully, which leaves SA-Net unable to ascertain a state value (as shown in Fig. 5.4e).

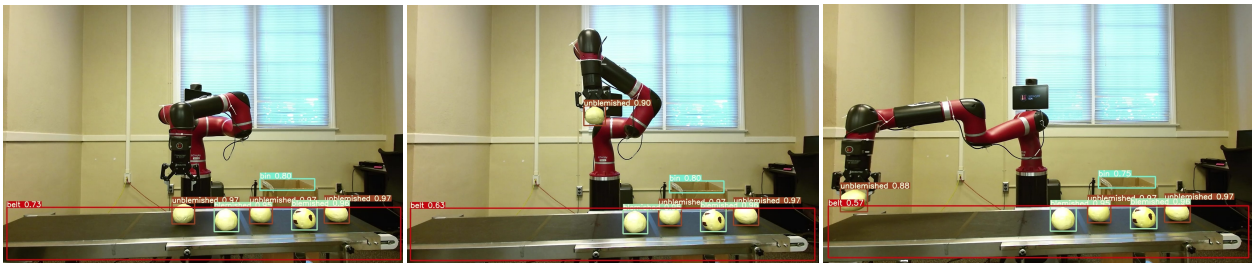


Figure 5.5: Snapshots of Sawyer sorting through the onions with bounding boxes detected in real-time by YOLO v5. Following the learned IRL policy throughout, in the first image, Sawyer approaches an onion based on 3D location estimated using rigid transforms. In the second image, Sawyer rotates the onion while holding it at a hover position to inspect for blemishes. In the third image, Sawyer makes a decision to place the onion back based on the policy action.

Results

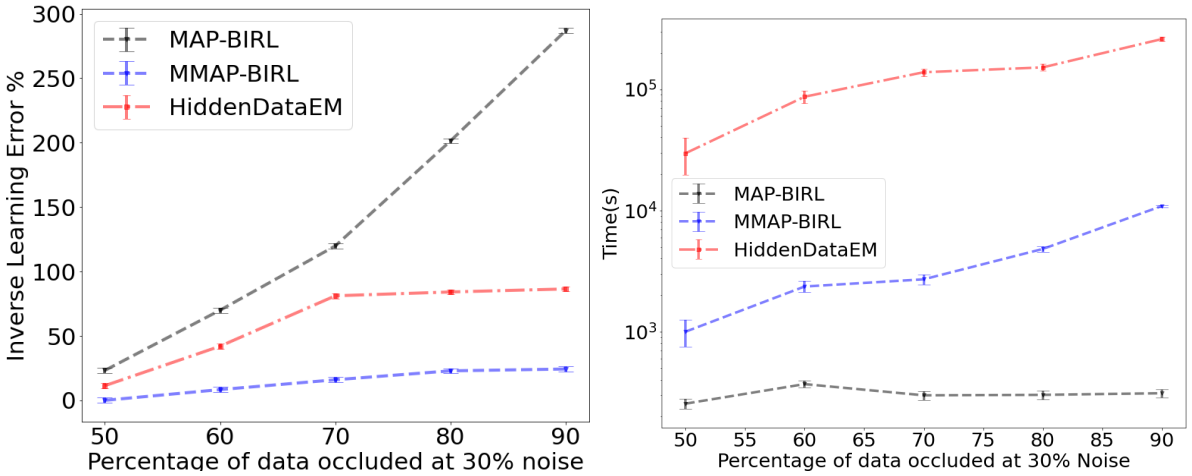
We recorded 12 trajectories from human demonstrations with an average of 4 state occlusions (due to the person blocking) in 135 state-action pairs per trajectory. Figure 5.6a compares ILE between MMAP-BIRL, the extended HiddenDataEM, and MAP-BIRL on these trajectories. For purposes of the evaluation,

greater occlusions were obtained by removing states in the trajectories at random. First, the degraded performance of MAP-BIRL due to noise and occlusion is evident from the significantly higher ILE shown by the technique. Notice that MMAP-BIRL continues to show a significantly lower ILE in comparison to HiddenDataEM on this larger domain. Equally important, it does so in much less time showing more than an order of magnitude in speed up, as is evident from Fig. 5.6b. Furthermore, the run times increase linearly in general for both methods as the rate of occlusion increases.

Method	(TP,FP,TN,FN)	Precision	Recall
MMAP-BIRL	$(23, 2, 18, 7)$	0.92	0.767
HiddenDataEM	$(16, 10, 15, 9)$	0.615	0.64

Table 5.1: Precision and recall of Sawyer physically sorting 50 onions on a conveyor using policies from rewards learned by MMAP-BIRL and HiddenDataEM, respectively. TP - True positive, FP - False positive, TN - True negative, FN - False negative.

We let Sawyer physically sort through 50 faux onions using the policies learned by both MMAP-BIRL and HiddenDataEM from the 12 recorded and processed trajectories [3]. Sawyer performs the sort by receiving the bounding boxes from YOLO, on which techniques such as orthogonal projection, pinhole camera model, and rigid transforms are used to obtain the coordinates of the onions in its 3D workspace. The robot picks up the onions and places them either in the bin or back on the conveyor after inspection. Figure 5.5 shows the bounding boxes detected in real-time by YOLO (Redmon et al., 2016).



(a) (b)

Figure 5.6: (a) MMAP-BIRL exhibits much better ILE performance with varying occlusion % at the estimated 30%-noise level. (b) MMAP-BIRL scales to this larger domain much better than the previous method with increasing occlusions and the same level of noise.

[3]Note that our MDP does not provide low-level control, which allows its variables to remain discrete. We follow a pipeline programming architecture in which the MDP's abstract actions map to motion planners in configuration space and plan in real-time.

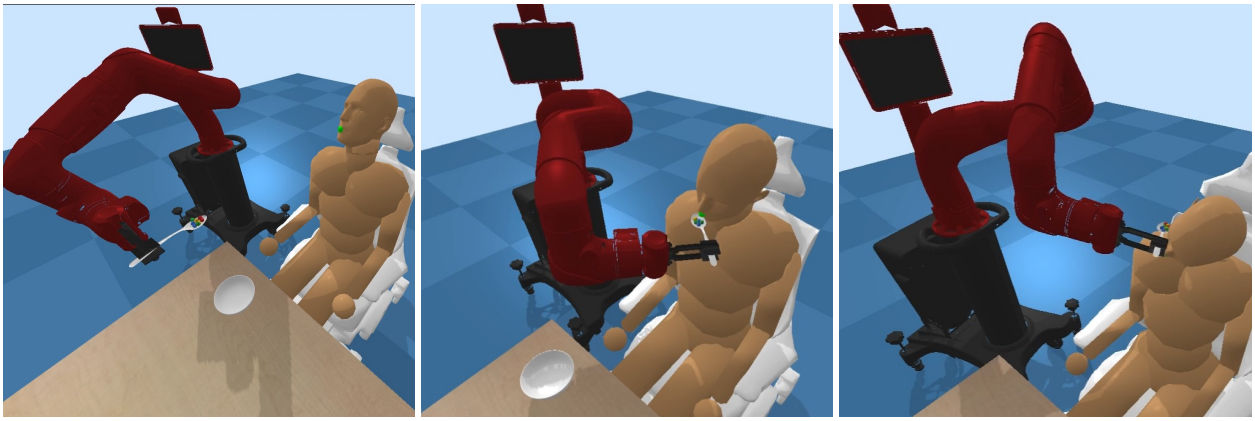
Sorting performance is measured using the domain-specific metrics of precision and recall where $precision = \frac{TP}{TP+FP}$ and $recall = \frac{TP}{TP+FN}$. Here, True Positive (TP) is the count of blemished onions in the bin, False Positive (FP) is the count of good onions in the bin mistaken as being blemished, True Negative (TN) is how many good onions remain on table, and False Negative (FN) is how many blemished onions remain on table mistaken as being good. From Fig. 5.1, we note that MMAP-BIRL exhibits a much better precision and recall compared to the previous HiddenDataEM method.

In summary, this use of MMAP in the context of IRL under uncertainty yields a new technique that significantly improves on the previous method in both the accuracy of the learned behavior and run time. We establish its usefulness toward robot learning on a use-inspired task in a complex environment.

Chapter 6

Cobots working in shared workspaces

In this chapter, we focus on the problems occurring when cobots synergistically work with humans in shared workspaces. We start by addressing interaction-level challenges and in the next chapter, we extend this framework to solve active collaboration-level challenges in human-robot teams.



(a) (b) (c)

Figure 6.1: Simulated Patient Assistance - Feeding. (a) Sawyer picking up the food from the table. (b) Feeding it to the human in the chair while the human also moves towards the spoon to cooperate. (c) Cobot encountering an adverse interaction by touching the human near the eye.

A noteworthy aspect of manipulation in shared workspaces is that without perfect coordination, agents (both robotic and human) might encounter adverse interactions. Conversely, certain interactions may in fact prove beneficial to task completion. For example, consider the scenario where a robotic agent is trying to feed a patient with special needs. The cobot uses a spoon to pick up food from the table and feed it successfully to the human seated beside it as shown in Fig. 6.1. A *necessary interaction* would be when the human and cobot move towards one another such that the spoon correctly touches the human's mouth.

On the contrary, the spoon or the cobot's arm hitting the human's face elsewhere would be undesirable and hence an *adverse interaction*.

Hand-coding a policy in such scenarios can be infeasible since the cobot's policy is a mapping of continuous joint torques to joint configurations of the cobot. A few previous methods addressing these interactions only models the robot's states and consider the human's impact on the environment can be fully captured through the transition function (L. Chen et al., 2021; S. Nikolaidis, Ramakrishnan, et al., 2015; S. Nikolaidis, Zhu, et al., 2017; S. Nikolaidis et al., 2013). However, this is insufficient in cases where each agent factors the other(s)' actions into their policy learning, since none of them follow a fixed policy during learning.

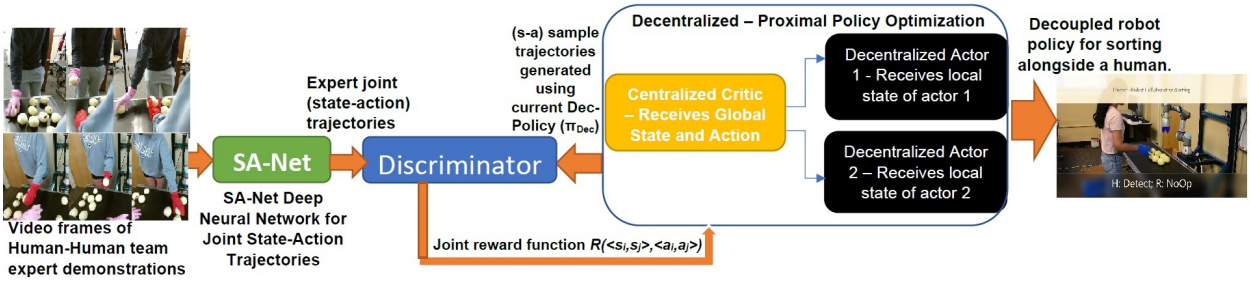


Figure 6.2: The end-to-end pipeline of Dec-AIRL. Time-synchronized raw video frames of a 2-person team sorting onions on a conveyor are input to SA-Net (Soans et al., 2020), which classifies the state and action. Each agent's state-action pairs are obtained through the camera opposite to them observing them sort. These are concatenated to form the global state-action pairs, which are used as training data for Dec-AIRL. Dec-AIRL uses a centralized critic and decentralized actors to learn a vector of policies. The decoupled robot policy is used to sort alongside a Human.

We present a novel method — Dec-AIRL that addresses a significant challenge in human-robot interaction (HRI) within shared workspaces, where the human's state is not perfectly observable by the robot. By introducing a decentralized approach combined with IRL (see Fig. 6.2), we intend to learn not only the task preferences but also conflict avoidance during adverse interactions, from joint expert demonstrations. Before explicating the experiment domains and how our results significantly improve upon existing state-of-the-art, we start by introducing our method - Dec-AIRL that learns a joint reward function, whose optimization in a Dec-MDP (Section 2.1.2) yields a vector of policies that simulate the observed behaviors of the two agents.

6.1 Method

We posit that a Dec-MDP is a better-suited model for pragmatic multi-agent IRL compared to the multi-agent Markov decision process (MMDP) or the Markov game, which have been utilized thus far. This is because an MMDP requires knowledge of the global state of the environment, which is impractical in the real world as it may include agent-specific attributes (e.g. joint angles) that may not be directly observable by the other agents.

6.1.1 IRL from Team Demonstrations

Following the same notations used in Section 2.1.2, let the joint experts' demonstration be given as:

$$\mathcal{X}^E = (\langle s_i^0, s_j^0\rangle, \langle a_i^0, a_j^0\rangle, \langle s_i^1, s_j^1\rangle, \langle a_i^1, a_j^1\rangle, ...\langle s_i^T, s_j^T\rangle, \langle a_i^T, a_j^T\rangle). \tag{6.1}$$

Dec-AIRL uses state-action samples drawn from both the expert (given) and the currently learned data, which is used by the discriminator to compute binary cross entropy (BCE) loss at each iteration. Let the expert's demonstrations be $\mathcal{X}^E$ (Eq. (6.1)) and trajectories drawn from the currently learned decentralized policy $\hat{\boldsymbol{\pi}}$ be denoted as $\hat{\mathcal{X}}$. Samples are drawn from $\mathcal{X}^E$ and $\hat{\mathcal{X}}$ at every iteration and used to minimize the distance between the expert's and learned state-action marginal distributions. The reward is updated using the discriminator's confusion (Gleave & Toyer, 2022) and the updated reward function is sent to the generator for forward-rollout policy learning. Dec-AIRL analytically represents the discriminator as: $D_{\boldsymbol{\theta}}(X) = \frac{e^{f_{\boldsymbol{\theta}}(X)}}{e^{f_{\boldsymbol{\theta}}(X)}+\pi(X)}$ and the reward update rule is given as

$$R_{\boldsymbol{\theta}}(X) \leftarrow \log D_{\boldsymbol{\theta}}(X) - \log(1 - D_{\boldsymbol{\theta}}(X)). \tag{6.2}$$

Eq. (6.2) when simplified yields: $f_{\boldsymbol{\theta}} - \log(\pi)$, which is the entropy-regularized reward formulation. The generator of Dec-AIRL uses a decentralized generalization of proximal policy optimization (Schulman et al., 2017) (Section 2.6) - Dec-PPO. Dec-PPO follows the same centralized training decentralized execution (CTDE) architecture as MA-PPO (C. Yu et al., 2021). Dec-PPO updates the value function as a squared-error loss:

$$L_t^{VF}(\omega) = \left(V^{\pi_\omega}(s^t) - \hat{V}_t^{targ}\right)^2.$$

where $\hat{V}_t^{targ}$ is the per-episode discounted reward-to-go and $V^{\pi_\omega}(s^t)$ is the predicted value of global state s^t and ω is the policy weight vector. For a dyadic system with agents i and j, the policy loss of agent i is given by

$$L_i^{CLIP}(\omega) = \mathbb{E}_{(s_i^t, a_i^t)\sim\pi_{\omega,i}}\left[min\left(\lambda_i^t\, A^{\pi_{\omega,i}}, clip(\lambda_i^t, 1-\epsilon, 1+\epsilon) \times A^{\pi_{\omega,i}}\right)\right].$$

$$\lambda_i^t = \frac{\pi_{\omega,i}(a_i^t|s_i^t)}{\pi_{\omega,i}^{old}(a_i^t|s_i^t)} \rightarrow \textit{ Importance sampling ratio.}$$

$L^{CLIP}(\omega)$ provides a pessimistic bound over the final objective by using a surrogate objective that picks the minimum of the clipped and unclipped objectives. By clipping the importance sampling ratio, the incentive of moving λ^t outside the interval $[1-\epsilon, 1+\epsilon]$ is removed. $A_{\pi_\omega,i}^t$ is calculated with respect to the reward estimates $R_{\boldsymbol{\theta}}(X)$ from Eq. (2.20). This clipped surrogate objective, combined with the policy entropy, handles the explore-exploit conundrum. This clipped surrogate objective, combined with the

policy entropy, handles the explore-exploit dilemma. The policy entropy loss is given as:

$$L_i^{ENT}(\omega) = \sigma H\left[\pi_{\omega,i}(s_i^t)\right].$$

where H is the policy entropy and σ is the entropy hyperparameter. The total loss is then given as:

$$L_i(\omega) = L_i^{CLIP}(\omega) + L_i^{ENT}(\omega). \tag{6.3}$$

The policy loss, entropy loss, and total loss apply analogously for agent j. At the end of training, the discriminator and generator return the learned common reward function and the converged vector of policies respectively. The environment state at each time step is observed (perfectly) through the robot's sensors and the corresponding action from the decentralized robot policy is performed by the robot. This execution continues until the task is completed or a goal state is reached.

6.2 Experiments

We evaluate Dec-AIRL on two robotic domains, a formative simulated patient assistance domain where the cobot is tasked with feeding a physically challenged human, and a summative realistic human-robot collaborative sorting domain where the human and cobot stand across from each other and sort onions on a line conveyor.

6.2.1 Simulated Patient Assistance

Our formative evaluation domain is one of many patient assistance domains available within the simulated open-source platform - *assistive-gym* (Erickson et al., 2020). Assistive gym consists of multiple environments involving a cobot assisting a patient with basic needs such as bathing, scratching, feeding, and dressing. These domains are built in the format of OpenAI-gym environments (Brockman et al., 2016a), and available in an MMDP format in version 0.1 and a Dec-MDP format in the latest version. For this work, we choose an environment that allows both the human and the robotic agents to actively collaborate and consists of both necessary and adverse interactions.

Setup

A Sawyer robotic arm (7 DOF cobot from Hahn Robotics previously Rethink Robotics) must safely and successfully feed a human, while the human seated in a chair, also moves towards the spoon to receive it, as shown in Fig. 6.1. Each agent – robot and human – follow a learned policy in order to successfully complete the task because the start state of neither agent is fixed and they do not follow predetermined paths to reach their respective goals. Therefore, each agent's action at every time step may be contingent on the other agent's action.

This task leads to several interactions between the agents: a *necessary interaction* in this scenario is where the robot brings the spoonful of food in contact with the human's mouth and the human moves towards the robot to receive it. On the other hand, the robot touching the human elsewhere, either with the spoon or its arm, is considered an *adverse interaction*. The resulting penalty is proportional to the force of contact. Fig. 6.1c shows an example of an adverse interaction where the robot is touching the human near the eye with the spoon while following a poorly learned policy.

The robot and human **state** is composed of the following values respective to the agent: *spoon - pose and orientation, human head - pose and orientation, spoon distance from human mouth, body joint angles, and the spoon force on the human*. The dimension of **actions** for the human and robot are 4 and 7 respectively based on the number of mobile joints. The global state S in the Dec-MDP would be a concatenation of the human's and robot's states.

Performance evaluation

For Dec-AIRL, we used 256 hidden nodes for the 2 hidden layers in both the g and h networks within the discriminator, a batchsize of 128, learning rate for both the discriminator and the actors as $3e-4$, discount factor and generalized advantage estimator (λ) both as 0.95, max gradient normalization as 0.5, and a random seed between 1 to 100. For Dec-PPO, we used the same hyperparameters as mentioned in gSDE (Raffin et al., 2022) since it has been tuned and tested on multiple Pybullet environments. The default hyperparameters were used for Co-GAIL and MA-AIRL.

Method	# steps	# adverse interactions	Total reward
Expert	24.5	4	148.6
MA-AIRL	131.2	12	108
Co-GAIL	164.5	19	91.4
Dec-AIRL (ours)	35	7	143.2

Table 6.1: Comparison of the number of steps, number of adverse interactions, and reward accrued per episode, averaged across 100 episodes. Dec-AIRL improves on both baseline methods significantly and is closest to the expert performance.

The expert trajectories were obtained from an "expertly" trained RL model. We compared the performance of Dec-AIRL with two known baselines, MA-AIRL and Co-GAIL. These were trained with perfectly observed global state-action pairs, while Dec-AIRL was trained with data that was only observable to each agent. All three algorithms were provided 10^5 steps of expert data $\mathcal{X}^E$ with each episode lasting a maximum of 200 time steps though usually less, and trained for 10^9 iterations. All three learned policies were deployed in a setting where the human state was perturbed by a value of 0.1 to depict a noisy perception by the robot for realism.

Table 6.1 compares the methods on the number of steps to complete the feeding, number of adverse interactions, and reward accrued per episode, averaged across 100 episodes. The results show that Co-GAIL and MA-AIRL converge to a joint policy that performs poorly in interacting states as compared

(a) (b) (c)

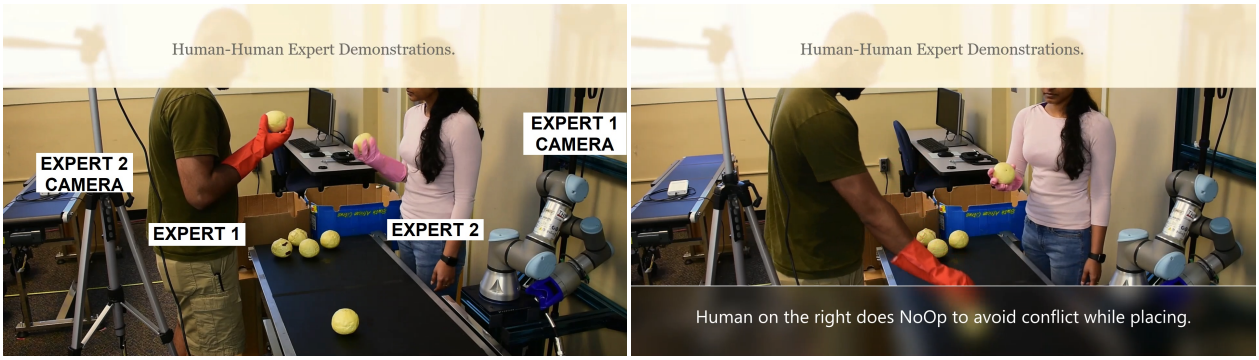
Figure 6.3: Comparison of the three learned policies implemented on an environment where the human's pose is not perfectly observed. (a) Co-GAIL policy and (b) MA-AIRL policy reach close to the mouth but don't manage to accurately feed the human and also end up dropping some food, while (c) Dec-AIRL policy takes a slightly longer path to the mouth than the expert but manages to feed the human correctly while avoiding adverse interactions.

to Dec-AIRL. Both MA-AIRL and Co-GAIL yield policies that prescribe actions conditioned on the global state. In other words, due to the choice of the underlying model (MMDP or Markov game), the robot is assumed to have access to the human's joint angles and by extension the accurate pose and orientation of the human head; it must also assess the force experienced by the human due to the spoon. However, these may not be observed perfectly by the cobot leading to suboptimal actions. As Dec-AIRL yields a decentralized policy for the cobot, which depends only on the parameters that it can observe, its performance is better aligned with the situation. Therefore, not only does Dec-AIRL generate fewer adverse interactions, but it also accrues a better reward per episode compared to the baselines. Figure 6.3 shows the comparison between the Dec-AIRL policy and the baseline policies when the robot is close to the human's mouth. Notice how with the baseline policies, the cobot reaches close to the mouth but is not able to reach the mouth accurately, meanwhile, with Dec-AIRL policy, the cobot is able to feed the human correctly, spills less food in the process and avoids adverse interactions.

6.2.2 Human-Robot Line Sorting

Our summative evaluation domain is a realistic use-inspired human-robot line sorting task where the physical cobot UR3e (from Universal Robots) is tasked with sorting onions along with a human on a conveyor belt after observing a human-human team perform the sort. A key challenge to a successful sort in the shared workspace is for both agents to avoid reaching for an onion on the table simultaneously or

placing a good onion back on the table in tandem after inspection — we refer to these situations as *adverse interactions*.



(a) (b)

Figure 6.4: Demonstration setup: Snapshots of a human-human (expert) demonstration where the two agents stand on opposite sides of a conveyor and sort faux onions. Observe that the camera placed behind expert 1 observes expert 2's state and actions, and vice versa. (a) Both agents inspect their respective onions during the sort. (b) Expert 2 does a No-Op action (pauses) to avoid conflict in a shared workspace while placing on conveyor.

We model the collaborative onion-sorting domain as a discrete Dec-MDP sans the reward function, which is learned. The factored local **state** for each agent is captured by 4 key variables yielding a total of 96 local states: *Onion_location*:{unknown, on conveyor, hover location, in front of face}; *EndEffector_location*:{on conveyor, hover location, in front of face, in bin}; *Prediction*:{unknown, good, bad} and *Interaction*:{true, false}. The value 'hover location' is a region of space just on top of the conveyor and 'in bin' indicates just inside a bin that holds discarded onions. The interaction variable is activated for states where the other agent also reaches out simultaneously to pick or place an onion on the table. An example global state where the robot's target onion is on the table, its end-effector is in the hover location, the onion prediction is bad, and the human's onion is in front of the face, the human's hand is also in front of the face, the prediction is good, and the agents are not in an interaction state, would be represented as ((on conveyor, hover location, bad, false),(in front, in front, good, false)). Prior to detection, each onion's location and status are unknown.

Each sorter performs one of six abstract **actions** in the Dec-MDP: *Detect*, which shifts the sorter's focus to a new onion on the table and also gives a preliminary prediction, *Pick up the onion* which is self-explanatory; *Inspect* the onion by rotating it and checking for blemishes; *Place onion on the conveyor* after it is picked and found to be unblemished; otherwise *Place onion in the bin* after it is picked. Both the agents have the same set of local state variables and local actions. [1] The Dec-MDP model is implemented as an environment in MA-Gym (Koul, 2019) for convenient dissemination.

[1]Note that our Dec-MDP model does not provide low-level control of the cobot's motion, which allows its variables to remain discrete. We follow a pipeline programming architecture in which the Dec-MDP's abstract actions map to motion planners in joint angles and plan in real-time.

Setup

Two depth cameras alongside each agent observe and record the other agent (human or robot) standing opposite (see Fig. 6.4). The time-synchronized, fused RGB-D frames from these cameras are quantized into appropriate state variables by SA-Net (Soans et al., 2020) while a trained YOLOv5 deep neural network model (Redmon et al., 2016) is used to detect and classify the onions as blemished or not, *which is imperfect*. These state recognitions are shown in red text on the frames in Fig. 6.5. For Dec-AIRL, we used the same hyperparameters as mentioned in Section 6.2.1 except for the size of hidden nodes in the g and h networks, which we modified to 128 hidden nodes for the 2 hidden layers, because the sorting Dec-MDP model is smaller and discrete compared to the assistive domain.

Figure 6.5: Image frames from a data set used to obtain the joint state-action trajectories $\mathcal{X}^E$ via SA-Net (Soans et al., 2020). The annotations on the images are the state-action predictions and the bounding boxes are generated by YOLOv5. The first frame is when the onion is on the conveyor and the human expert is about to pick. The second is when the human is inspecting and finding a good onion. The third is when the human is placing the onion back on the conveyor.

Performance evaluation

We recorded 10 rounds of sorting by 4 different teams of 2 humans each. Each round is a sort of a batch of 10 onions. This yielded 40 trajectories with an average of 650 frames per trajectory, containing a total of about $25,000$ joint state-action pairs. Similar to the previous simulated experiment, we compare Dec-AIRL's performance with the two baselines: Co-GAIL (C. Wang et al., 2022) and MA-AIRL (L. Yu et al., 2019). To obtain the global state on which the MMDP policy conditions its actions, the RGB-D

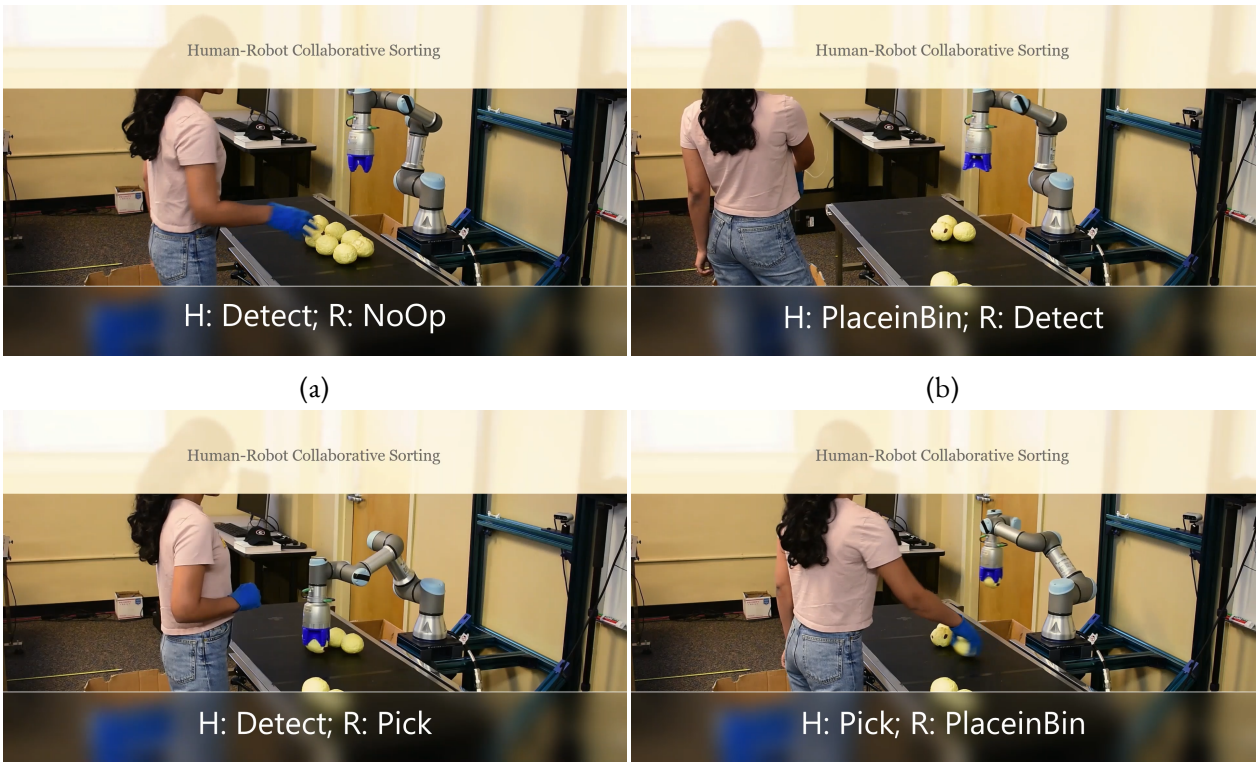


Figure 6.6: Frames from cobot-human collaboratively sorting onions on the conveyor using a Dec-AIRL learned policy. (a) The cobot recognizes that the human is about to pick and does No-Op to avoid an adverse interaction. (b) The cobot performs detect to choose an onion, while the human places her onion in the bad onion bin. (c) The human performs detect to choose an onion on the conveyor, while the robot picks the next onion. (d) The human picks the onion it chose while the robot places its onion in the bad bin.

frames from the two cameras are time synchronized through ROS at each time step. All three methods were run for 10^5 iterations, which resulted in near convergence. We compare their performance using the empirical *learned behavior accuracy* (LBA), a previously-used metric (Arora & Doshi, 2021b) that gives the proportion of prescribed actions from the learned policy for an agent which matches with that of the optimal policy. We extend this metric to measure for both agents. Table 6.2 shows the LBAs for increasing numbers of input state-action pairs, which yields agent behaviors of varying quality. Observe that, (a) all three methods' LBA improves as more training data is available as we may expect, and (b) Dec-AIRL's inversely learned behavior for the agents improves significantly on that learned by MA-AIRL and Co-GAIL for the various sets of input trajectories. A key reason for this improvement is that MA-AIRL and Co-GAIL learn policies, which prescribe actions that led to more adverse interactions compared to the actions from Dec-AIRL. For example, the best policies yielded 153 interactions for Co-GAIL and 96

interactions for MA-AIRL compared to 27 for Dec-AIRL across $2,304$ states of the sort. A key reason for this is that the human's blemished onion assessment, which is available during training as part of the global state, cannot be observed during execution. It may lead the cobot to pick an onion thinking that the human will discard its picked onion into the bin. However, if the noisy assessment has mislabeled the onion being inspected by the human as blemished, the human actually places it back on the table leading to a collision.

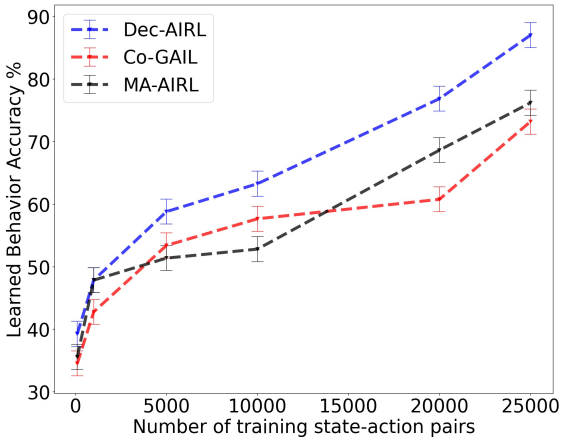


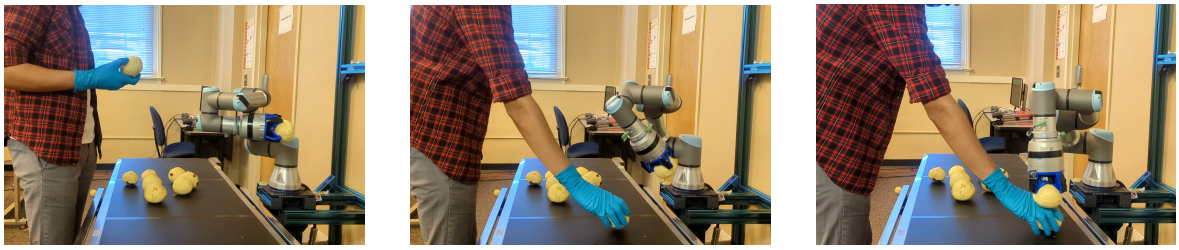
Table 6.2: The empirical LBA comparison between Dec-AIRL, MA-AIRL and Co-GAIL with varying levels of training data. 25000 states from SA-Net trajectories were used to obtain the robot action from these policies and compared to the ground truth robot action.

Method	(TP,FP,TN,FN)	Precision	Recall
Hum-hum	$(4, 1, 5, 0)$	0.8	1.0
Hum-cob (MA-AIRL)	$(3, 2, 3, 2)$	0.6	0.6
Hum-cob (Dec-AIRL)	$(3, 2, 4, 1)$	0.6	0.75

Table 6.3: Average precision and recall from 5 human-cobot sorting sessions with 10 onions sorted per session using policies learned via MA-AIRL and Dec-AIRL. As can be seen, the human-cobot team's performance with Dec-AIRL policy improves on MA-AIRL policy performance in terms of recall. TP - True positive, FP - False positive, TN - True negative, FN - False negative.

We deployed the best performing decentralized policy of one of the agents to control a physical cobot UR3e to guide collaboration with a human to sort faux onions. We ran 5 rounds of cobot-human sorting where multiple members of the lab not involved in this paper played the role of the human sorter. Our evaluation focuses on how well the cobot sorts in the team and whether it avoids adverse interactions; the human sorters follow a fixed behavior in sorting the onions.

The cobot performs the sort by receiving the bounding boxes from YOLO v5, upon which rigid transforms are applied to obtain the coordinates of the onions in its 3D workspace. The cobot and the human each pick up the onions and place them either in the bin or back on the conveyor table after inspection. We measure the team's sorting performance using the domain-specific metrics of precision and recall where *precision = TP/(TP + FP)* and *recall = TP/(TP + FN)*. Here, true positive (TP) is the count of blemished onions in the bin, false positive (FP) is the number of good onions in bin, true negative (TN) is the number of good onions remaining on table, and false negative (FN) is the count of blemished onions remaining on table. From Table 6.3, we note that Dec-AIRL leads to the cobot-human team exhibiting recall that is better than that of MA-AIRL but still somewhat far out from the precision and recall of the human-human team, which is indicative of room for improvement. Figures-6.6 and 6.7 show some frames from the cobot-human collaborative sort when the cobot is using the policy learned from



(a) (b) (c)

Figure 6.7: As MA-AIRL exhibits a higher LBA compared to Co-GAIL, we selected MA-AIRL as the baseline to run the sorting experiment with the human teammate. (a) Both the cobot and human are engaged in inspection. (b) The human decides to place the onion back on the conveyor because he assesses it as a good onion. Meanwhile, the cobot is also placing its onion back because it wrongly estimated the global state as it cannot see the onion from the human's perspective. This particular onion had a blemish not visible while on the conveyor, which led the cobot to (erroneously) reason that the human will discard the onion. (c) Both agents place their onion on the conveyor thereby encountering an adverse interaction.

Dec-AIRL and MA-AIRL, respectively. The latter clearly shows an interaction caused by the baseline policy.

In summary, our method Dec-AIRL learns a decentralized vector of policies through the centralized training and decentralized execution paradigm that allows the robot to factor human actions into its decision-making without knowing the human state attributes perfectly. Thus, Dec-AIRL takes us one step closer to realizing effective HRC in pragmatic scenarios like manufacturing industries.

Chapter 7

Open Human-Robot Collaboration

In the ever-evolving landscape of robotics and AI, the collaboration between humans and robots has become a desideratum. This partnership leverages the distinctive and often complementary strengths of both humans and robots (Mukherjee et al., 2022; Villani et al., 2018) to provide an impetus for work, innovation, and societal advancement. Robots, possessing sensory perception and intelligent decision-making capabilities, serve as essential collaborators in the quest for efficiency, precision, and creativity across various applications. The HRC paradigm doesn't aim to replace human labor; instead, it focuses on augmenting it, enhancing capabilities, and empowering humans to address challenges that were previously deemed insurmountable.

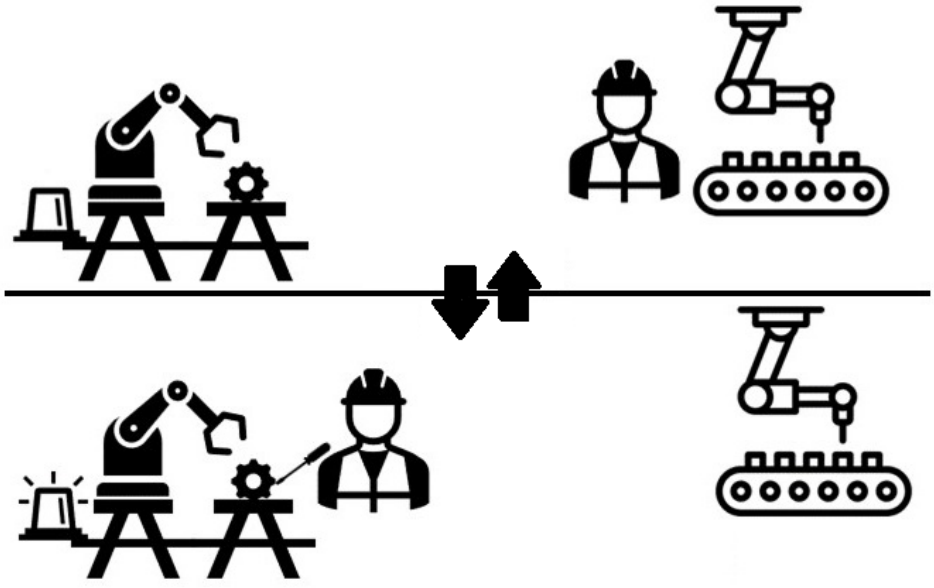


Figure 7.1: Open HRC - Illustration of a factory floor scenario where a collaborative robot and a human operator are working on two different tasks independently. The robot reaches a point where it requires human assistance, signals, and calls a human operator. The human operator arrives, assists the robot in completing the collaborative sub-task, and then leaves to attend to other tasks.

In order for humans and robots to collaborate in a shared workspace, both types of agents (human and robotic) must factor in the behavior of the other during decision-making. This scenario naturally lends itself to a multiagent system (Ferber & Weiss, 1999) that can be used to model and learn behavioral policies for the agents. So far in this dissertation, we have talked about a number of challenges related to robotic learning, especially in real-world scenarios. However, this chapter handles an entirely new direction in human-robot collaboration - *agent openness*.

Much prior research on human-robot collaboration (HRC) often assumes a predetermined and unchanging set of agents throughout the task. However, in certain task domains, the efficiency of the human-robot interaction could be enhanced if the human joins the system when needed and exits upon completing their role in the task. For example, envision a factory floor setup (see Fig. 7.1) where a collaborative robot initiates the assembly of a part and, on requiring assistance, signals the human for help through an alarm or flashing light. The human factory worker arrives, assists the robot with the collaborative subtask, and then leaves to attend to other tasks. We term such a system, where human or robotic agents can enter and leave the task at any point in time, as an *open* HRC (OHRC). Notice that OHRC systems (OHRCS) may be seen as a type of the broader open agent systems (Eck et al., 2023) that exhibit agent openness.

7.1 Open Human-Robot Collaborations

Open HRC (OHRC) refers to the paradigm of modeling HRC with agent openness (AO), where agents (human and robotic) can join or leave the task at any point. In this paper, we consider only AO, however, certain HRC scenarios may also demand type openness (TO) - where agents' types change dynamically during the task, and task openness (TaO) - where the goals change with task progress. In OHRC, new agents join the task either when called in by one of the currently active agents or on their own volition, and any agent can choose to leave the task at any point. A noteworthy feature of OHRC is that while the human is absent from the environment, the robot could switch to industrial mode of operation, thereby working at its maximum speed to optimize throughput and switching back to collaborative mode when the human enters to safely team-up with them.

Before we go into the specifics of our model and our inverse RL method, let's take a bird's eye view of *open* HRC and its challenges. In OHRC, new agents join the task either when called in by a currently active agent or of their own volition, and any agent can choose to leave the task at any point. To explicate this, we start by classifying OHRC broadly into two categories:

- **Collaboration-for-efficiency (CE):** Whereas tasks can be completed by the robotic agent alone, collaborating with human(s) could improve task efficiency. An example of CE would be tasks like line-sorting and packing (Sengadu Suresh et al., 2023), where both precision and recall need to be maximized.
- **Collaboration-due-to-requirement (CR):** This pertains to tasks that contain subtasks that a robotic agent cannot complete alone. An example of CR would be when the task involves highly dexterous manipulation or when the subtask requires multiple agents collaborating simultaneously to complete (see Fig. 7.2b).

These two categories will be illustrated in the context of each domain of interest in the following sections.

7.1.1 Use-Inspired Domains for Motivation

In order to tether the concepts of OHRCS to realistic examples, we provide two motivating domains: Collaborative robots in manufacturing, and assistive robotics. In both domains, we discuss cases where CE and CR may be applicable and how they can be addressed.

Collaborative Robots in Manufacturing

Consider the task of collaborative furniture assembly as shown in Fig. 7.2a. The goal of this task is to assemble a table consisting of multiple parts: base, leg-support1, leg-support2, leg1, leg2, two screws for each leg-support to connect them to the base, and two screws for each leg to screw them into their respective supports, as shown in Fig. 7.2c. The task can be completed in multiple valid orders. For instance, one may position both leg-supports on the base and screw them in before positioning their corresponding legs and screwing the legs into their respective supports. Alternatively, one may position leg-support1, screw it into the base, place the leg1, screw it into the leg-support1, and analogously repeat the sequence for the other parts to complete the assembly. Notice that the simple positioning actions can be done independently by the robot, while the screwing action requires the assistance of a human.

- **CE:** In this furniture assembly example, the robot can independently position all the parts on the base at their respective locations. However, if a human is present in the environment, they can assemble parts in parallel, thus completing the positioning sooner. Nonetheless, availing a human's help for trivial positioning tasks is an ineffective use of their time and energy. Therefore, it is imperative to optimize the available resources to enhance task efficiency while minimizing human effort. Only if the task is time-sensitive or safety-critical can the robot engage the human's services from the beginning to speedily complete it.
- **CR:** Once the positioning of the supports is complete, the supports need to be screwed into the base before assembling the legs of the table. This screwing subtask requires dexterous manipulation and another agent to hold the part aligned with the screw holes. Even if the robot could screw in the part, since the part requires another hand to hold it in place, collaborating with a human is crucial. However, as mentioned before, since the assembly can be performed in multiple orders, the robot, in addition to learning the optimal method to assemble the parts, needs to learn when to call the human for help, and how to collaborate with them.
- **Other complexities:** Since human agents have limited time and energy, they may decide to leave after completing a subtask to focus on other tasks. Specifically, in the furniture assembly case, if the robot calls the human after positioning the first support, the human may arrive to assist with screwing the support in but may leave after that. In that case, the robot must call the human again after placing the second support. Alternatively, the human may position the first leg alongside its support while the robot positions the second support. Since these subtasks may take different durations, if one agent finishes their subtask sooner and needs assistance, the other may choose to complete their current subtask before assisting them or disregard their current subtask and assist them immediately. Lastly, there may be a case where the human is forced to wait a few timesteps due to the unavailability of valid subtasks; leaving and

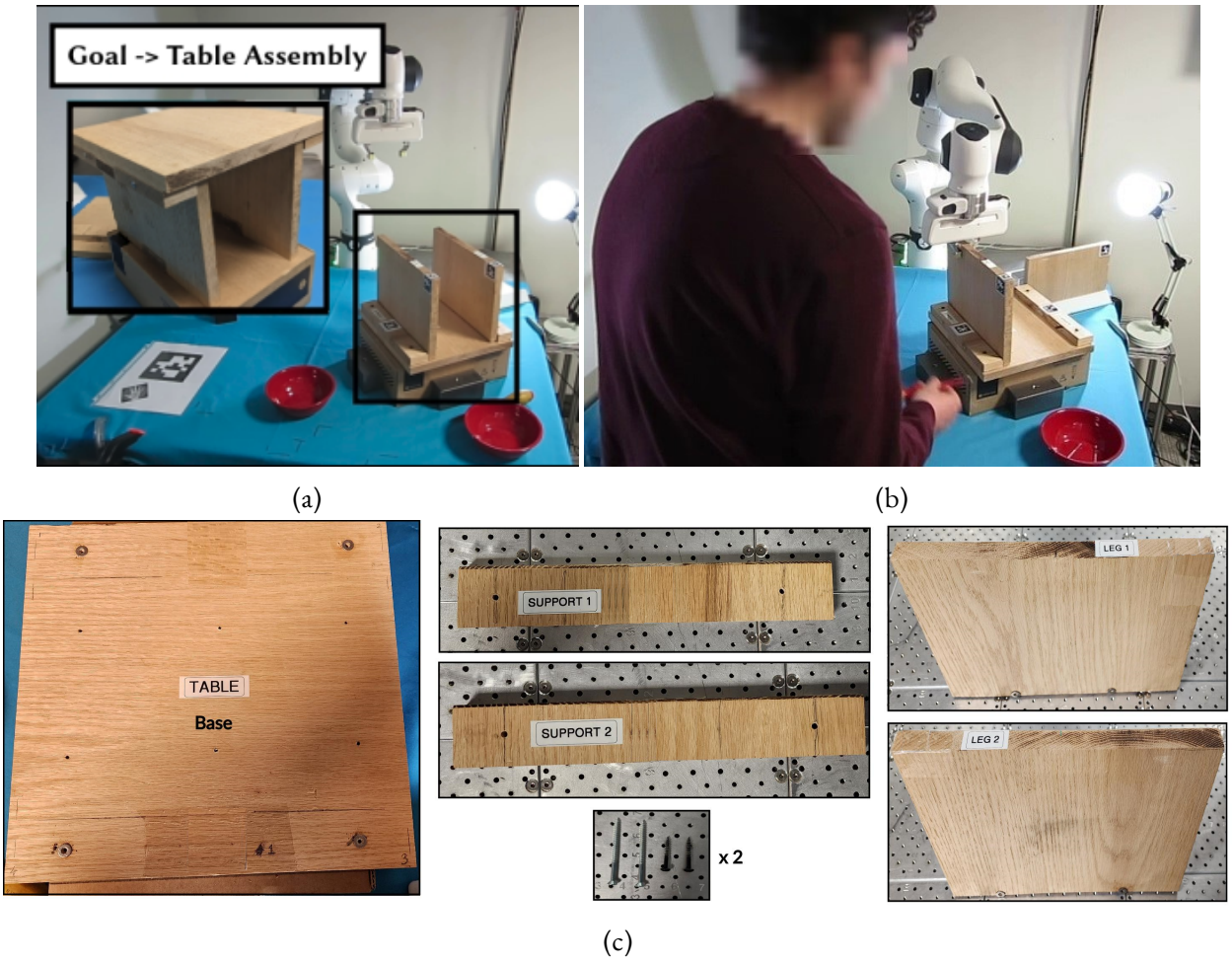


Figure 7.2: A wooden table assembly task involving placing and screwing actions. (a) The assembled table. (b) The human screws one of the legs in while the robot holds it in place. (c) The base, supports and screws, and legs of the table.

rejoining at that point may be inefficient. For example, consider the case where the only subtasks left are positioning and screwing the last leg in place. Until the robot positions the leg, the human cannot screw it in, and exiting at this point only to rejoin shortly after may not be worthwhile.

Assistive Robotics

With the recent advances in robotics, assistive robots have become an increasingly necessary tool to provide care and assistance to the elderly and people with motor impairments. An assistive robot may aid them in daily activities such as bed-bathing, itch-scratching, feeding, etc., as shown in Fig. 7.3. As such, usually, one robot interacts with the subject to aid them; however, certain subtasks, such as bed-bathing, may require

collaboration with additional agents to be accomplished. For example, to assist a patient in bathing, the robot may need to stand them up or turn them over and may require a hand on either side to lift them and turn them around. Evidently, this is an OHRC, and modeling it as a closed system may diminish the quality of patient care.

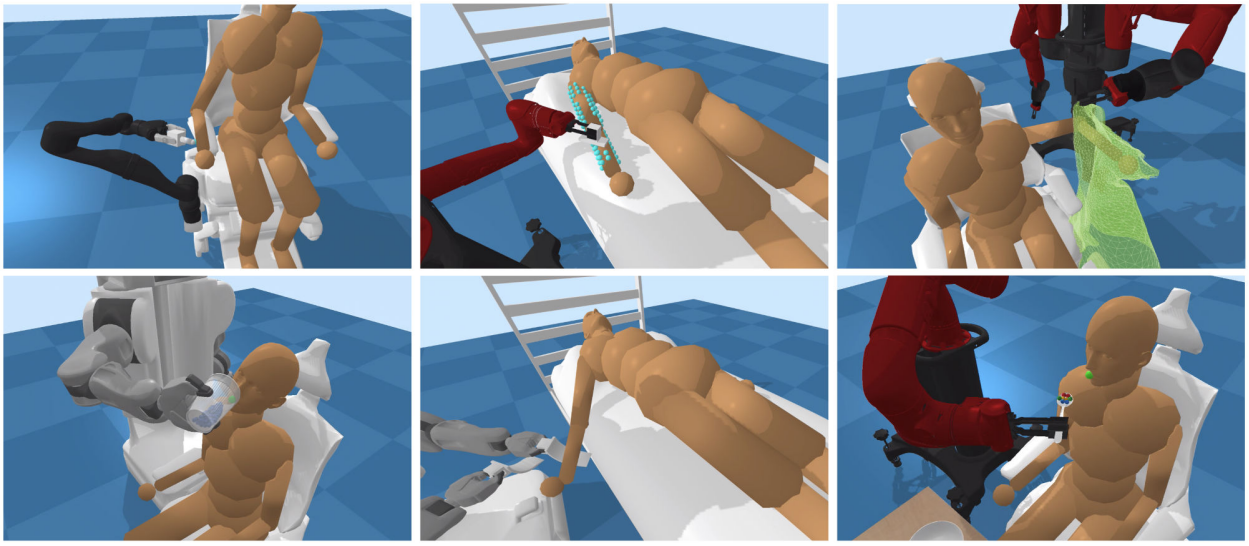


Figure 7.3: Illustration of a few assistive tasks that could be performed by a collaborative robot such as itch-scratching, changing, feeding, bed-bathing, etc. Image credit: Assistive Gym (Erickson et al., 2020).

- **CE:** In certain cases, multiple subtasks could be performed in parallel to enhance assistance, for example, the person could be feeling cold while the robot is engaged in feeding them. In this case, the robot could either stop feeding them to cover them with a blanket or finish feeding them before covering them with a blanket. Since these subtasks could be completed in parallel with another agent on the scene, a new agent (human or robotic) could arrive, cover the subject with a blanket, and then leave to attend to other tasks.
- **CR:** Alternatively, the robot could accomplish all other tasks on its own but when it reaches the bed-bathing scenario, as mentioned before, the subject may need to be held on both sides and flipped over. In this case, the robot may call another human or robot for assistance. Once the subject has bathed, the new agent could leave to attend to other tasks.

7.1.2 Novel Challenges of OHRC

In order to systematically characterize the challenges in OHRC, we classify them into three overarching categories:

1. **Modeling challenges:** The first important challenge to address in OHRCS is establishing an efficient way to model openness in HRC. Since HRC by definition is collaborative, typically the model has a single reward function designed toward the common goal. However, since the agents involved in

HRC cannot perfectly observe all the attributes of the other agents (for example, human joint angles), the model has to support a decentralized execution approach. Models such as decentralized Markov decision processes (Dec-MDP) (Goldman & Zilberstein, 2004) can be extended to model agent openness. The global state and global action at a given timestep can be formed using the pooled states and pooled actions of the *agents active at that timestep*, thus incorporating openness. This would require the learning paradigm to accept inputs of states and actions that vary in size at every timestep based on the currently active agents. This global state, global action, and a common reward function can be used for training, to obtain decentralized policies (one for each agent that participated in the task during training). These decentralized policies would map the agent's local state to their local action and factor in the current team of that agent, to learn the appropriate action mapping.

Certain modeling challenges of HRC may also manifest in OHRC, such as designing an appropriate reward function. However, this becomes more challenging in OHRC since multiple combinations of teams may accomplish the task similarly. Additionally, AO may be stochastic, where the agent may enter and exit the task with a certain probability, further increasing the decision-making complexity. Other confounding factors like sensor noise may render determining if an agent is present or absent, uncertain.

Finally, a critical challenge in navigating a shared workspace is recognizing and correctly modeling the *necessary interactions*. For instance, in the furniture assembly domain, the screwing subtask requires the robot to hold the part in place while the human screws it in. Such interactions can be termed *necessary* interactions. On the contrary, if both the robot and human are positioning parts in parallel and both agents try to position a part at the same location simultaneously, these can be termed as *adverse* interactions.

OHRC further complicates such interactions due to agent openness. The robot needs to factor in the human's presence or absence first and then decide on a strategy to navigate potential interactions. Therefore, designing a model that can appropriately handle the challenges and complexities of OHRC is an avenue for future research.

2. **Behavioral challenges:** Learning behavioral policies for agents involved in OHRC is an ambitious task. These agent policies have to account for: the currently active agents in the environment, actively collaborating agents, when new agents are needed, and when the current agent(s) must exit the task. A popular approach used in multiagent reinforcement learning (Albrecht et al., 2023) is centralized training and decentralized execution, which could be extended to consider agent-openness, to learn OHRC policies. Additional factors such as partial observability and stochasticity may complicate behavioral policy learning by providing noisy and uncertain feedback about the agents and the environment.

   Agent-based modeling techniques (Gilbert, 2019; S. Nikolaidis, Forlizzi, et al., 2017; V. V. Unhelkar, 2020), Theory of Mind (Gmytrasiewicz & Doshi, 2005; Leslie et al., 2004) or Prospect Theory (Kwon et al., 2020) could be incorporated to learn the entry and exit pattern of an agent, based on prior experience, their current energy level, level of rationality, etc. Such latent decision-making factors may also manifest in other forms of openness, such as task-openness (TaO) or type-openness (TO). For

example, the human, upon joining the task may decide to focus on a different subtask than the one initially intended. The robot must be capable of recognizing and adapting to such TaO. Furthermore, humans may modify their behavior mid-task by, say, becoming less collaborative due to fatigue, and such TO must be handled by the robot's policy.

3. **Implementation challenges:** While all the modeling and learning challenges are quite complex, implementing learned policies in the real world and achieving seamless collaboration in OHRC may be the most challenging of all. Depending on the robot and the task itself, several unforeseen challenges may arise. For example, physical limitations such as dynamic environmental occlusions or abruptly malfunctioning sensors may make it difficult to assess the presence of a particular agent, further complicating the decision to call a new agent.

   Furthermore, due to limited signal strength or environmental distractions, agents may not receive the call to join the task (say a loud machine on the factory floor prevents the human from noticing the robot's call for help). Humans may also decide to abruptly exit the task due to an emergency, or their shift ending, which may throw off the learned policy since it may not have happened during training. Sensing such abnormalities and adapting accordingly is one of the major implementation challenges. A data-driven model that learns from human-human team experiences (e.g. inverse RL (Arora & Doshi, 2021a) or imitation learning (Hussein et al., 2017)) combined with iterative improvement, based on live feedback (such as active learning (Settles, 2009)) could mitigate such difficulties.

7.2 Our multiagent model and IRL method for OHRCS

To address the aforementioned challenges of OHRC, we adopt an inverse reinforcement learning (IRL) (Arora & Doshi, 2021b) based approach. In this section, we describe our contributions:

1. We present oDec-MDP, a novel multiagent decision-making framework to model agent openness in OHRCS.

2. We develop a novel IRL technique - oDec-AIRL that generalizes our decentralized IRL method - Dec-AIRL (Sengadu Suresh et al., 2023), to learn the underlying reward function and its corresponding vector of policies using the oDec-MDP as the behavioral model.

We validate our contributions on two experimental domains: a popular simulated urban firefighting (Chandrasekaran et al., 2016; Eck et al., 2020; Kakarlapudi et al., 2022) and a physical human-robot collaborative furniture assembly.

7.2.1 Open Collaboration Model

oDec-MDP generalizes Dec-MDP to model agent openness in a decentralized, collaborative setting. Formally, our oDec-MDP can be defined as:

$$\text{oDec-MDP} \triangleq \langle Ag, C, \boldsymbol{S}, \boldsymbol{A}, \Gamma, T, R, \rho \rangle$$

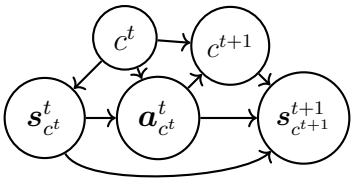


Figure 7.4: The oDec-MDP graphical model for two timesteps t and $t+1$. Given the collab team ID c^t at timestep t, $\boldsymbol{s}^t_{c^t}$ is formed by combining the local states of all agents in c^t. All agents' local actions from c^t combined form $\boldsymbol{a}^t_{c^t}$, which leads to c^{t+1}, given c^t. c^{t+1}, $\boldsymbol{a}^t_{c^t}$ and $\boldsymbol{s}^t_{c^t}$ together lead to the next state $\boldsymbol{s}^{t+1}_{c^{t+1}}$ at time $t+1$.

- Ag is the finite set of all agents and $|Ag| = N$ is the maximum number of agents;
- $C : \mathcal{P}(Ag) \to \mathbb{N}$ assigns a unique number identifier to each collaborating team and $\mathcal{P}$ denotes the powerset excluding the empty set. For convenience, we let C also denote the set of all assigned identifiers;
- Global state space $\boldsymbol{S} = \bigcup_{c=1}^{c=|C|} S_c$ where $c \in C$ and S_c denotes the set of states of the team identified by c;
- Global action space $\boldsymbol{A} = \bigcup_{c=1}^{c=|C|} A_c$ where $c \in C$ and A_c denotes the set of *joint* actions of the team identified by c. For instance, if team c involves agents i and j whose action sets are A_i and A_j, respectively, then $A_c = A_i \times A_j$;
- Team transition model $\Gamma : C \times \boldsymbol{A} \times C \to [0,1]$ gives the distribution of the new teams given the current team and action letting agent(s) enter or exit;
- $T = \{T_c, T'_c \,|\, c = 1, 2, \ldots, |C|\}$, where *intra-team* state transition model $T_c : S_c \times A_c \times S_c \to [0,1]$ gives the distribution over the team's next state and *inter-team* state transition model $T'_c : S_c \times C' \times S_{c'} \to [0,1]$ gives the distribution over the next team's state. Both are available for all $c, c' \in C$;
- Common reward function shared by all agents in each team c, $R_c \triangleq R(S_c, A_c, c)$ and $R_c : S_c \times A_c \to \mathbb{R}$;
- Start state and team prior distribution $\rho : S \times C \to [0,1]$.

We model the collaboration using the oDec-MDP framework [1] and an open teamwork trajectory of length $\mathcal{T}$ contains the collaborating team ID, team state, and team action at each time step:

$$X^E \triangleq \left(\langle c, \boldsymbol{s}_c, \boldsymbol{a}_c\rangle^1, \langle c, \boldsymbol{s}_c, \boldsymbol{a}_c\rangle^2, \langle c', \boldsymbol{s}_{c'}, \boldsymbol{a}_{c'}\rangle^3 \ldots \langle c'', \boldsymbol{s}_{c''}, \boldsymbol{a}''_c\rangle^{\mathcal{T}}\right).$$

Notice from the trajectory that the starting team with ID c persists for the first two time steps followed by a change to team c'. If the team with ID c at time step $t = 1$ is a dyad with agents i and j, then the policy $\boldsymbol{\pi}$, the team state $\boldsymbol{s}^t_c$, and team action $\boldsymbol{a}^t_c$ are vectors of the two individual agents' policies, their

[1] Note that our oDec-MDP is also locally fully observable (Goldman & Zilberstein, 2003).

partial states, and their actions respectively:

$$\boldsymbol{\pi}_c \triangleq \langle \pi_i, \pi_j \rangle;\ \boldsymbol{s}_c^t \triangleq \langle s_i^t, s_j^t \rangle;\ \text{and}\ \boldsymbol{a}_c^t \triangleq \langle a_i^t, a_j^t \rangle.$$

The likelihood of the first two time steps of X^E is obtained using the parameters of oDec-MDP as:

$$\begin{aligned}
&\Pr(c, \boldsymbol{s}_c^1, \boldsymbol{a}_c^1, c, \boldsymbol{s}_c^2, \boldsymbol{a}_c^2) = \Pr(c, \boldsymbol{s}_c^2, \boldsymbol{a}_c^2 | c, \boldsymbol{s}_c^1, \boldsymbol{a}_c^1) \Pr(c, \boldsymbol{s}_c^1, \boldsymbol{a}_c^1) \\
&= \Pr(c, \boldsymbol{s}_c^2, \boldsymbol{a}_c^2 | c, \boldsymbol{s}_c^1, \boldsymbol{a}_c^1) \Pr(\boldsymbol{a}_c^1 | c, \boldsymbol{s}_c^1) \Pr(c, \boldsymbol{s}_c^1) \\
&= \Pr(\boldsymbol{a}_c^2 | c, \boldsymbol{s}_c^2) \Pr(c | c, \boldsymbol{a}_c^1) \Pr(\boldsymbol{s}_c^2 | \boldsymbol{s}_c^1, \boldsymbol{a}_c^1) \Pr(\boldsymbol{a}_c^1 | c, \boldsymbol{s}_c^1) \Pr(c, \boldsymbol{s}) \\
&= \Pr(a_i^2 | c, s_i^2) \Pr(a_j^2 | c, s_j^2) \, \Gamma(c, \boldsymbol{a}_c^1, c) \, T_c(\boldsymbol{s}_c^1, \boldsymbol{a}_c^1, \boldsymbol{s}_c^2) \Pr(a_i^1 | c, s_i^1) \Pr(a_j^1 | c, s_j^1) \Pr(c, \boldsymbol{s}_c^1) \\
&= \underbrace{\pi_i(a_i^2 | c, s_i^2)}_{\text{Policy of i at t=2}} \underbrace{\pi_j(a_j^2 | c, s_j^2)}_{\text{Policy of j at t=2}} \underbrace{\Gamma(c, \boldsymbol{a}_c^1, c)}_{\text{Team transition}} \underbrace{T_c(\boldsymbol{s}_c^1, \boldsymbol{a}_c^1, \boldsymbol{s}_c^2)}_{\text{intra-team state transition}} \underbrace{\pi_i(a_i^1 | c, s_i^1)}_{\text{Policy of i at t=1}} \underbrace{\pi_j(a_j^1 | c, s_j^1)}_{\text{Policy of j at t=1}} \underbrace{\rho(c, \boldsymbol{s}_c^1)}_{\text{given prior}}.
\end{aligned} \tag{7.1}$$

A locally fully observable Dec-MDP lets each agent's policy condition its action on the agent's partial view of the state. Next, we show how the likelihood is obtained for the second and third time steps of X^E when the team changes. The derivation of the likelihood proceeds analogously to the above steps:

$$\begin{aligned}
&\Pr(c, \boldsymbol{s}_c^2, \boldsymbol{a}_c^2, c', \boldsymbol{s}_{c'}^3, \boldsymbol{a}_{c'}^3) \\
&= \pi_i(a_i^3 | c', s_i^3) \, \pi_j(a_j^3 | c', s_j^3) \, \Gamma(c, \boldsymbol{a}_c^2, c') \, T_c'(\boldsymbol{s}_c^2, c', \boldsymbol{s}_{c'}^3) \, \pi_i(a_i^2 | c, s_i^2) \, \pi_j(a_j^2 | c, s_j^2) \, \rho(c, \boldsymbol{s}_c^2).
\end{aligned} \tag{7.2}$$

The key difference between Eqs. 7.1 and 7.2 is that the latter involves the inter-team transition function T_c' due to the change of team from time step $t = 2$ to $t = 3$. For the sake of completeness, we also show below how the value of a given team state at timestep t, $\boldsymbol{s}_c^t$, is obtained, which defines the value function of our oDec-MDP:

$$\begin{aligned}
V(\boldsymbol{s}_c^t) &= \max_{\boldsymbol{a}_c^t} \mathbb{E}_{c', \boldsymbol{s}_{c'}^{t+1}} \left[R(\boldsymbol{s}_c^t, \boldsymbol{a}_c^t, c) + \gamma V(\boldsymbol{s}_{c'}^{t+1}) | \boldsymbol{s}_c^t, c \right] \\
&= \max_{\boldsymbol{a}_c^t} \ R(\boldsymbol{s}_c^t, \boldsymbol{a}_c^t, c) + \gamma \sum_{C'} \sum_{\boldsymbol{S}_{c'}} \Pr(c', \boldsymbol{s}_{c'}^{t+1} | c, \boldsymbol{s}_c^t, \boldsymbol{a}_c^t) \, V(\boldsymbol{s}_{c'}^{t+1}) \\
&= \max_{\boldsymbol{a}_c^t} \ R_c + \gamma \sum_{C'} \sum_{\boldsymbol{S}_{c'}} \Gamma(c, \boldsymbol{a}_c^t, c') \, T_c'(\boldsymbol{s}_c^t, c', \boldsymbol{s}_{c'}^{t+1}) V(\boldsymbol{s}_{c'}^{t+1}).
\end{aligned}$$

The discriminator $D_{\boldsymbol{\theta}}(X)$ of oDec-AIRL learns a common reward function R_c contingent on c, s, and a. This common reward function is then used by oDec-PPO to learn a vector of policies (one for each agent). oDec-AIRL minimizes the reverse KL divergence (Ghasemipour et al., 2020b) between the learner's and expert's marginal team*id*-state-action distribution $KL(\mathrm{P}_\pi(c, s, a) || \mathrm{P}_{exp}(c, s, a))$.

7.3 Experiments

In this section, we consider two domains, a popular simulated toy domain used in previous open multiagent systems literature - urban firefighting (Chandrasekaran et al., 2016; Eck et al., 2020, 2023; He et al., 2023; Kakarlapudi et al., 2022), and a realistic dyadic collaborative human-robot furniture assembly domain. In both domains, we focus on the agent being called in at the right time. Nonetheless, our method applies analogously to agents exiting the task. In order to establish the efficacy of an open system over a closed one, we use an ablation study to compare the learned behavior of oDec-AIRL with that of Dec-AIRL on the two domains [2].

7.3.1 Urban Firefighting

In this domain, the goal is for the agent(s) to extinguish fires within a 3x3 grid as efficiently as possible. There are three active fires — a large, medium, and small fire as shown in Fig. 7.5 with intensities 0.9, 0.6, and 0.3 respectively at the beginning. Each firefighter has 4 cardinal direction actions, a CallAgent action, and an Extinguish action. A CallAgent action calls one agent at a time (up to 3 agents) and the Extinguish action at a fire location reduces its intensity by 0.1. Each agent's local state contains - their location, the fire locations and intensities, and the number of teammates at their location. Agents need to decide whether or not to call another agent to extinguish the fires effectively. The challenge in this domain is that the learned reward function must maintain a delicate balance between the cost incurred by having additional agents present vs the reward accrued by extinguishing fires sooner. Note that the total number of steps per episode is slightly greater in an open system compared to a closed one since agents are being called in one at a time as needed.

Table 7.1: Urban Firefighting learned policies comparison

Average of 10000 timesteps					
Max No. Agents	Method	Num Steps Active Per Eps			Episode Rew
		Agent 0	Agent 1	Agent 4	
2	**oDec-AIRL**	$\mathbf{18.06 \pm 1.13}$	$\mathbf{13.06 \pm 0.61}$	-	$\mathbf{32.22 \pm 0.59}$
	Dec-AIRL	16.87 ± 0.09	16.87 ± 0.09	-	30.37 ± 0.07
3	**oDec-AIRL**	$\mathbf{12.36 \pm 0.57}$	$\mathbf{9.27 \pm 0.12}$	$\mathbf{8.27 \pm 0.02}$	$\mathbf{37.86 \pm 0.13}$
	Dec-AIRL	10.33 ± 1.29	10.33 ± 1.29	10.33 ± 1.29	35.73 ± 0.70

Results: To provide expert data for training, we generated simulated trajectories based on human preferences, of 10^6 total timesteps for both oDec-AIRL and Dec-AIRL, trained both methods for 10^7 timesteps, and compared their best-learned behaviors using average episode reward and the average number of timesteps each agent is active per episode. As can be observed from Table 7.1, in the behavior learned by Dec-AIRL, all agents are present throughout the task duration as expected, and in turn incur a higher

[2] We have developed an open and a closed Dec-MDP version of the two domains in Gym-Classics (Daley, 2021) and MA-Gym (Koul, 2019) respectively for easy dissemination. Our code is publicly available on GitHub.

Figure 7.5: A complete episode of the Urban Firefighting domain with learned oDec-AIRL policies. Colored bubbles green and blue represent CallAgent and Extinguish actions respectively. In Figure 7.5a, Agent 0 heads towards the medium fire, and Figure 7.5b, calls Agent 1 to join. Subsequently, Agent 1 moves to the large fire in Figure 7.5c, while Agent 0 calls Agent 2. In Figure 7.5d, all agents extinguish fires at their locations. After extinguishing the medium fire, Agent 0 moves to the small fire, and Agent 2 assists Agent 1 with the large fire in Figure 7.5e. Figure 7.5f shows agents at their final locations performing the Extinguish action. This visualization was built using PyGame (McGugan, 2007).

step-cost. In comparison, oDec-AIRL policies only engage the second and third agents for 9.27 and 8.27 steps respectively in the three-agent case and accrue a higher average episodic reward.

7.3.2 Collaborative Furniture Assembly

In this realistic dyadic human-robot collaborative furniture assembly task, the goal is to assemble a table consisting of multiple parts: base, leg-support1, leg-support2, leg1, leg2, two screws for each leg-support to connect them to the base, and two screws for each leg to screw them into their respective supports, as

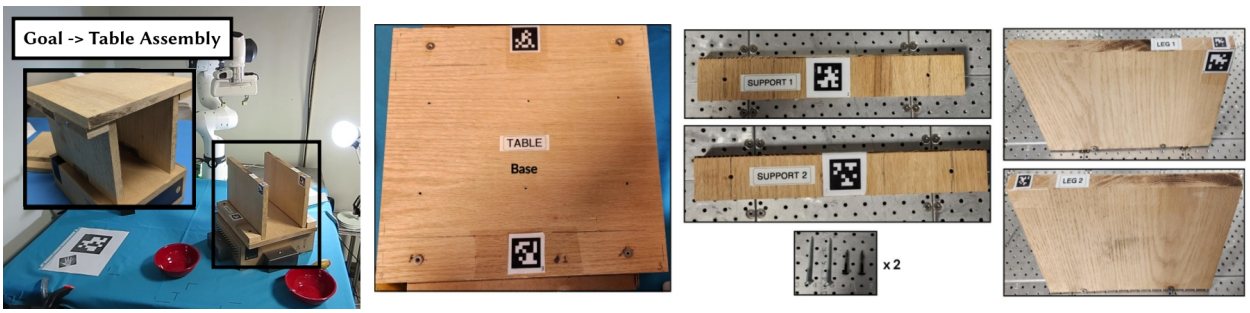


Figure 7.6: A collaborative table assembly task that involves placing and screwing together various wooden components. *Left:* The assembled table. *Right:* The individual components required for assembly, including the table base, two supports, two legs, and the necessary screws.

shown in Fig. 7.6. The task can be completed in multiple valid orders. For instance, one may position both leg-supports on the base and screw them in before positioning their corresponding legs and screwing the legs into their respective supports. Alternatively, one may position leg-support1, screw it into the base, place the leg1, screw it into the leg-support1, and analogously repeat the sequence for the other parts to complete the assembly. Notice that the simple positioning actions can be done independently by the robot, while the screwing action requires the assistance of a human. While the speed of assembly could be increased by having the human position parts in parallel from the beginning, the step-cost incurred due to the human's presence would be quite high. This means that the learned reward function must optimize both the reward obtained by completing the assembly sooner and the step-cost due to the human being present. In other words, the optimal behavior must only call the human into the task when imperative.

Each agent has 8 discrete actions: *ChooseTask* - This randomly assigns a valid next task to perform, *Pick* - Agent picks up the current part, *Place* - Agent places the current part at the goal location, *HoldInPlace* - Agent holds the current part steadily at its current location, *ScrewIn* - Agent screws the current part into place , *CallAgent* - This calls the human into the task, *ResetTask* - Agent places the current part back to its original location, *NoOp* - No action. The local state of each agent in the expert's oDec-MDP consists of three discrete variables: *TaskName* - which takes a valid task name from eleven discrete values when a ChooseTask action is performed; *TaskStatus* - which describes the current status of the task through one of seven discrete values; *Collab* - which provides the current collaboration level between unavailable, partial and full collaboration.

If the human is called in for assistance with a screwing subtask, upon completion of that subtask, the human may choose to stay idle by doing *NoOp* until the robot needs help again or may decide to participate by positioning other parts in parallel. The upside to the latter is that the task is completed sooner and the team receives a better reward; the downside is that if the human is engaged in a different task while the robot requires help, the human must perform a *ResetTask* action to place the current part back before helping the robot, which would extend the task duration.

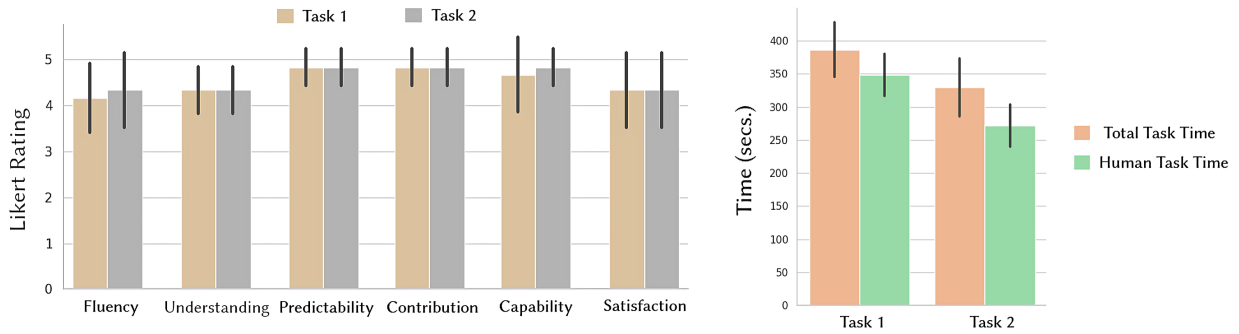


Figure 7.7: *Left:* Findings for subjective measures on a 5-point scale. *Right:* The average total duration of tasks and the average time allocated to human agents starting from the Call Agent action.

Simulation Experiments: We generated simulated expert trajectories of 10^6 total timesteps using human preferences and trained both oDec-AIRL and Dec-AIRL for 10^7 timesteps. These simulated trajectories were generated using a Python script based on human preferences and domain knowledge. This script was designed to generate team behavior that optimally completes the task while minimizing human time and effort. In these trajectories, the human agent on average intervenes 4 times per episode to perform the screwing action. When the human is called in depends on the task order, which is randomly selected in each episode to maximize policy exploration. We compared their best-learned behaviors using average episode reward and average number of timesteps the human and robot are present per episode. Notice that by following the Dec-AIRL policies, the human is active for the entire task duration, while with oDec-AIRL policies, the human is only active for 13.2 timesteps per episode as shown in Table 7.2. Naturally, oDec-AIRL policies accrue a better episode reward than the baseline policies.

Table 7.2: HRC Furniture Assembly learned policies comparison

Average of 10000 timesteps				
Max Num Agents	Method	Num Steps Active Per Eps		Episode Reward
		Robot	Human	
2	**oDec-AIRL**	$\mathbf{19.61 \pm 0.94}$	$\mathbf{13.20 \pm 0.98}$	$\mathbf{4.06 \pm 0.58}$
	Dec-AIRL	16.34 ± 0.97	16.34 ± 0.97	1.74 ± 0.60

Real-world HRC experiment: In our pilot study, we conducted experiments with human subjects collaboratively assembling a table with a Franka Emika Research-3 7-DoF robotic arm. We enlisted 6 participants following a within-subjects design. Participants were briefed on the assembly task, the robot's action space, and its intended goal. They were instructed to engage in the assembly task and cooperate with the robot when it initiated a 'CallAgent' action to request assistance. This request was presented via an on-screen graphical interface pop-up window. To aid in marker-based state estimation, we deployed an externally mounted Intel Realsense D435 camera with April Tags on the assembly components.

Additionally, we integrated human hand tracking using a deep learning model (**zhang2020**), which was meticulously calibrated within the experimental setup. An AprilTag positioned just outside the workspace was used to signal the completion of actions, which participants activated after each task step.

Table 7.3: Subjective Measures

Fluency	**Understanding**	**Predictability**	**Contribution**	**Capability**	**Satisfaction**
The robot and I collaborated fluently to accomplish the task.	I feel the robot had a good understanding of the task.	I was never surprised by the robot's actions.	The robot called for my assistance at the appropriate time.	I feel that my time and effort were valued by the robot.	I feel satisfied with the performance of the system.

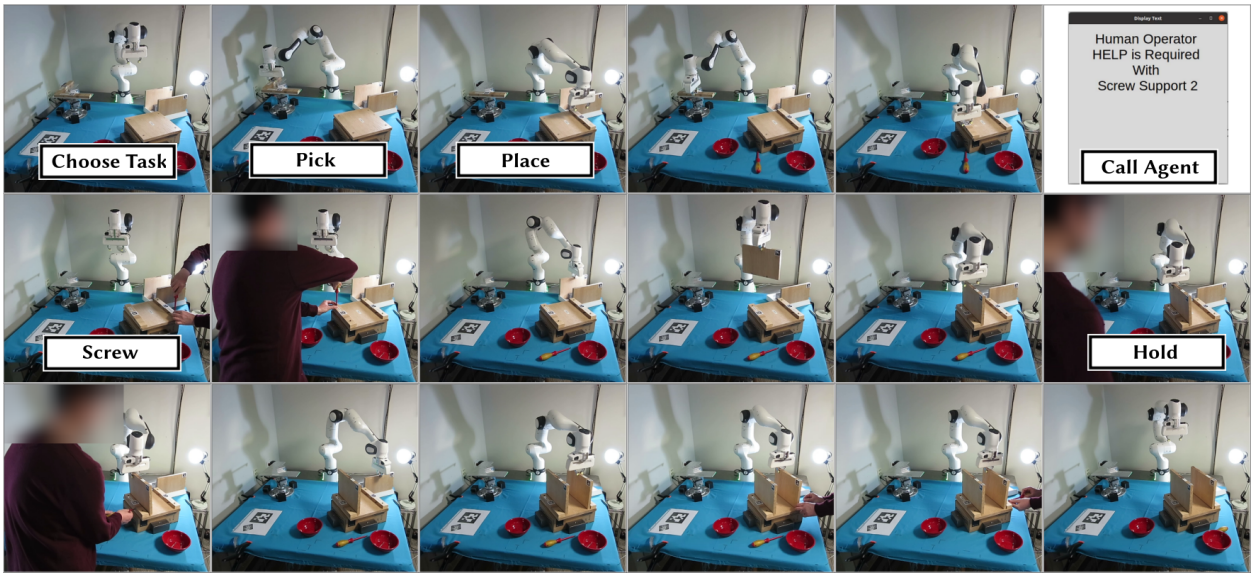


Figure 7.8: Snapshots capturing key moments from a typical trial in the human-robot pilot study. At each step, the robotic agent executes actions based on its learned policy oDec-AIRL, while the human participant is encouraged to emulate the learned human policy. Examples of each action are illustrated upon their initial occurrence. Following the completion of the 'Place' action for Support1, the robot prompts for human assistance through a pop-up notification for the 'Call Agent' action.

For manipulated variables, we shuffled task sequences and selected two variations of the collaborative table assembly task. Consequently, each participant performed the assembly twice. Task 1 entailed positioning leg-support2 and screwing it into the base, then positioning leg2 and securing it into its support, and analogously for leg-support1 and leg1 to complete the task. Conversely, task 2 involved positioning leg-supports 2 and 1, screwing them into the base, positioning leg1, securing it into its support, and repeating the process for leg2 to complete the assembly. Our focus was on examining the framework for real-world open human-robot collaboration and determining the optimal timing of the 'CallAgent'

action based on these task variations. For other measures, we measured task completion times and the duration spent by the human across both tasks (Fig. 7.7, right). We created six statements for subjective evaluation (Table 7.3), and asked participants to rate their level of agreement with these statements on a 5-point Likert scale (inspired by the Godspeed Series Questionnaire (Bartneck et al., 2008)).

All trials of the study were successful. Fig. 7.8 shows the Franka Research-3 robot assembling the table as per task order 2, calling the human for assistance at the appropriate time using the on-screen popup, after which the human and the robot collaboratively complete the rest of the assembly. Fig. 7.7 (left), shows the subjective ratings of the real-world experiments. Fig. 7.7 (right), shows the quantitative measures of the real-world experiments. Task 1 takes an average of 386.76 ± 41.19 secs for completion, while Task 2 takes 348.42 ± 32.28 secs. Through the 'Call Agent' action, on average, human agents only spend 329.49 ± 43.98 secs on Task 1 and 271.82 ± 31.55 secs on Task 2 demonstrating successful OHRC through an average time saving of approximately 18.39% for the human across both tasks.

Therefore, through these experimental domains we validate our claim that using our oDec-MDP model and oDec-AIRL method, we can successfully realize AO in HRC and improve upon the quality of HRC as compared to closed systems.

Chapter 8

Human-Robot Collaboration With Changing Human Types

With the advent of AI and robotics, robots that can make decisions under uncertainty are becoming increasingly common. Such intelligent robots can safely work alongside humans to effectively contribute to several aspects of human endeavors. This framework is commonly referred to as human-robot collaboration (HRC). HRC offers a paradigm shift in several key domains such as healthcare, manufacturing, and industrial automation by combining diverse yet complementary strengths of human and robotic agents to optimize task efficiency and throughput.

To realize seamless HRC, robots must not only optimize tasks but also recognize and adapt to the latent factors that influence human decision-making. HRC involves complex interactions and behavioral attributes that contribute to successful task completion. To capture these nuances in learned behavior, expert agents (usually humans) can demonstrate optimal behaviors as a team. These team demonstrations can then be used to learn the underlying expert preferences for tasks and collaboration. Cardinal to this framework is inverse reinforcement learning (IRL), which captures expert preferences from demonstrations using an underlying reward function. While IRL has been successfully applied to several single-agent domains (Adams et al., 2022; Arora & Doshi, 2021b) it remains underexplored in multiagent scenarios. Some prior works in IRL applied to HRC (S. Z. Nikolaidis, 2014; Sengadu Suresh et al., 2023) often assume that the human teammate follows a fixed behavior throughout, and doesn't account for realistic latent factors influencing human behavior such as prior biases or energy levels.

To demonstrate, envision a factory floor where humans stand across each other and sort produce on a conveyor. The optimal sorting behavior is to immediately discard visibly blemished produce and closely inspect seemingly unblemished items before deciding to discard or return them to the conveyor. However, prolonged periods of sorting can lead to fatigue, resulting in a decline in sorting quality. Fatigued sorters may become idle until they regain their energy. To maintain overall throughput, the robot must recognize and adapt to its fatigued teammate by accelerating its sorting speed. This 'super-collaborative' mode should only be maintained while the human is fatigued, as the increased speed poses a risk to active

humans and consumes more energy. This human-centric solution allows the robot to actively adapt to human energy levels throughout the task and maximize team efficiency.
We make three key contributions in this chapter:

- We present a novel multi-agent decision-making framework, TB-Dec-MDP, which captures evolving agent types, enabling agents in pragmatic HRC settings to detect their teammates' current type and adjust their behavior to maximize team reward.
- We formulate a novel decentralized IRL method, TB-Dec-AIRL, which learns a type-based reward function and a corresponding policy vector (one for each agent). This approach accurately captures the expert's underlying preferences, allowing adaptation to the human agent's evolving types and promoting seamless human-robot collaboration in shared workspaces.
- Finally, we demonstrate the efficacy of our learned human-centric solution on a realistic human-robot collaborative line sorting domain with a UR3e cobot.

By demonstrating the enhancement in collaborative efficiency via better average rewards per episode (on the MAGym (Brockman et al., 2016b; Koul, 2019) simulated environment), and increased throughput compared to the baseline (on the real-world robotic task execution), we establish the efficacy of TB-Dec-AIRL over Dec-AIRL and conclude by providing some avenues for future research.

8.1 Type-Based Decentralized AIRL for Human-Robot teaming

In this section, we formalize our novel multiagent model - Type-Based Dec-MDP (TB-Dec-MDP) and our novel IRL framework - Type-Based Dec-AIRL (TB-Dec-AIRL) to learn a type-contingent reward function and a corresponding vector of policies (one for each agent) that account for the other agents' behavioral changes during the task and adapts accordingly.

We model realistic human-robot collaborative settings where agents adjust their behavior during tasks due to latent factors such as trust-level or fatigue. In these situations, since all agents are working toward a shared goal, each must anticipate and adapt to the behavioral changes of others to complete the task effectively. While we present this approach in the context of human-robot collaboration, our model and IRL method apply to various similar domains such as automated healthcare systems or smart homes where the autonomous agent must recognize and adapt to evolving human types.

For instance, in healthcare settings, autonomous agents (e.g., robots, AI systems) assist patients by recognizing and adapting to key human types to provide personalized care. These types could include patients with medical conditions who may get sicker during the course of the day and recover once medicated. This requires consistent monitoring and tailored interventions, to provide effective medical support. Similarly, in smart home applications, the autonomous agent may need to detect if the human feels hot or cold and update the thermostat accordingly. This requires the agent to monitor the human's actions to adapt to their requirements constantly.

8.1.1 Type-Based Collaboration model

In addition to shared task attributes, each agent i has its own specific attributes and maintains a model of all other agents, including a belief about their joint types. Agent i's initial belief is based on prior knowledge of these types. The combination of i's private attributes and the common task attributes constitutes i's local state (s_i). Agent i makes decisions based on s_i and its belief about the other agents' joint type. At each timestep, agent i observes other agents' actions noisily and updates its belief accordingly (see Fig. 8.1). All agents collaborate in a decentralized manner to optimize a shared task reward. Since each agent has perfect access to its own local state and only maintains a belief about the other agents' types, our TB-Dec-MDP can be considered a generalized form of a mixed-observability MDP.

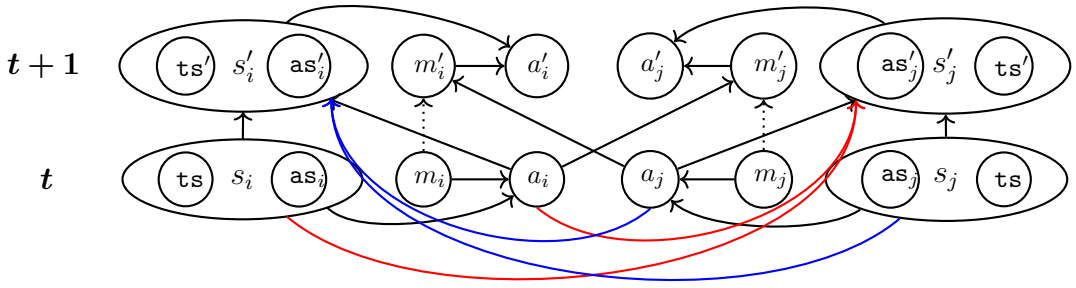


Figure 8.1: TB-Dec-MDP graphical model illustrates a dyadic team with agents i and j, for two timesteps $\boldsymbol{t}$ and $\boldsymbol{t+1}$. Local state of agent i (s_i) is a combination of i's private attributes $\texttt{as}_i$ and common task attributes $\texttt{ts}$. Model m_i holds i's current belief over the others' type θ_j. The dotted link updates model m_i using the other agent's action at the t. These apply analogously for agent j. All agents transition jointly to the next state.

A TB-Dec-MDP is formally defined as:

$$\mathcal{TB}-\mathcal{DM} \triangleq \Big\langle Ag, S, A, T, R, M \Big\rangle$$

- **Agent Identifiers (Ag) and the Number of Agents (N):**
  - Ag represents the set of all agent identifiers within the system. Each agent is uniquely identified, allowing for individual tracking and interaction within the environment.
  - The total number of agents in the system is denoted by N, where $N = |Ag|$. This cardinality is crucial for determining the complexity of interactions and the scalability of the model.
- **Joint State Space (S):**
  - The joint state space S is defined as the Cartesian product of the individual local states of all agents, i.e., $S = S_1 \times S_2 \times \ldots \times S_N$. This represents the complete configuration of the system at any given time.

- Each agent i has a local state S_i, which is further decomposed into:
  - $\texttt{tS}$: The *task-state*, which is common across all agents. It encapsulates the shared aspects of the environment or task that all agents are aware of and interact with.
  - $\texttt{aS}_i$: The *agent-specific state*, containing private attributes unique to agent i. This includes:
    - Θ_i: The *type* of agent i, representing its inherent characteristics, capabilities, or roles within the system.
- Therefore, $S_i = \texttt{tS} \times \texttt{aS}_i$, allowing each agent to maintain both shared and private information.

- **Joint Action Space (A):**
  - The joint action space A is the Cartesian product of the individual action sets of all agents, expressed as $A = A_1 \times A_2 \times \ldots \times A_N$. This encompasses all possible combinations of actions that agents can take simultaneously.
  - Each agent i has a local action set A_i, which includes all actions available to that agent. These actions represent the possible decisions or moves an agent can make at each time step.

- **State Transition Function (T):**
  - The state transition function $T : S \times A \times S \to [0, 1]$ defines the probability of transitioning from one joint state to another, given a particular joint action.
  - It is important to note that the types of agents, Θ_i, transition *exogenously*, meaning their transitions are governed by external factors and are not directly influenced by the state transition function T.
  - This separation ensures that while agents can act and influence the environment, their inherent types remain consistent unless altered by external dynamics.

- **Common Reward Function (R):**
  - The reward function $R : S \times A \to \mathbb{R}$ assigns a real-valued reward to each state-action pair, representing the immediate benefit or cost to the agents for taking action a in state s.
  - This reward is *common* to all agents, implying that it reflects a shared objective or goal that all agents are collectively trying to optimize. This can facilitate cooperative behavior or alignment of incentives among agents.

- **Agent Models (M):**
  - The set of agent models $M = M_1 \times M_2 \times \ldots \times M_N$ encapsulates the internal representations and beliefs each agent holds about the other agents.
  - Each agent i's model M_i is defined as:

$$M_i = \Big(\Theta_{-i},\ b_i,\ F_i\Big)$$

* **Types of Other Agents (Θ_{-i}):**
  - $\Theta_{-i} = \prod_{j \neq i} \Theta_j$ represents the combined set of types for all agents except agent i. This aggregation allows agent i to consider the various types of other agents when making decisions.
* **Belief about Other Agents' Types (b_i):**
  - $b_i \in \Delta(\Theta_{-i})$ denotes agent i's current belief distribution over the types of other agents. Here, $\Delta(\Theta_{-i})$ represents the probability simplex over Θ_{-i}.
  - This belief allows agent i to reason about and predict the behaviors of other agents based on their types, facilitating strategic interactions.
* **Type Transition Function (F_i):**
  - $F_i : \Theta_{-i} \times A_{-i} \to \Delta(\Theta_{-i})$ defines how agent i updates its beliefs about other agents' types based on observed actions.
  - Specifically, given the current types of other agents and their actions, F_i provides a new belief distribution over Θ_{-i}. This function captures the dynamics of how agent i interprets and adapts to the behaviors of others over time.

This formulation can be seen as a broader case of the *Ex Interim* expected utility formalism in Bayesian games (Shoham & Leyton-Brown, 2008) where each agent has perfect knowledge of their own type and has a mixed strategy profile of the others' type. For a dyadic team with agents i and j, the joint policy is given as $\boldsymbol{\pi} = \langle \pi_i, \pi_j \rangle$ where $\pi_i : S_i \times b_i \to A_i$.

8.1.2 IRL from team demonstrations

The belief update equation for agent i is given as:

$$b_i'(\theta_j' \mid a_j, b_i) = \beta \sum_{\theta_j} \Pr(\theta_j' \mid a_j, \theta_j) \Pr(\theta_j) = \beta \sum_{\theta_j} F_i(\theta_j' \mid a_j, \theta_j)\, b_i(\theta_j). \tag{8.1}$$

The joint experts' demonstrations contain state-action pairs as defined in Eq. (2.2) and a single belief trajectory corresponding to the expert trajectory is given as:

$$\hat{b}^E_{\boldsymbol{\theta}} = (\langle \theta_i^0, \theta_j^0 \rangle, \langle \theta_i^1, \theta_j^1 \rangle, ..., \langle \theta_i^{\mathcal{T}}, \theta_j^{\mathcal{T}} \rangle\rangle). \tag{8.2}$$

The discriminator optimization objective is defined as:

$$\mathcal{D}_{KL}(\rho_{\boldsymbol{\pi}_\omega}(\boldsymbol{s}, b_{\boldsymbol{\theta}}, \boldsymbol{a}) \parallel \rho_{\mathcal{X}^E}(\boldsymbol{s}, b_{\boldsymbol{\theta}}, \boldsymbol{a})) \approx \max_{\boldsymbol{\alpha}} \mathbb{E}_{(\boldsymbol{s}, b_{\boldsymbol{\theta}}, \boldsymbol{a}) \sim \mathcal{X}^E, \hat{b}^E_{\boldsymbol{\theta}}} \Big[\log D_{\boldsymbol{\alpha}}(\boldsymbol{s}, b_{\boldsymbol{\theta}}, \boldsymbol{a})\Big] +$$
$$\mathbb{E}_{(\boldsymbol{s}, b_{\boldsymbol{\theta}}, \boldsymbol{a}) \sim \boldsymbol{\pi}_\omega} \Big[\log\Big(1 - D_{\boldsymbol{\alpha}}(\boldsymbol{s}, b_{\boldsymbol{\theta}}, \boldsymbol{a})\Big)\Big]. \tag{8.3}$$

Note that $\mathcal{D}_{KL}$ [1] is a function of the belief module parameters ϕ through its dependence on the belief states. Therefore the IRL objective for policy optimization is written as:

$$\min_{\boldsymbol{\omega}} \mathcal{D}_{KL} \approx \min_{\boldsymbol{\omega}} \max_{\boldsymbol{\alpha}} \mathbb{E}_{(\boldsymbol{s},b_{\boldsymbol{\theta}},\boldsymbol{a})\sim\mathcal{X}^E,\hat{b}^E_{\boldsymbol{\theta}}}\Big[\log D_{\boldsymbol{\alpha}}(\boldsymbol{s},b_{\boldsymbol{\theta}},\boldsymbol{a})\Big] + \mathbb{E}_{(\boldsymbol{s},b_{\boldsymbol{\theta}},\boldsymbol{a})\sim\boldsymbol{\pi}_\omega}\Big[\log\Big(1 - D_{\boldsymbol{\alpha}}(\boldsymbol{s},b_{\boldsymbol{\theta}},\boldsymbol{a})\Big)\Big].$$

During the forward rollout, the joint belief over other agents' type is obtained from the belief module at each timestep and factored into each agent's policy learning to learn a vector of policies that capture the expert's behavioral preferences. The gradient for policy optimization is given by:

$$\nabla_{\boldsymbol{\omega}}\mathcal{D}_{KL} \approx \nabla_{\boldsymbol{\omega}}\left[\mathop{\mathbb{E}}_{\boldsymbol{\pi}_\omega}\left(\log D_{\boldsymbol{\alpha}}(\boldsymbol{s},b_{\boldsymbol{\theta}},\boldsymbol{a}) - \log\left(1 - D_{\boldsymbol{\alpha}}(\boldsymbol{s},b_{\boldsymbol{\theta}},\boldsymbol{a})\right)\right)\right].$$

Given fixed belief parameters ($\boldsymbol{\phi}$), the required gradient for policy optimization for agent i is obtained as:

$$L_i^{CLIP}(\omega) = \mathop{\mathbb{E}}_{\pi_{\omega,i}}\Big[min\Big(\lambda_i\, A^{\boldsymbol{\pi}}, clip(\lambda_i, 1-\epsilon, 1+\epsilon)\, A^{\boldsymbol{\pi}}\Big)\Big] \text{where } \lambda_i^t = \frac{\pi_{\omega,i}(a_i^t|s_i^t,b_i^t)}{\pi_{\omega,i}^{old}(a_i^t|s_i^t,b_i^t)}.$$

where the advantage function is computed as:

$A^{\boldsymbol{\pi}_\omega} = Q^{\boldsymbol{\pi}_\omega} - V^{\boldsymbol{\pi}_\omega}$.

where the action value-function $Q^{\boldsymbol{\pi}_\omega}$ is given by

$$Q^{\boldsymbol{\pi}_\omega} = \mathbb{E}_{\boldsymbol{\pi}_\omega}\Big[\sum_{t'=t}^{\infty}\gamma^{t'-t}\Big(\log D_{\boldsymbol{\alpha}}(\boldsymbol{s}^{t'},b^{t'}_{\boldsymbol{\theta}},\boldsymbol{a}^{t'}) - \log\Big(1 - D_{\boldsymbol{\alpha}}(\boldsymbol{s}^{t'},b^{t'}_{\boldsymbol{\theta}},\boldsymbol{a}^{t'})\Big)\Big)\Big].$$

and the state-value function $V^{\boldsymbol{\pi}_\omega}$ is given as

$V^{\boldsymbol{\pi}_\omega}(b) = V(\boldsymbol{s},b_{\boldsymbol{\theta}}) = \max_{\alpha\in\Gamma_{\boldsymbol{\theta}}(s)}(\alpha \times b_{\boldsymbol{\theta}})$ where $\alpha = \langle V(\boldsymbol{s},\theta_1), V(\boldsymbol{s},\theta_2), V(\boldsymbol{s},\theta_3)\ldots V(\boldsymbol{s},\theta_n)\rangle$.

For implementation purposes, we use a GRU-cell (Cho et al., 2014) to represent our belief module (ref Fig. 8.2) as it has been empirically demonstrated to have an equivalent representation.

8.1.3 Theoretical Analysis

We assume that the type transition kernel (F_i in Eq. (8.1)) acts as a contraction mapping, i.e., $\exists\, 0 \leq \delta < 1$ such that $\mathcal{D}_{TV}(b^{t+1}, b^t) \leq \delta \cdot \mathcal{D}_{TV}(b^t, b^{t-1})$, where b refers to the *joint* belief and $\mathcal{D}_{TV}$ denotes the total variation distance. This is justified as Bayesian belief propagation is convergent for the type of networks shown in Fig. 8.1. Then the following holds (proof is in the Appendix in the supplement):

[1] We mention just $\mathcal{D}_{KL}$ from here on for succinctness. However, the input arguments are the same as the ones shown in Eq. (8.3).

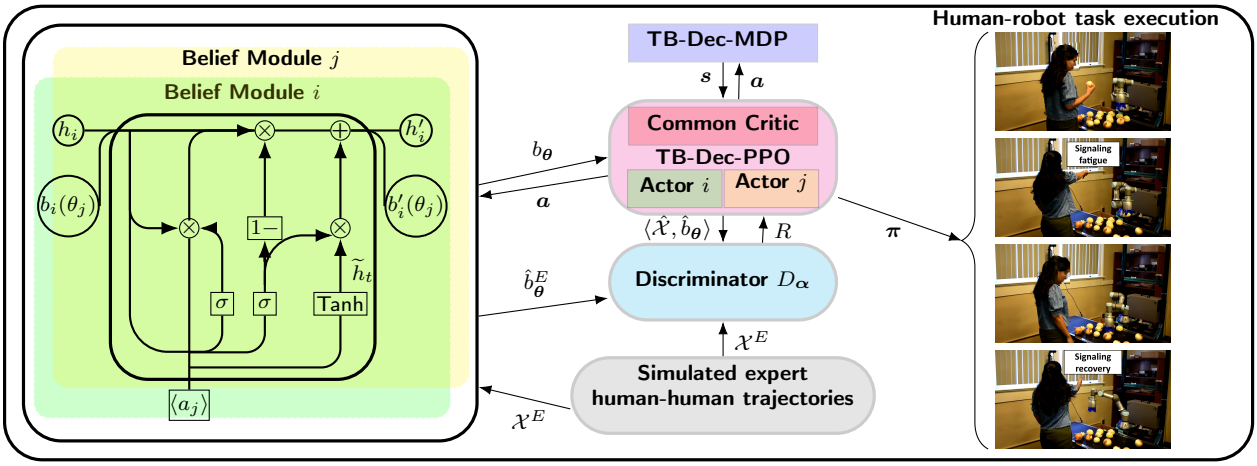


Figure 8.2: The TB-Dec-AIRL architecture for a dyadic team with agents i and j, and simulated human-human expert trajectories ($\mathcal{X}^E$). Agent i's belief module uses agent j's actions to generate belief states of j's type, and vice versa, creating joint $\hat{b}^E_{\boldsymbol{\theta}}$ trajectories. Similarly, TB-Dec-PPO interacts with TB-Dec-MDP, the type-based reward function R, and the belief module to generate $\langle \hat{\mathcal{X}}, \hat{b}_{\boldsymbol{\theta}} \rangle$. Both $\langle \mathcal{X}^E, \hat{b}^E_{\boldsymbol{\theta}} \rangle$ and $\langle \hat{\mathcal{X}}, \hat{b}_{\boldsymbol{\theta}} \rangle$ train the discriminator $D_{\boldsymbol{\alpha}}$ to update R until convergence. The learned policy $\boldsymbol{\pi}$ is then applied in real-world HRC, where the robot follows its learned policy and the human emulates the human policy behavior. Here h denotes the hidden state of the GRU, and b denotes the normalized belief.

Theorem 1 *If the i-th agent's discriminator error compared to the i-th expert is small, that is, $\|D_i^t - D_i^E\| \leq \epsilon \; \forall i = 1, \ldots, N$, then the conditional log-likelihood (LL) of data is bounded:*

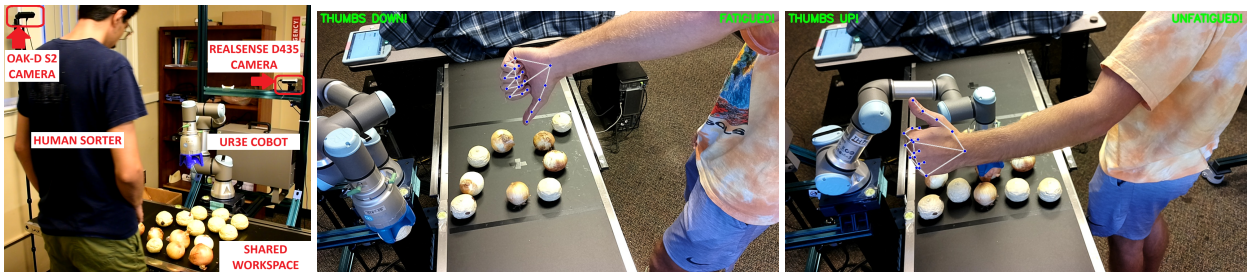
$$LL(\mathcal{X}|R^E) - LL(\mathcal{X}|R^t) \leq \frac{8|\mathcal{X}|N\epsilon}{1-\gamma(1-\delta/2)},$$

where R^E and R^t are the expert and learned common *reward functions at iteration t.*

As adversarial inverse learning algorithms have a convergence rate of $\mathcal{O}\left(\frac{1}{(1-\gamma)^3\sqrt{t}}\right)$ (Guan et al., 2021), it follows that as ϵ decreases at that rate, the average regret in log-likelihood error approaches 0.

8.2 Experiments

In this section, we consider a collaborative HRC domain to validate our model and our IRL method. We use a realistic use-inspired human-robot collaborative produce sorting domain where a human and a UR3e cobot stand across each other and sort onions on a line conveyor. In this collaborative produce sorting domain, the optimal sorting behavior involves briefly assessing each onion on the conveyor. If an onion is blemished, it should be immediately picked up and discarded in the bad bin. If it appears unblemished, it should be picked up for a closer inspection. If it remains unblemished, it is returned to



(a) (b) (c)

Figure 8.3: (a) shows the HRC sorting setup with a Realsense D435 behind the robot detecting objects on the conveyor and the OAK-D S2 camera to the left of the human monitoring their actions. (b) and (c) depict a human sorter signaling fatigue and recovery to the robot during sorting.

the conveyor; otherwise, it is discarded in the bad bin. Both the human and the cobot traverse the shared workspace and work in a decentralized manner, while the cobot accommodates changes in the human's type during the sort.

Simulation Experiments The simulated environment for the collaborative sorting domain was developed as a text-based discrete state-action domain on MA-Gym (Koul, 2019) based on domain knowledge from Sengadu Suresh et al., 2023. Each agent's belief update was represented using a pretrained GRU model, which was trained with ground truth labels from the Gym environment and hand-coded policy actions. As shown in Fig. 8.4, upon convergence, the GRU model is near-identical to a classical Bayesian belief update. In this environment, each agent's state contains 5 discrete variables: *Onion location* - this takes one of 4 values based on the current onion location; *End-effector location* - this also receives one of 4 values based on the current end-effector location; *Prediction* - this takes one of 3 values based on blemished, unblemished, or unknown prediction label of the onion in focus; *Self-type* - this holds the value of the current agent's type; *Indication* - variable indicating if the current agent's type change has been communicated to the other agent.

Table 8.1: **HRC Type-Based Produce Sorting Learned Policies Comparison on MA-Gym**

Average of 1000 episodes		
Method	Onions Sorted Per Eps	Eps Reward
Upper Bound (Expert)	64 ± 1	64 ± 1
TB-Dec-AIRL	$\mathbf{56 \pm 2}$	$\mathbf{55.7 \pm 0.78}$
Dec-AIRL	45 ± 2	43.2 ± 0.65

Similarly, each agent has 9 discrete actions: *No-op* - No operation, *Detect* - Choose any onion on the conveyor, *Pick* - Pick up the chosen onion from the conveyor, *Detect-pick* - A combined action to choose any onion on the conveyor and immediately pick it up, *Inspect* - Inspect the picked onion, *PlaceOnCon-*

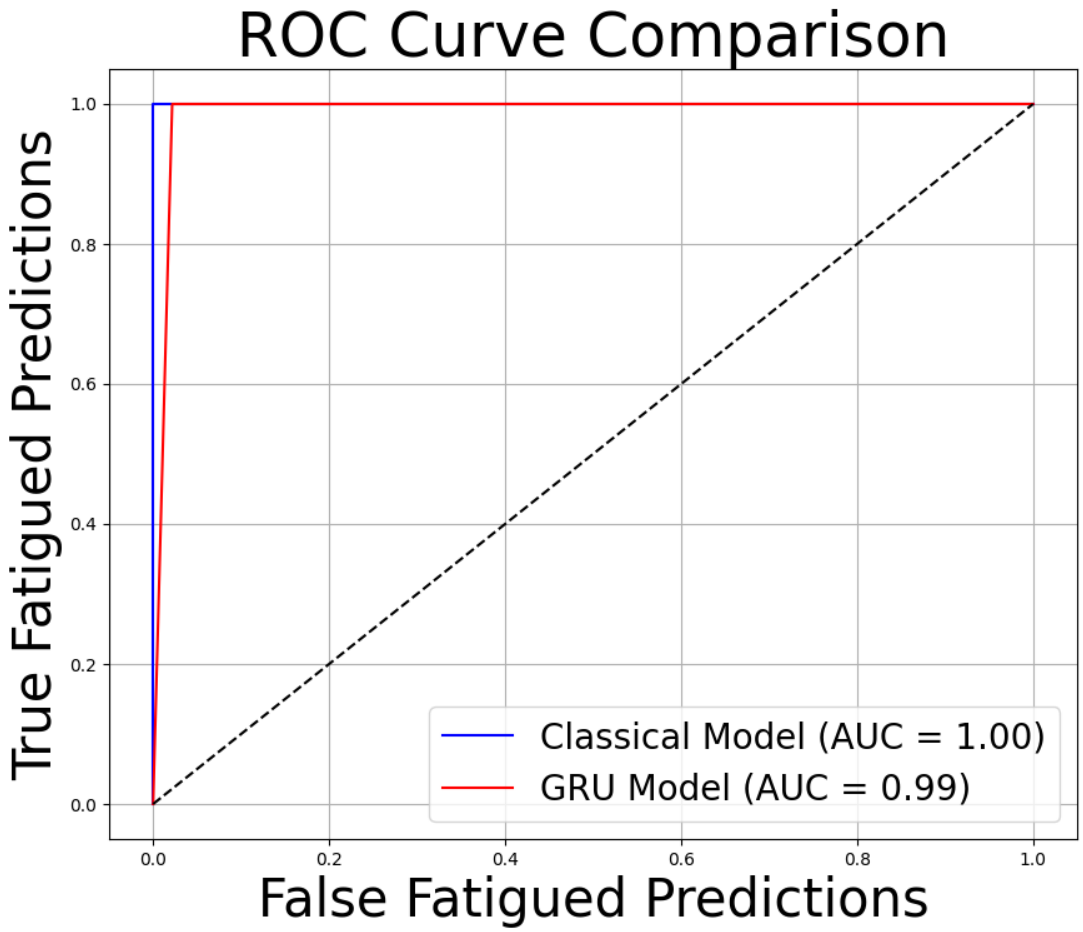


Figure 8.4: Figure shows the receiver operating characteristic (ROC) plot comparing the pretrained GRU module predictions, a classical Bayesian belief module predictions, and the ground truth, for 100 episodes. Notice that a well-trained GRU model performs near-identical to a classical belief update.

veyor - Place the held onion back on the conveyor, and *PlaceInBin* - Place the held onion in the discard bin. The last two actions for the human are *Thumbs-up* - gesture indicating unfatigued type to the robot, and *Thumbs-down* - gesture indicating fatigued type to the robot. Similarly, the robot's last two actions are *Speed-up* - Increase movement speed to maintain throughput and enter super-collaborative mode and *Slow-down* - Reduce movement speed to return to regular collaborative mode. The *evaluation reward function* (used to compare the expert and learned IRL trajectories) assigns a $+1$ reward for successfully sorting an onion and a -1 penalty for incorrect sorting (e.g., placing a bad onion back on the conveyor).

Results As depicted in Table 8.1, the policy learned by TB-Dec-AIRL accrues a higher average reward compared to the baseline Dec-AIRL policy and performs similarly to expert behavior. This is because

our method uses the belief over the human's type to adjust robot behavior accordingly. Notice that this adjusted behavior differs from the robot simply defaulting to a single sorter mode. The policy organically learns that when the human is fatigued, fewer onions are sorted, thereby reducing the team's overall reward. This understanding allows the robot to choose 'super-collaborative' actions, which in turn increase the team's throughput. This behavior is demonstrated in the expert trajectories where one of the humans in the human-human team rests every so often and the other human speeds up during this time to maintain throughput. Although both methods operate within the same HRC Dec-MDP framework, the baseline policy relies solely on each agent's local state attributes to decide actions. In contrast, TB-Dec-AIRL's policy takes into account both the agent's local state and their belief about the other agent(s)' type. This enables the robot to use 'super-collaborative' actions at the appropriate times, leading to higher rewards.

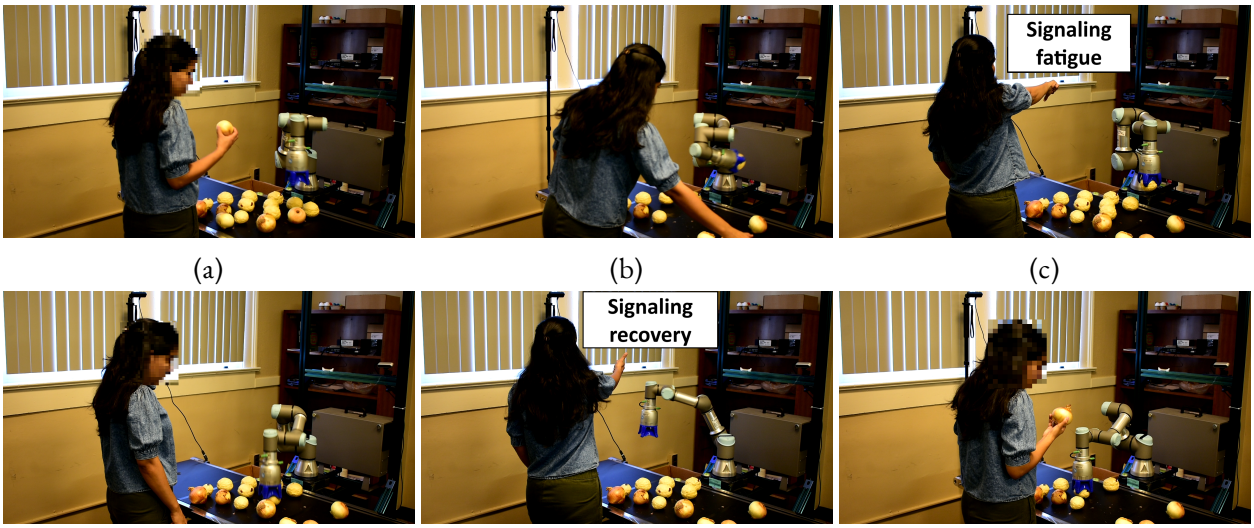


Figure 8.5: Key frames from a human-robot collaborative sort are shown. In Fig. 8.5a, the human and robot begin sorting, with the human inspecting an unblemished onion while the robot attempts to pick an onion. Fig. 8.5b captures the the human placing an onion back on the conveyor while the robot inspects the picked onion. In Fig. 8.5c, the human indicates fatigue with a thumbs-down gesture to the OAK-D camera. The robot responds by entering super-collaborative mode in Fig. 8.5d, moving faster while the human performs a NoOp. Finally, in Fig. 8.5e, the human signals recovery with a thumbs-up gesture, and both agents collaboratively complete the sort in Fig. 8.5f.

Physical robot experiments To sort alongside the human, we utilize a Universal Robots UR3e 6-DOF cobot equipped with a Realsense D435 RGB-D camera for onion detection and an OAK-D S2 RGB-D camera for human action estimation. The raw RGB frames from the Realsense camera are processed using the popular object detection network YOLOv7, which generates bounding boxes around the onions on the conveyor. By combining these bounding boxes with their corresponding depth information, we compute the real-world 3D locations of the onions using rigid transforms and a pinhole camera model. Concurrently, we employ a hand-tracking method (**zhang2020**), fine-tuned for our application, on the

RGB input from the OAK-D camera. This output is similarly processed to determine the 3D location of the human hand, which is then used to assess the human's actions at each step.

The learned policy, exported to a CSV file, is loaded into a finite-state machine that controls the robot according to the policy at each timestep. After the fatigued human signals their condition with a thumbs-down gesture, they remain idle. Upon recognizing this change, the cobot shifts to a 'super-collaborative' mode, increasing its speed to sort more onions. After a few timesteps, the human indicates their recovery with a thumbs-up gesture and resumes sorting. The robot then updates its belief of the human's type and returns to its regular 'collaborative' mode.

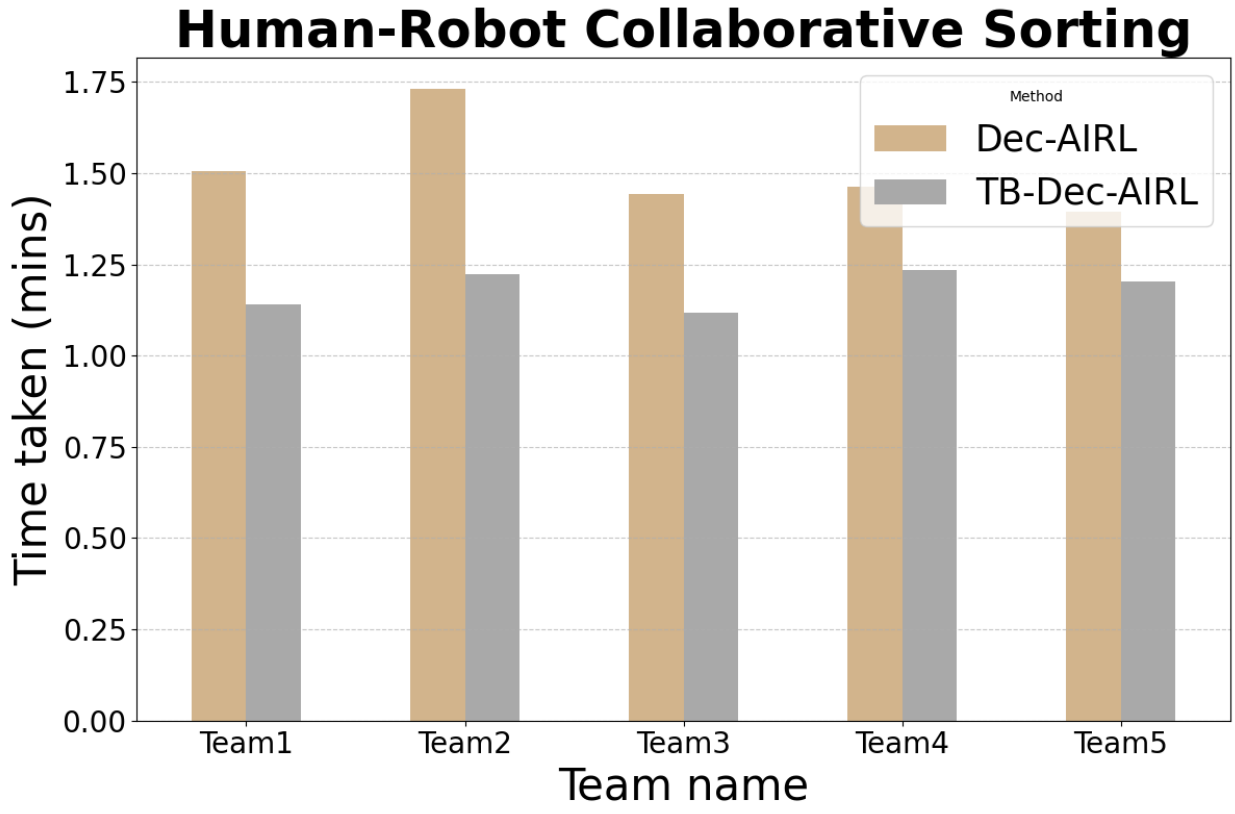


Figure 8.6: Figure shows the average collaborative sort time for 15 onions with a UR3e cobot on a line conveyor. Each human-robot team performed 3 runs with TB-Dec-AIRL policy and 3 runs with Dec-AIRL policy.

Results: Five human experimenters (not involved in this paper) conducted six rounds of onion sorting alongside the cobot. During the first three rounds, the cobot followed our TB-Dec-AIRL policy, and for the remaining three rounds, it adhered to the Dec-AIRL policy behavior. All trials were successful. We recorded the number of onions sorted and the time taken for each round, comparing these results with those from the baseline policy (the ablation study), as shown in Fig. 8.6. Our findings indicate that the TB-Dec-AIRL policy significantly improves upon the baseline.

Chapter 9

Future work

So far in this thesis, we addressed numerous challenges related to robotic learning in real-world scenarios. In this section, we discuss potential future extensions of our work and their relevance to robotics and HRC scenarios.

9.1 MMAP-BIRL for IRL under Occlusion and Sensor Noise

One of the first challenges we addressed was learning meaningful representations from noisy and incomplete data from realistic robotic sensors. Using marginalization combined with a sensor model we learned a reward distribution that sufficiently solved the problem. However, there are some aspects of this method that can be extended to further generalize it to HRC scenarios and more complex robotic domains.

- **Multiagent MMAP-BIRL for HRC scenarios:** In HRC scenarios, in order to learn how a robot can collaborate with a human in a shared workspace, currently, we use data from a human-human team to model and learn collaborative dynamics. This data from the human-human expert team demonstrating the task is recorded through RGB-D cameras placed behind each human. However, occlusions naturally manifest when two or more agents share a workspace. The problem of IRL under occlusion and sensor noise becomes much more profound when interaction-related features become noisy and occluded along with task-related features. Therefore, by extending the MMAP-BIRL method to multiagent (HRC) domains, we may learn a more robust representation from such imperfect expert data.

- **Generalizing MMAP-BIRL to continuous domains:** In our gradient computation, although the reward function and its weights θ are continuous variables since our state-action features are discrete, the gradient computation is somewhat simplified. By extending this gradient computation to work with continuous state-action domains, MMAP-BIRL can be directly applicable to real-world scenarios without any need for discretization.

- **Considering the bias introduced by an informative prior:** Many times in real-world scenarios, it is hard to estimate a good and informative prior to begin with; Bayesian IRL techniques suffer

when a good prior is not available. A future extension of the method could find a way to remove or at least ameliorate the dependency level of Bayesian IRL on its prior.

- **Learning the sensor model:** In our work, we assume that an accurate sensor model is available for the real sensor we use to record the expert data. However, our empirical estimate of the sensor model may not be perfect and a future extension of the method may attempt to learn the sensor model along with the task itself. There has been prior work in learning the observation model (Arora et al., 2019) in an online fashion. However, the effectiveness and scalability of this technique in real-world domains is not well established.

- **Model-free generalization:** Currently MMAP-BIRL is a model-based technique that assumes access to an accurate model of the world. It uses this model of the world in multiple key points of learning such as the forward-backward search (used to narrow the candidates for marginalization), and its transition dependency during likelihood computation. A future work may generalize MMAP-BIRL formulation such that it learns in a model-free fashion. A place to start such a model-free generalization would be using this previous work of Model-Free MLE using IRL (Jain et al., 19).

9.2 Dec-AIRL for Human-Robot Teaming using IRL

The next challenge we handled was modeling and learning behavioral policies for human-robot collaboration in shared workspaces, using human preferences. To achieve this ambitious goal, we ascribed a Dec-MDP to a two-agent team working collaboratively on the same task, and modeled necessary and adverse interactions through interactive and non-interactive state representations. We generalized a previous deep IRL technique - adversarial IRL to learn in a decentralized manner, and used human-human team demonstrations to learn a common reward function and a vector of policies (one for each agent).

However, this work can be further extended in multiple directions to be more robust, and address many other HRC-related problems:

- **Modify Dec-AIRL's discriminator to recognize key state-action pairs:** Currently AIRL follows a GAN architecture that weights all input data points on the same level. But if certain state-action pairs correspond to goals, safety-features or HRC interactions, they must be given higher importance in reward learning than the steps leading to them. Similar to *eligibility traces used in forward RL* techniques, Dec-AIRL's discriminator must be modified to recognize the underlying features of such key steps in the trajectory and place a greater weight (whether positive or negative) on them.

- **Learning from failure:** Expert trajectories in IRL typically only contain the optimal way to solve the problem and never showcases the ways in which the task can fail. Especially in cases like HRC, adverse interactions within shared workspaces must be avoided at all costs and just learning the optimal way to avoid them is sometimes not sufficient. By using such failure trajectories (Shiarlis et

al., 2016), the reward function can learn to assign a high negative penalty to such adverse interactions, thus preventing the policy distribution from ever reaching such actions.

- **Dealing with noisy expert trajectories:** Although there has been some work in this direction (Arora and Doshi, 2021a; L. Chen et al., 2021; P. S. Suresh and Doshi, 2022), it still hasn't been sufficiently researched and solved. Since Dec-AIRL works on a delicate balance between the discriminator and the generator, noisy expert input would throw off the entire learning paradigm. If such noisy behavior is allowed by the environment dynamics, Dec-AIRL would learn a reward function that is optimal for such behavior demonstrated by the expert. Instead, either by allowing some domain knowledge (using a prior), by weighting expert demonstrations based on their level of optimality (Brown et al., 2020), or by modifying the discriminator to just use the expert demonstrations as a starting point to guide the forward RL policy towards the right direction (instead of trying to learn the underlying preferences exactly), Dec-AIRL could be modified to work with noisy trajectories.

- **Dealing with inaccurate world model:** The paradigm of RL and IRL heavily relies on having a simulated model of the world that accurately reflects the real-world environment. This assumption can be relaxed to use, say, multiple simulated environments that, in expectation represent the world near-optimally and by sampling from all such environments and learning a generalized representation of the reward function.

- **Dealing with agent asynchronicity:** Humans and robots naturally work at different paces and the Markovian assumption used in modeling MDPs limits the action duration to last for one timestep and forces all agents to transition to the next state simultaneously. However, if certain agents use temporally extended actions, how can Dec-AIRL and its underlying model be adjusted to accommodate asynchronous transitions and temporally extended actions? How does this affect the collaboration between humans and robots? These are some interesting directions that can be pursued in the future. Although a previous work (Amato et al., 2019) uses macro-actions combined with stochastic stopping criteria to address this, it only ameliorates the problem by allowing temporally extended actions. Still, it doesn't discuss how this affects the collaboration between agents.

- **Avoiding local optimas:** This happens mostly because forward RL techniques either fail to explore enough during learning or take large steps that drag the policy distribution far from the optimal direction. Finding this delicate balance during exploration is still an unsolved problem in RL. But can Dec-AIRL's discriminator be modified to learn a reward function that provides extra incentive for effective exploration? How does this impact the Maximum Causal Entropy formulation and consequently, the entropy regularization term used in Dec-AIRL's reward learning?

9.3 Agent Openness using Decentralized AIRL

Further into our endeavor in HRC, we developed a novel multiagent model and a corresponding IRL-based solution to allow Agent Openness during task execution. This permits the robot's learned policy to request human assistance when it requires it. By allowing agents, both human and robotic, to enter and exit the system as needed, this system provides the flexibility and efficiency of only handling the complexity of the currently active agents at any given step. Additionally, this allows the human to concurrently work on multiple tasks and save their time and energy that would otherwise be spent either staying idle until their help is needed or wasted on trivial tasks. This is a pioneering effort to introduce openness into a domain hitherto modeled as a closed system. Nonetheless, the following future extensions to this work could make it more robust and easily applicable to real-world HRC domains:

- **Introducing other types of openness:** So far in this work we only considered Agent Openness (AO), which allows agents to freely join or leave a system by invitation or on their own accord. However, there are other modalities of openness that commonly manifest in real-world systems (Eck et al., 2023). Task Openness (TaO) is when the goals of one or more agents shift from their initial assessment. This may organically manifest when a human joins the task and decides that a different subtask is of higher priority than the one the robot invited them for. The model must be able to capture such changes in goals and the learning paradigm must dynamically adjust its policy to complete these updated goals. Alternatively, another type of openness termed Type Openness (TO) may allow agents to change their collaboration type in the middle of the task. For instance, a human agent may decide mid-task to focus on the goal that maximizes their own reward instead of the team reward as a whole. This may sometimes lead to suboptimal or even negative rewards for the robot or for the entire team. Therefore, the underlying model has to be modified to properly represent the reward function structure for all possible types of agents that may present themselves and the learned policy must dynamically adjust its actions to changing agent types.

- **Modelling additional levels of uncertainty in the task:** Other factors such as mental states that influence human decision-making; environment, and agent stochasticity that prevents deterministic transitions, and partial observability that obfuscates the true state and action, further complicate modeling and learning policies that allow for different forms of openness. For instance, an example of agent stochasticity may be that the human is far away from the robot at certain times and standing closer to the robot at others. This may stochastically alter the time of entry of the human into the environment. Environment stochasticity may manifest through environmental factors such as dynamic obstacles blocking the robot's view of the human, thus preventing it from knowing if the human is present or absent. Partial observability may be caused due to noisy sensors used for feedback, limited field-of-view of sensors, or malfunctioning equipment. All these realistic factors have to be considered when modeling openness in HRC context.

- **Modelling the human to better understand openness:** By modeling the human and observing if their energy level is depleted, the robot can request a replacement for the human (through AO)

since a fatigued human operating machinery could lead to undesirable consequences. Also, by modeling the human, the robot can sense the change in agent type (Tao) and adjust its behavior accordingly to maximize team reward. An accurate human model may also inform the robot of the shift in goals of the human (TO) so that the robot may reorient itself toward the new targets.

9.4 Type-Based Dec-AIRL for Changing Human Types in HRC

In this work, we focus on closed multiagent systems where a fixed number of agents are present throughout the task. We develop a model that includes an agent model and a task model for each agent. The agent model allows each agent to maintain a belief over the other agents' types and relevant attributes, while the task model captures attributes pertinent to the task itself. Our IRL-based solution leverages expert demonstrations and a belief module to learn a type-based reward function and corresponding policies for each agent. These policies adjust actions based on current beliefs about the other agents. This approach enables the robot to model and detect human fatigue, modifying its behavior to maximize the team's reward. It also captures hidden mental states of the other agents, allowing better reasoning before action selection.

Discussion: In this work, we model the task as a mixed-observability MDP (MOMDP). Agents are fully aware of their own local state attributes and maintain a belief over the other agents' joint type, similar to the Ex Interim expected utility formalism in Bayesian games. In partially observable scenarios, agents may use probing actions to trigger the environment or other agents to reveal information for updating their beliefs (Gopalan & Tellex, 2015; Sadigh et al., 2016; Whitney et al., 2017). Although our models, TB-Dec-MDP and TB-Dec-AIRL, do not explicitly prohibit this information gathering, our current HRC setup only considers cases where the human voluntarily performs an action that reveals their type to the robot. This assumption may not apply to all HRC scenarios, as there may be situations where the robot needs to request information or probe the environment. For example, Williams et al. (2019) use clarification requests to obtain relevant contextual information, employing the Dempster-Shafer theory to clarify topics the robot is uncertain about. While gestures and physical interfaces like buttons are effective, speech remains a crucial modality for effective HRC.

Limitations: TB-Dec-AIRL and its underlying model, TB-Dec-MDP, offer enhanced flexibility for modeling realistic HRC scenarios. However, we operate under the assumption that expert trajectories are available as state-action pairs (a common assumption in IRL). While the task attributes of the state are observable using sensors like RGB-D cameras, certain hidden state attributes such as the agent's own type, is a mental state. In our scenario, we leveraged simulated human-human trajectories to seamlessly generate expert trajectories. However, future work may reframe the state definition to attributes that can be fully captured through observable traits.

Future work: Future research could explore estimating the policies of other agents within each agent's model (similar to I-POMDPs Gmytrasiewicz and Doshi, 2005) to assess their level of rationality and adapt accordingly. Given that humans are boundedly rational Simon, 1990, the robot could enhance its adaptability to different human teammates by estimating their rationality levels. Furthermore, while TB-Dec-AIRL is designed for simultaneous action scenarios, the human and cobot may complete actions at different rates. Therefore, TB-Dec-AIRL may need to be extended to handle time-varying actions, possibly by incorporating a variant of the Dec-Semi-MDP Goldman and Zilberstein, 2008; Sutton et al., 1999. Future work may also utilize simulated studies to estimate scalability of TB-Dec-AIRL with increasing number of agents.

9.5 General comments on our methods

In our recent works, oDec-AIRL and TB-Dec-AIRL, we introduced two forms of non-stationarity in human behavior: agent-openness and dynamic types. We proposed IRL techniques to address these forms using expert human-human team demonstrations. While IRL sidesteps the complex reward design problem, it has certain key issues that need to be acknowledged and handled.

Firstly, although many platforms simulate realistic robot dynamics (Coumans & Bai, 2021; Mittal et al., 2023; Tao et al., 2024; Todorov et al., 2012), there is a notable lack of platforms that simulate realistic human motion and dynamics. Consequently, we rely solely on real human-human demonstrations, which limits the quantity and variety of data since real humans cannot provide extensive hours of training data.

Secondly, the well-known problem of covariate shift (Sugiyama & Kawanabe, 2012) constrains the robot's ability to generalize in non-stationary HRC environments, leading to a mismatch between training and execution scenarios.

Additionally, human experts may perform actions beyond the robot's capabilities, such as dexterous maneuvers or reaching beyond the robot's workspace. Our work assumes enough human-human demonstrations to capture the necessary variance for the robot to generalize effectively and that these demonstrations remain within the robot's action-space. These assumptions may be challenging to scale across multiple task domains.

Finally, robots may encounter unforeseen non-stationarity during execution, as human behavior is inherently unpredictable (Cziko, 1989). To handle this, robots must dynamically switch between actions. Mac-DecPOMDPs (Liu et al., 2017) extend Semi-MDPs (Sutton et al., 1999) with a stochastic stopping criterion for actions, enabling asynchronous agent operation. Additionally, our work focuses on learning high-level discretized actions rather than low-level joint angles, which can be further achieved using Hierarchical MDPs (McGovern et al., 1998).

Future methods must address these practical limitations and advance the efficiency of HRC through improved inverse learning techniques. Exploring ways to simulate realistic human behavior, mitigating covariate shift, and ensuring demonstrations align with the robot's action-space will be crucial for the continued development of robust HRC systems.

Chapter 10

Conclusion

Envisioning seamless collaboration between humans and robots in everyday tasks, we tackled multiple challenging problems posed by HRC in shared workspaces. We systematically disentangled the modeling-, behavioral-, and implementation-level challenges and proposed novel learning-from-observation-based solution techniques using multiagent inverse reinforcement learning (IRL). By utilizing realistic HRC domains like collaborative line sorting and furniture assembly, we effectively demonstrated the effectiveness of our methods compared to existing state-of-the-art.

Our first contribution - MMAP-BIRL (Suresh et al. 2022) - addressed the limitations of realistic feedback sensors such as RGB-D cameras, often used in AI and Robotics. We demonstrated how a robot observing a human expert through such sensors may lose portions of the expert's trajectories due to people moving in front of it, and how sensor noise can make it hard to ascertain the exact state and action the demonstrating expert is in. Subsequently, we highlighted the importance of including the missing data, noting that it may hold information key to successful reward learning through IRL. Using a novel mathematical formalism, we incorporated the noisy and missing data into the learned reward function, making it informative enough to solve the task efficiently. We validated this by using the learned policy to sort onions on a line conveyor. We used precision and recall as metrics to evaluate our learned policy's effectiveness compared to a state-of-the-art baseline (HiddenDataEM) policy.

Consecutively, in our second contribution - Dec-AIRL (Sengadu Suresh et al., 2023), we posited that a Dec-MDP is better suited for pragmatic human-robot collaboration and generalized a popular deep-IRL technique - adversarial IRL to work in decentralized settings. Using expert human-human team demonstrations, we captured the *necessary* and *adverse* interactions encountered by human and robotic agents within the shared workspace. We learned a vector of behavioral policies (one for each agent) that successfully navigated such intricacies while efficiently completing the task. We validated our learned policies by deploying them in a real-world HRC task, where human teammates work collaboratively with a robot to sort onions on a conveyor. By comparing the precision and recall of our learned policy to a state-of-the-art baseline technique (Co-GAIL), we established that our method significantly improves upon existing art.

In our third contribution, we pioneered the idea of *agent-openness* in HRC (Suresh et al. n.d., 2024) where an agent (human or robotic) could join or leave the task as needed. We posited that in most real-world HRC domains, the human is only needed for a short duration to help with dextrous manipulation subtasks and can then leave to focus on other tasks. To realize this ambitious idea, we proposed a novel multiagent model — oDec-MDP to model AO in HRC. Also, we proposed a corresponding novel IRL technique — oDec-AIRL, that uses expert demonstrations to learn a reward function sophisticated enough to solve OHRC problems. We conducted pilot studies where the human and robot collaboratively assembled a table. Using the results of 2 task variations for each of our 6 human subjects, we showed that AO enhances HRC by allowing the robot to call the human for assistance at the right time.

Finally, in our last contribution, motivated by the understanding that human behavior evolves due to latent decision-making factors, we developed a novel multi-agent model and an associated IRL method. This approach learns from human-human team demonstrations, enabling a collaborative robot (cobot) to work effectively with a human by accommodating dynamic changes in the human's type, thereby enhancing collaboration within a shared workspace. Our results indicate that our novel TB-Dec-AIRL method outperforms the previous decentralized IRL method, Dec-AIRL (which does not account for agent types). Although our model is designed for human-robot collaboration (HRC), it applies to any multiagent domain in related settings. Thus, TB-Dec-AIRL brings HRC one step closer to real-world deployment in collaborative environments.

While several conceptual and practical hurdles remain in the path to smooth human-robot teaming, in this thesis, we tackled and solved a few key challenges. We envision a future where humans and robots work side-by-side to achieve a level of productivity and efficiency previously considered insurmountable. We believe that addressing the practical limitations and advancing the efficiency of HRC through improved inverse learning techniques will be crucial. We expect our work to serve as a foundation for future research, paving the way for seamless human-robot collaboration and the realization of a productive and efficient partnership between humans and robots.

Appendix A

Technical Challenges

Collaborative teamwork within a shared workspace presents a slew of challenges that include modeling, behavioral, and technical challenges. The first two are addressed through IRL policies as described in the previous chapters, however, for seamless execution of these learned cobot policies in a shared workspace, we need to solve an array of non-trivial technical challenges.

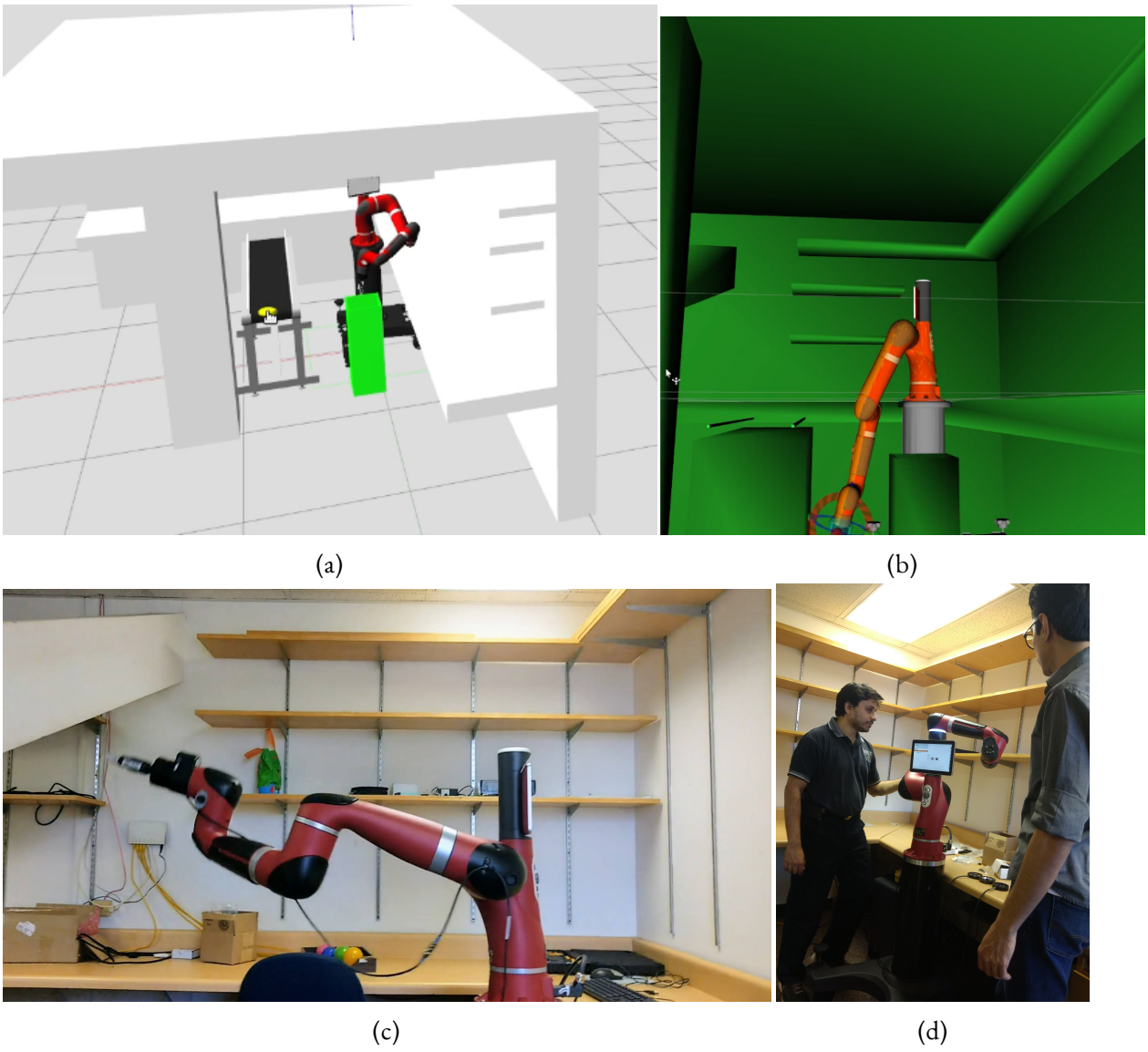
Figure A.1: Cobot UR3e deployed beside an industrial conveyor at a real-world onion sorting shed.

Cobots working in dynamic environments need to constantly update their estimate of the world around them using feedback obtained from sensors such as RGB-D cameras. Based on the updated model of the world, the cobot has to adjust its movement and path planning. The first step to this process is to obtain an accurate estimate of the static obstacles (e.g. walls, tables, conveyor, etc) within its reachable workspace [1].

[1]The reachable workspace of a cobot is usually a (hemi) sphere spanning from the cobot's base joint to the tip of the end-effector when the cobot is linearly stretched out

A.1 Mapping Static Obstacles

When our robotics research began in 2019, the cobot Sawyer was situated in a room containing several static obstacles such as pillars, shelves, and tables. In order to provide the cobot with the knowledge of the static obstacles around it, we measured all obstacles within its reachable workspace and built accurate 3D CAD models to provide accurate feedback to the cobot during path planning.



(a) (b)

(c) (d)

Figure A.2: Static obstacles designed according to real-world specs and built into the simulation. (a) Sawyer cobot simulation built within the room, with the conveyor, bad-bin, tables, and the walls around it (b) Sawyer cobot planning within Moveit Rviz around the static obstacles. (c) A photograph showing Sawyer's path planning that avoids static obstacles. (d) Setting up Sawyer at the lab for the first time.

As can be seen from Fig. A.2a and Fig. A.2b, the entire room around the cobot was mapped out accurately using CAD models and loaded into Gazebo and Moveit to provide ROS the requisite information to plan around obstacles. This simulated environment was first built on Ubuntu 16.04 - ROS Kinetic.

A.2 Path Planning

In order for human and robotic agents to work in a shared workspace, the cobot needs to move predictably and follow intuitive paths that make the human feel safe. However, the default planning pipeline used in robot operating system (ROS) - open motion planning library (OMPL), generates plans by default, in the Cartesian space [2].

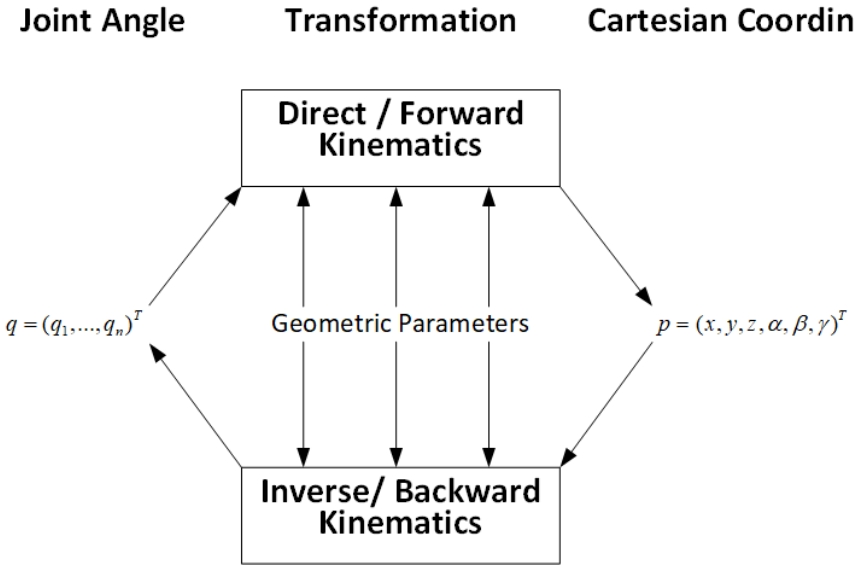


Figure A.3: Forward kinematics (FK) is when the joint angles of the cobot are known and the corresponding end-effector coordinates in Cartesian space are to be calculated. Inverse kinematics (IK) is when the desired end-effector coordinates are known and the corresponding joint angles are to be calculated. Image source: Wikipedia.

As inverse kinematics (IK) is ill-posed, the default path-planner picks the first-available plan, which often turns out to be convoluted and unsuitable for shared workspaces. Therefore, the appropriate way to plan a path would be in the joint-space (also known as configuration-space (C-space)) [3].

Arriving at a systematic solution to avoid convoluted trajectories was the second major challenge we solved. In order to make the default planners follow more intuitive trajectories, we tried applying kinematic constraints by restricting movement along one or more axes while planning. This provided

[2]The Cartesian coordinates (also called rectangular coordinates) of a point are a pair of numbers (in two-dimensions x,y) or a triplet of numbers (in three-dimensions x,y,z) that specify signed distances from the coordinate axis.

[3]The configuration of a cobot is a specification of the position of all joints of the cobot, and the space of all such configurations is called the configuration space (C-space).

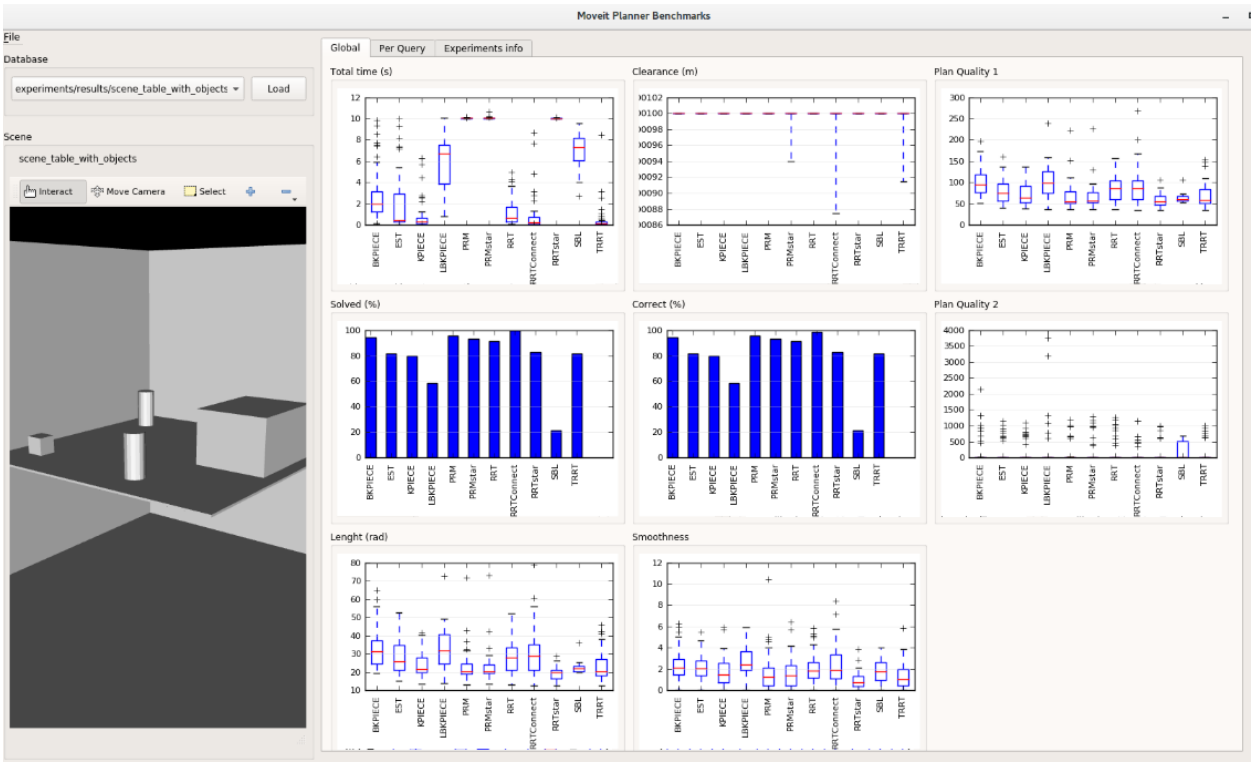


Figure A.4: "Shadow Robot Company" (n.d.) performed rigorous testing of the different planners available in OMPL and provided researchers with the tools to solve their specific problems. They use the well-known motion planning framework MoveIt!, and arrive at the results shown in the figure.

slightly improved trajectories, however, failed to find a valid plan multiple times; due to the available space for planning being severely restricted. Finally, after experimenting with an array of planners and multiple variations of kinematic constraints, we found a way to implement the entire planning in C-space. This was a ***crucial breakthrough*** that led to smooth, short, and intuitive trajectories.

Finally, the Moveit PID controller (Visioli, 2006), as shown in Fig. A.5 requires periodic tuning since it is relatively a primitive form of control system for precise pick-and-place applications. The PID errors combined with the tolerance of Moveit path-planning add up to a margin of error over time during continuous manipulation. We had to tinker and fine-tune these tolerances from time to time to maintain the precision and accuracy of cobot movement.

A.3 Handling Cobot Safety Violations

Cobots by definition are built to work collaboratively in shared workspaces. However, these features come with caveats. A cobot performs safe manipulation alongside humans at the cost of high speed and efficiency

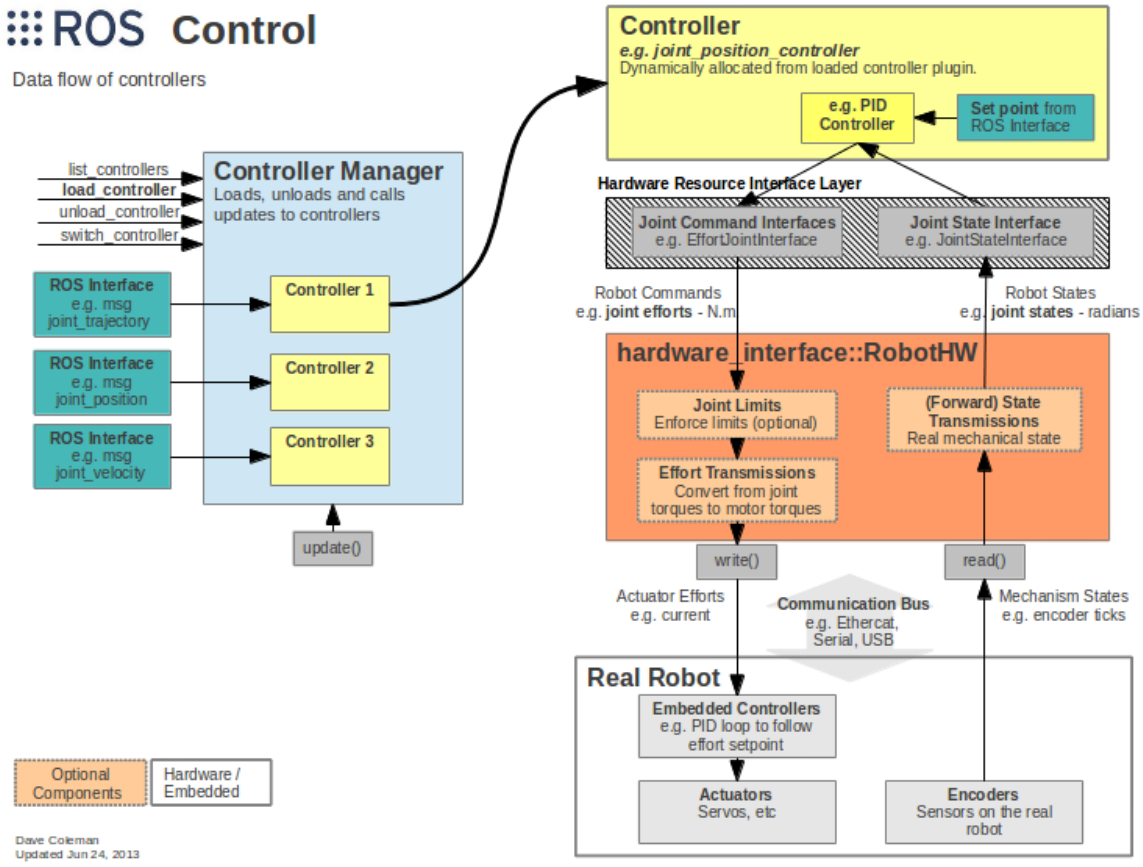


Figure A.5: Figure shows the architecture of Moveit ROS control. Credit: Moveit.

that industrial robots (that have no safety features built in) possess. To provide some perspective on the limitations posed by these safety features, consider our onion-sorting domain, where a human and cobot are opposite each other, sorting onions on a conveyor line. Through empirical estimates, we observed that a human takes around $3 - 5$ seconds to sort an onion (even considering thorough inspection). While a cobot, moving at its default speed takes around $15 - 20$ seconds to sort an onion.

This means that there will be a marked difference in the number of onions sorted by the human as compared to the cobot. In order to allow the decision-making pipelines to obtain sensor feedback, use it to form the agent(s) states and provide policy actions to perform, the human has to significantly slow down (and even wait idly) at many places to allow the cobot to catch up. This diminishes the experience of the human teammate working alongside the cobot.

While the obvious solution seems to be to increase the speed of the cobot, the built-in safety features of the cobot restrict moving the cobot beyond a prescribed velocity. Additionally, moving at increased

Figure A.6: Figure shows the protective stop occurring on a Universal Robots cobot while manipulating objects. Image credit: Universal Robots

speeds means that the transmission and reception of the signals between the cobot and the host machine needs to match that speed. Also, the other parts of the HRC architecture, such as the perception pipeline and the intercommunication between internal nodes of the control stack must synchronize accordingly. Finally, increased velocity increases error in cobot joint movement and raises the chances of the cobot running into protective stops [4].

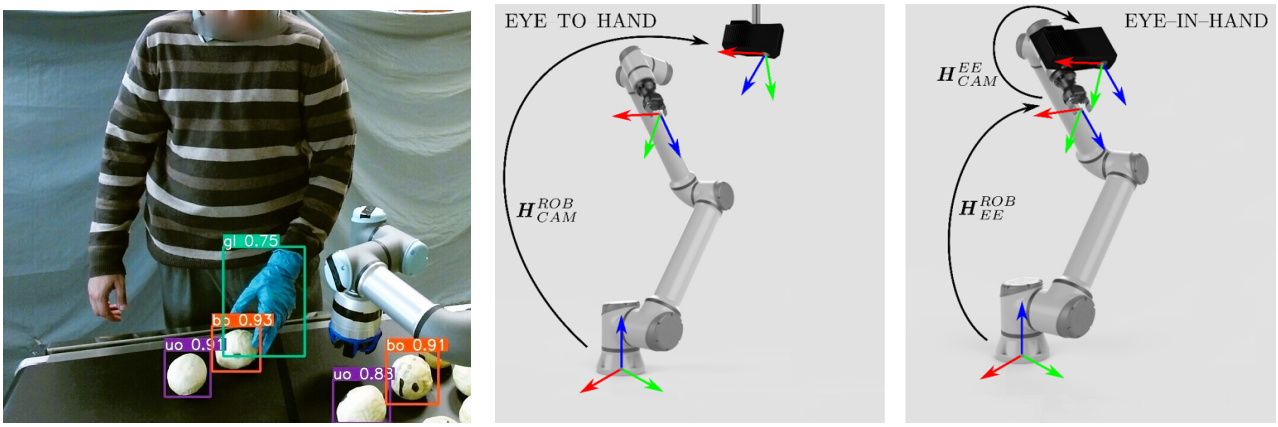
In order to handle these issues, we had to modify the cobot's ROS driver code to override certain safety clauses, improve communication between the cobot and the host machine, and increase the threshold for protective stops. To understand and implement said overrides took non-trivial time and effort. Currently, the cobot sorts one onion in 7 seconds on average and runs into much fewer safety violations.

A.4 Perception Pipeline Challenges

Perception is arguably the most widely used feedback system in robotics and AI. Consequently, processing such sensor feedback appropriately to recognize desiderata is critical to learning and decision-making. For instance, for a cobot to accurately localize the object it needs to manipulate, especially in a shared

[4]Protective stop is a state of a cobot in which the cobot abruptly stops, its joints get locked and the cobot's controller gets disconnects from the host machine to prevent anomalous motions.

workspace, is key to smooth task completion and effective collaboration. The perception pipeline needs to perform multiple functionalities such as object detection, classification, localization and transformation into the world coordinate system. Additionally, for safe HRI, the pipeline must also provide feedback about the human within the shared workspace.



(a) (b)

Figure A.7: (a) Live feed from a Realsense D435 RGB-D camera being processed by the YOLOv7-DeepSort network in the perception pipeline to recognize objects on the conveyor line and the human within the shared workspace. The bounding box labels mention if it is: gl - glove, uo - unblemished onion, bo - blemished onion. (b) The two most commonly used rigid transformation methods in robotics - eye-to-hand (camera not attached to the gripper) and eye-in-hand (camera is attached to the gripper).

While in retrospect coherently putting the pieces together seems almost intuitive, finding the exact goals that the perception pipeline must achieve, the right tools to use for this, and the correct order to use them in was no mean feat. In 2019 when the research began, the software and hardware resources available to us, were sparse.

A.4.1 Classification Networks

We started with YOLOv3 (You Only Look Once - version 3) in 2019, a state-of-the-art object classifier network at that time. Despite being the best, YOLOv3 was prone to misclassifications, missed detections, and excessive usage of computational resources.

Alongside the progressive collection of training data for YOLO and the countless manhours spent labeling all the data, we also gradually moved from YOLOv3 to YOLOv5 to YOLOv7 and finally to a YOLOv7−DeepSort combination that best recognizes objects and keeps track of them effectively. Currently, we have over 9000 labeled images that comprise both faux and real (blemished and unblemished) onions. While we have noticed a marked difference in precision and accuracy of the latest version of the network (notice Fig. A.7a), it still seems to be heavily influenced by changes in ambient lighting, object scale, and scene background. More diverse data collection and training is currently underway.

A.4.2 Rigid Transformations

Before we talk about the challenges in computing rigid transforms, we need to first understand the basics of these transforms and why they are important.

Brief Overview of Homogenous Transformations

Classical computer vision and robotics use analytical Linear Algebra techniques called rigid transformations that allow the conversion of an object's 6D pose (x, y, z, roll (ϕ), pitch (θ), yaw (ψ)) from one frame of reference to another. Usually, these transformations are applied to convert a pose estimate from the camera frame to the cobot frame of reference.

Let's consider a point P in a 2D or 3D space. A rigid transformation of this point can be represented mathematically as $P' = R \cdot P + T$ where P' is the transformed point, R is a rotation matrix, P is the original point, and T is a translation vector. In 2D, the rotation matrix R is a 2×2 matrix:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

where θ is the rotation angle. In 3D, the rotation matrix R is a 3×3 matrix. Various representations such as Euler angles, axis-angle representation, or quaternions can be used to describe rotations in 3D. Typically quaternions are preferred in robotics due to their advantages over other representations like Euler angles:

1. **No Gimbal Lock:** Quaternions avoid gimbal lock, a problem encountered with Euler angles where two axes become parallel, losing a degree of freedom.
2. **Numerical Stability:** They are numerically stable, especially for continuous rotations over time.
3. **Efficiency:** Quaternion operations are computationally efficient compared to rotation matrices or Euler angles.

Euler angles (ϕ, θ, ψ) to angle-axis to quaternion conversion:

1. Compute the axis of rotation (x, y, z) using the following formulas:

$$x = \frac{\sin(\phi/2)\cos(\theta/2)\sin(\psi/2) + \cos(\phi/2)\sin(\theta/2)\cos(\psi/2)}{\sqrt{1 - \cos^2(\theta/2)\cos^2(\psi/2)}}$$
$$y = \frac{-\sin(\phi/2)\cos(\theta/2)\cos(\psi/2) + \cos(\phi/2)\sin(\theta/2)\sin(\psi/2)}{\sqrt{1 - \cos^2(\theta/2)\cos^2(\psi/2)}}$$
$$z = \frac{\sin(\phi/2)\sin(\theta/2)\cos(\psi/2) + \cos(\phi/2)\cos(\theta/2)\sin(\psi/2)}{\sqrt{1 - \cos^2(\theta/2)\cos^2(\psi/2)}}$$

2. Compute the angle of rotation α:

$$\alpha = 2\cos^{-1}\left(\cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right)\right)$$

Then we can use the angle-axis output (x, y, z, α) to convert to quaternion $q = \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right)(xi + yj + zk)$ where i, j, and k are imaginary units.

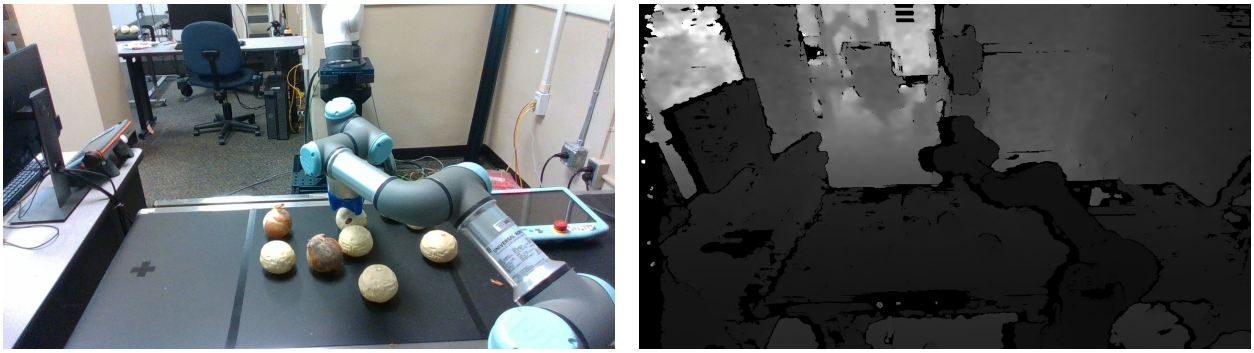
A homogeneous transformation matrix represents the transformation between two coordinate frames in a 3D space. In the context of transforming from the camera frame to the robot frame, let $T_{\text{cam\_to\_robot}}$ be the homogeneous transformation matrix. It can be written as:

$$T_{\text{cam\_to\_robot}} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

where R is a 3×3 rotation matrix representing the rotation from the camera frame to the robot frame, and t is a 3×1 translation vector representing the translation from the camera frame to the robot frame. This transformation matrix encapsulates both rotation and translation between the camera and robot frames. By understanding these mathematical representations, one can effectively perform rigid transformations between different coordinate frames, facilitating tasks such as object localization and manipulation.

Challenges in Computing Rigid Transforms

In order to compute the rigid transform of a given point in space, the cartesian coordinates (x, y, z) of the point in the camera frame must be obtained accurately. Using the camera intrinsics (focal length, optical center, scaling, etc) and camera extrinsics (rotation and translation of camera from robot origin), the homogenous transform of this 3D point in space can be calculated with respect to the robot frame.



(a) (b)

Figure A.8: (a) A sample RGB frame from a Realsense D435 RGB-D camera overlooking the workspace. (b) The corresponding depth frame from the same camera. Note that the dark black portions in the depth frame correspond to missing depth values and white portions lie outside the depth range.

However, since we find the objects of interest in a scene using YOLO and as established previously, YOLO's bounding boxes are inaccurate in several instances. This causes the (x,y) of the object in the 2D RGB frame to be estimated inaccurately. Added to this, in all RGB-D cameras, the max depth resolution is much lower compared to the max RGB resolution. This means that the depth estimate, in the best case, is off by a few centimeters, and in the worst case, unavailable due to the absence of a depth point at a given location. This causes the z coordinate of our object in the camera frame also to be incorrect.

Also, the estimated camera location, whether measured through manual methods or hand-eye calibration packages, carries a margin of error. This combined with the object's 3D coordinate error adds to an error of up to $4 - 5$cm along any axis during a particular measurement. By switching to the YOLOv7-DeepSort combination with better training data, and averaging out the depth values within the region of interest around the desired point, we were able to contain the total error within 2cm.

A.5 Hardware, software, and other challenges

When I began this research in Spring 2019, we were equipped with a Kinect v2 RGB-D camera and had recently acquired a Sawyer 7-DOF collaborative robot. However, Rethink Robotics, the manufacturer of Sawyer, had just declared bankruptcy, leaving us without customer support for any of their products. Additionally, the Robotiq 2f-85 gripper we purchased was not directly compatible with Sawyer, making the setup process unnecessarily challenging. Compounding these issues were the limitations and bugs in the then-stable version of the Robot Operating System (ROS) - Kinetic, making it quite difficult to get the robot and camera functional for our use case.

Eventually, I managed to stabilize the robot's operation and integrated the Kinect camera with YOLOv3, which was the latest version of YOLO at the time, to train and recognize onions on a conveyor. We gradually upgraded our hardware, moving to a Universal Robots UR3e cobot and acquiring a Realsense D435 RGB-D camera, along with a powerful PC with a GPU. These upgrades significantly improved the efficiency of our entire pipeline. However, some challenges persisted.

The robot's movement was quite slow, and every time I attempted to increase its speed, the ROS driver would disconnect. After thorough investigation, I realized the issue stemmed from the driver communicating with the robot through a socket connection. The driver software had a built-in timeout with a keep-alive counter to ensure signals were exchanged at a certain frequency. Any slight change to this setting would break the socket connection, requiring a system restart. Despite numerous people raising issues on their GitHub repository over the past few years, this remains an unresolved problem. To bypass this issue, I modified the frequency of establishing a connection with the robot to a much higher value, such as 1500, allowing the driver to reconnect faster than the connection was being dropped. I then adjusted the velocity and acceleration limits mentioned in this file, speeding up the robot beyond its prescribed limits.

While these fixes temporarily allowed the robot to remain connected and move at a much faster velocity, the camera detections remained unreliable for several reasons. Firstly, YOLO's limitations prevented it from consistently detecting and classifying all objects within the scene. Secondly, a crucial oversight was

that I was directly sending the ROS image from the live camera feed to YOLO for detection. In contrast, YOLO's offline loader for training and testing used OpenCV to load images and internally swapped the axes to convert BGR images to RGB. Since I was bypassing this conversion, YOLO was only using data from one correct channel to make predictions. This was a **major bug fix** that drastically improved the quality of YOLO's predictions.

Following this, there was still a certain level of error in the resulting transforms of objects within the scene, especially when they were oddly shaped (as onions usually are) and the conveyor was moving. I identified several contributing factors to this issue. Firstly, since the detections were obtained using bounding boxes, the centroid of these boxes did not necessarily correspond to the centroid of the objects. We needed to switch to a segmentation-based detection model to obtain better results. Secondly, I expanded a box around the centroid of the bounding box until a region of interest (ROI) was obtained that was void of null values from the missing points in the depth point cloud. Then I used either the maximum or average of these ROI points to determine the transform of the object.

However, I later realized that this method was inefficient, particularly for oddly shaped objects. In the absence of a segmentation model to get the exact boundaries of the object, I used a circular ROI (approximately 85% the size of the bounding box) and obtained the average of the 95th percentile and the 5th percentile within this circle to estimate the correct depth. These values were empirically determined to work well for the specific use case of onion-sorting within the limited resources available to us.

While these still do not exhaustively cover all the implementation challenges faced, they summarize the majority of the significant challenges faced and fixed along the way.

A.6 Human-robot collaboration challenges

Since most challenges occurring during human-robot collaboration can be attributed to one of the aforementioned challenges, in this section we will briefly highlight some key aspects that are challenging to implement in real-world HRC.

1. **Tracking Human State and Actions**: Maintaining awareness of the human's movements is challenging when the cobot occludes certain areas, YOLO misses detections, or the human steps out of the camera's view. Solutions could involve adding sensors or predictive models to estimate positions when visual input is absent.

2. **Preempting MoveIt Trajectories**: Adapting MoveIt trajectories dynamically is difficult with existing robotic path-planning. To navigate around obstacles, smoother, flexible path-planning can enhance safety and trajectory control.

3. **Safe Path-Planning with Concurrent Movements**: As both the cobot and human move in close proximity, dynamic path adjustments are required to prevent collisions. Cooperative algorithms help the cobot adjust paths based on human movement predictions, ensuring mutual safety.

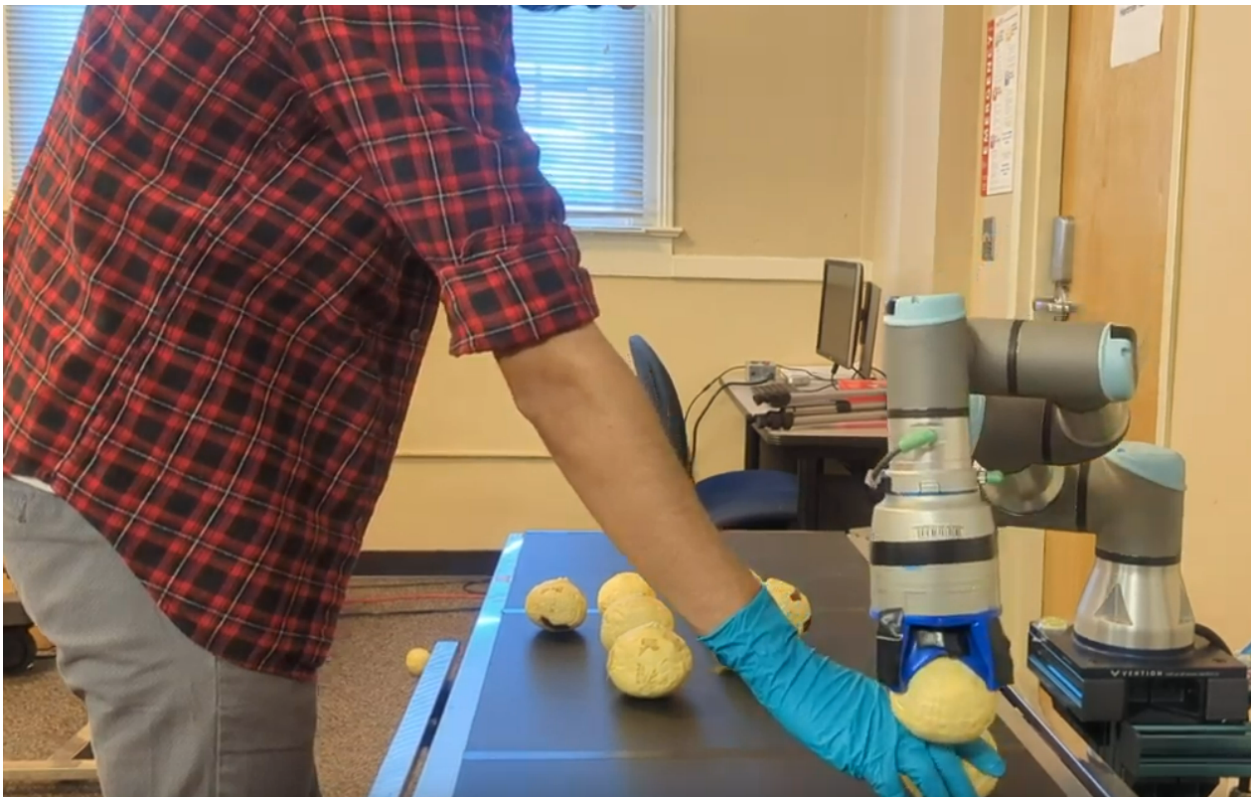


Figure A.9: Figure shows a cobot colliding with human after they entered the workspace during cobot trajectory.

4. **Efficient but Predictable Cobot Movements**: Balancing speed and predictability allows the cobot to be efficient without startling the human. Setting specific acceleration profiles within comfortable limits for the human improves task flow and comfort.

5. **Dynamic Goal Adjustment**: If the human picks up an object the cobot is targeting, the cobot should dynamically switch to another task, requiring real-time human action monitoring and goal adaptation to prevent redundancy.

Together, these adjustments create a safer and more efficient collaborative environment. Check out some interesting tools available in current version of Moveit2: TOTG, efficient pick, better grasp, movement constraints.

Appendix B

MMAP-BIRL

B.1 Algorithm for MMAP-BIRL using Gradient Ascent

Algorithm 1: MMAP-BIRL

Input: MDP, $\mathcal{Y}$, step-size δ_n, ϵ
Output: Learned reward function $R_{\boldsymbol{\theta}}$

Sample $R_{\boldsymbol{\theta}}$ from the prior distribution
Initialize $\Pi \leftarrow \varnothing$, $\delta \leftarrow \infty$
if $\nabla_{\boldsymbol{\theta}} Pr(R_{\boldsymbol{\theta}}|\mathcal{Y})$ *not in* Π **then**
    if *no. of occlusions* $> o$ **then**
        $\mathcal{Z} \leftarrow$ Bidirectional_Search(MDP, $\mathcal{Y}$)
        $\nabla_{\boldsymbol{\theta}} Pr(R_{\boldsymbol{\theta}}|\mathcal{Y}) \leftarrow$ Compute_MMAP _Gradient(MDP, $\mathcal{Y}$, $\mathcal{Z}$)
$\pi \leftarrow$ Solve_MDP($R_{\boldsymbol{\theta}}$)
$H^{\pi} \leftarrow$ Compute_Reward_Optimality_Region(π) as shown in Eq. 2.12
$\Pi \leftarrow \{\langle \pi, H^{\pi}, \nabla_{\boldsymbol{\theta}} P(R_{\boldsymbol{\theta}}|\mathcal{Y})\rangle\}$
while $\delta > \epsilon(1-\gamma)/\gamma$ **do**
    $R_{\boldsymbol{\theta}_{new}} \leftarrow R_{\boldsymbol{\theta}} + \delta_n \nabla_{\boldsymbol{\theta}} Pr(R_{\boldsymbol{\theta}}|\mathcal{Y})$
    if $R_{\boldsymbol{\theta}_{new}}$ *is not in the reward optimality region* H^{π} **then**
        Repeat steps 3 to 8 using $R_{\boldsymbol{\theta},new}$
        if *isNewEntry*$(\langle \pi, H^{\pi}, \nabla_{\boldsymbol{\theta}} P(R_{\boldsymbol{\theta}_{new}}|\mathcal{Y})\rangle)$ **then**
            add $\langle \pi, H^{\pi}, \nabla_{\boldsymbol{\theta}} P(R_{\boldsymbol{\theta}_{new}}|\mathcal{Y})\rangle$ to Π
    else
        ReuseCachedGradient(Π)
    $\delta \leftarrow |R_{\boldsymbol{\theta}} - R_{\boldsymbol{\theta}_{new}}|$
    $R_{\boldsymbol{\theta}} \leftarrow R_{\boldsymbol{\theta}_{new}}$
return $R_{\boldsymbol{\theta}}$

The algorithm for MMAP-BIRL is shown in Algorithm 1. It computes the initial gradient $\nabla_{\boldsymbol{\theta}} P(R_{\boldsymbol{\theta}}|\mathcal{Y})$ for the initially sampled weights (line 1), performs forward rollout (line 7) to find the policy corresponding to these weights, computes the reward optimality region (line 8) and stores all of these (line 9). Notice that the gradient requires inferring possible Z, a set considerably narrowed down by forward-backward search. Then, we repeatedly update the reward weights according to the update rule (line 11). If the weights fail to satisfy the optimality condition (line 13), we find a new gradient and proceed with the remaining steps. On convergence, we return the learned weights (line 21).

B.2 Complexity analysis

Complexity analysis of the MMAP-BIRL algorithm from Section B.1 consists of two main components: (1) obtain the gradient and its optimality region using MMAP, and (2) reuse a cached gradient if $R_{\boldsymbol{\theta}_{new}}$ is within the reward optimality region, otherwise recalculate the gradient. The first part involves a bidirectional search to obtain the list of possible candidates for the occluded block. Its worst-case complexity is $\mathcal{O}((|O_l||S||A|)^{d/2})$ where d is the length of the occluded block, but typically less because not all observations are always possible. Computing the gradient involves parsing all the features for each trajectory whose complexity is $K\mathcal{T}|\mathcal{X}|$. Next, solving the MDP using policy iteration with current reward weights has the policy space as the worst-case complexity $\mathcal{O}((|S|^{|A|})$ but more typically it is $\mathcal{O}(|S| \cdot N_{iter})$, where N_{iter} denotes the number of iterations until convergence, which is variable. And, the complexity of the optimality region H^{π} computation is dominated by the multiplication of two matrices whose complexity is $\mathcal{O}(|S|^3|A|)$. The second part involves updating the reward weights using the previously computed gradient, whose complexity is K, and checking if $R_{\boldsymbol{\theta}_{new}}$ is within the range of one of the cached entries in Π, whose complexity is $\mathcal{O}(K|\Pi|)$. If a cached gradient cannot be used, then the complexity of the first part applies.

Appendix C

Dec-AIRL

C.1 Algorithm for Dec-AIRL

Algorithm 2: Dec-AIRL

Input: $\mathcal{DM}$ sans R and T; Exp trajs $\mathcal{X}^E$.
Output: Learned joint reward function R.

1 Initialize decentralized policy $\boldsymbol{\pi}$, Discriminator $D_{\boldsymbol{\theta}}$.
2 **for** $iter \leftarrow 0$ **to** $train\_iters$ **do**
3 Generate joint trajectories $\hat{\mathcal{X}}$ using $\boldsymbol{\pi}$.
4 Sample joint (s, a) pairs $\hat{\mathcal{Y}}$, $\mathcal{Y}^E$ from $\hat{\mathcal{X}}$, $\mathcal{X}^E$, respectively.
5 **for** $ep \leftarrow 0$ **to** $disc\_epochs$ **do**
6 Train disc $D_{\boldsymbol{\theta}}$ via BCE to classify expert data $\mathcal{Y}^E$ from learned policy data $\hat{\mathcal{Y}}$.
7 Update reward $\hat{R} \leftarrow \log D_{\boldsymbol{\theta}}(S, A, S') - \log(1 - D_{\boldsymbol{\theta}}(S, A, S'))$.
8 **for** $ep \leftarrow 0$ **to** $generator\_epochs$ **do**
9 Train generator $G(\hat{R}) \leftarrow$ Dec-PPO.
10 Get updated policy $\boldsymbol{\pi}^{\boldsymbol{L}} \leftarrow G(\hat{R})$.
11 **return** $\hat{R}$, $\boldsymbol{\pi}^{\boldsymbol{L}}$

Algorithm 2 shows the algorithm for Dec-AIRL. It takes the model $\mathcal{DM}$ without the reward function and transition function (as we run model free), the expert trajectories $\mathcal{X}^E$ as input. The aim is to learn the common reward function for the task R. It starts by generating a random decentralized policy vector $\hat{\boldsymbol{\pi}}$ (line 1), and a discriminator $D_{\boldsymbol{\theta}}$ with random weights is initialized. The algorithm begins its iterative updates until the end of training iterations (line 2). In each iteration, joint trajectories $\hat{\mathcal{X}}$ of the agents are generated using the current policy $\hat{\boldsymbol{\pi}}$, followed by picking some state-action pairs $\hat{\mathcal{Y}}$ from $\hat{\mathcal{X}}$ and some state-action pairs $\mathcal{Y}^E$ from $\mathcal{X}^E$ (line 4). Using these samples, the discriminator is trained to differentiate between the expert and learned sample data using BCE loss (line 6). From the trained discriminator, the updated reward $\hat{R}$ is then extracted in line 7. Next, the generator $G(\hat{R})$ is trained with a centralized critic and decentralized actors-based PPO (Dec-PPO) to generate the forward rollout vector of policies. Finally, the learned reward function $\hat{R}$ and converged policy $\boldsymbol{\pi}^L$ are returned by the algorithm.

Appendix D

Open-Dec-AIRL

D.1 Algorithm for Open-Dec-AIRL

Algorithm 3: oDec-AIRL

Input: $\mathcal{ODM}$ sans R_c and T; Exp trajs $\mathcal{X}^E$.
Output: Learned common reward function R_c

1 Initialize decentralized policy $\boldsymbol{\pi}$, Discriminator $D_{\boldsymbol{\theta}}$.
2 **for** $iter \leftarrow 0$ **to** $train\_iters$ **do**
3 Generate joint trajectories $\hat{\mathcal{X}}$ using $\boldsymbol{\pi}$
4 Sample joint (c, s, a) minibatch $\hat{\mathcal{Y}}$ and $\mathcal{Y}^E$ from $\hat{\mathcal{X}}$ and $\mathcal{X}^E$, respectively.
5 **for** $ep \leftarrow 0$ **to** $discriminator\_epochs$ **do**
6 Train discriminator $D_{\boldsymbol{\theta}}$ to minimize $KL(\mathrm{P}_{\pi}(c, s, a)||\mathrm{P}_{exp}(c, s, a))$.
7 Update reward $R_c \leftarrow \log D_{\boldsymbol{\theta}}(C, S, A, C', S') - \log(1 - D_{\boldsymbol{\theta}}(C, S, A, C', S'))$
8 **for** $ep \leftarrow 0$ **to** $generator\_epochs$ **do**
9 Train generator $G(R_c) \leftarrow$ oDec-PPO.
10 Get updated policy $\boldsymbol{\pi^L} \leftarrow G(R_c)$.
11 **return** R_c, $\boldsymbol{\pi^L}$

The oDec-AIRL algorithm shown in Algorithm 3 takes the oDec-MDP ($\mathcal{ODM}$) without the reward function and transition function, and the expert trajectories $\mathcal{X}^E$ as input. The goal is to learn a common reward function for the task R_c that best explains the behavior seen in $\mathcal{X}^E$, and the corresponding vector of learned policies.

The algorithm begins by initializing a random decentralized policy vector $\hat{\boldsymbol{\pi}}$ (line 1), and a discriminator $D_{\boldsymbol{\theta}}$ with random weights. This step repeats until the end of training iterations (line 2). In every iteration, it generates joint trajectories $\hat{\mathcal{X}}$ of the agents using the current policy vector $\hat{\boldsymbol{\pi}}$. It then samples minibatches of team*id*-state-action triplets from $\hat{\mathcal{X}}$ and $\mathcal{X}^E$ to yield $\hat{\mathcal{Y}}$ and $\mathcal{Y}^E$ respectively (line 4). It trains $D_{\boldsymbol{\theta}}$ over these minibatches to minimize the reverse KL divergence between the expert and learned distribution (line 6).

Using $D_{\boldsymbol{\theta}}$'s confusion it extracts an updated reward R_c in line 7. This reward function R_c is then provided as an input to train the generator $G(R_c)$ using oDec-PPO that learns the forward rollout vector of policies. Finally, the learned reward function R_c and converged policy vector $\boldsymbol{\pi}^L$ are returned by the algorithm.

Appendix E

TB-Dec-AIRL

E.1 Algorithm

Algorithm 4: TB-Dec-AIRL

Input: $\mathcal{TB} - \mathcal{DM}$ sans R and T; Exp trajs $\mathcal{X}^E$ sans other agent types.
Output: Learned common type-based reward function R

1 Initialize generator ($G_{\boldsymbol{\omega}}$) with policy vector $\boldsymbol{\pi^L}$, Discriminator $D_{\boldsymbol{\alpha}}$, and Belief module $B_{\boldsymbol{\phi}}$.
2 **for** $iter \leftarrow 0$ **to** $train\_iters$ **do**
3 Use $\boldsymbol{\pi^L}$ and $B_{\boldsymbol{\phi}}$ to step through the environment and generate joint trajectories $\langle \hat{\mathcal{X}}, \hat{b}_{\boldsymbol{\theta}} \rangle$
4 Obtain expert state-action and belief trajectories tuple $\langle \mathcal{X}^E, \hat{b}^E_{\boldsymbol{\theta}} \rangle$ by passing $\mathcal{X}^E$ through $B_{\boldsymbol{\phi}}$
5 Sample joint $(\boldsymbol{s}, b_{\boldsymbol{\theta}}, a, \boldsymbol{s}', b'_{\boldsymbol{\theta}})$ pairs from $\langle \hat{\mathcal{X}}, \hat{b}_{\boldsymbol{\theta}} \rangle$ and $\langle \mathcal{X}^E, \hat{b}_{\boldsymbol{\theta}} \rangle$, respectively
6 **for** $ep \leftarrow 0$ **to** $discriminator\_epochs$ **do**
7 Train discriminator $D_{\boldsymbol{\alpha}}$ via BCE loss to classify $\langle \mathcal{X}^E, \hat{b}^E_{\boldsymbol{\theta}} \rangle$ from $\langle \hat{\mathcal{X}}, \hat{b}_{\boldsymbol{\theta}} \rangle$
8 Update reward $R \leftarrow \log D_{\boldsymbol{\alpha}}(\cdot, \cdot, \cdot) - \log(1 - D_{\boldsymbol{\alpha}}(\cdot, \cdot, \cdot)$
9 **for** $ep \leftarrow 0$ **to** $generator\_epochs$ **do**
10 Train generator $G_{\boldsymbol{\omega}}(R) \leftarrow$ TB-Dec-PPO.
11 Get updated policy $\boldsymbol{\pi^L} \leftarrow G_{\boldsymbol{\omega}}(R)$.
12 **return** R, $\boldsymbol{\pi^L}$

The TB-Dec-AIRL algorithm (Algorithm **??**) uses our novel model $\mathcal{TB} - \mathcal{DM}$ without the reward and transition functions (as TB-Dec-AIRL is model-free) and the expert trajectories $\mathcal{X}^E$ (see **??**) to learn the task's common reward function R. It starts by generating a random decentralized policy vector $\boldsymbol{\pi^L}$ with generator $G_{\boldsymbol{\omega}}$ (line 1), loading the pre-trained belief module $B_{\boldsymbol{\phi}}$, and initializing the discriminator $D_{\boldsymbol{\alpha}}$ with random weights $\boldsymbol{\phi}$ and $\boldsymbol{\alpha}$ (see Fig. E.1). $\hat{b}^E_{\boldsymbol{\theta}}$ is then obtained by passing X^E through $B_{\boldsymbol{\phi}}$ to obtain expert belief trajectories (see Eq. (8.2)). The algorithm iterates through updates until training concludes (line 2). In each iteration, it generates joint trajectories $\langle \hat{\mathcal{X}}, \hat{b}\boldsymbol{\theta} \rangle$ using the current policy vector $\boldsymbol{\pi}$ and belief module $B_{\boldsymbol{\phi}}$ (line 3). (State, belief-state, action)-tuples are then sampled from these trajectories and from $\langle \mathcal{X}^E, \hat{b}^E_{\boldsymbol{\theta}} \rangle$ (line 5). The discriminator is trained to distinguish between expert and learned samples using BCE loss (line 7). The updated reward R is extracted from the trained

discriminator (line 8). The generator $G_{\boldsymbol{\omega}}(R)$ is then trained with a centralized critic and decentralized actors using TB-Dec-PPO to produce the policy rollout vector. Finally, the learned reward function R and the converged policy $\hat{\boldsymbol{\pi}}^L$ are returned.

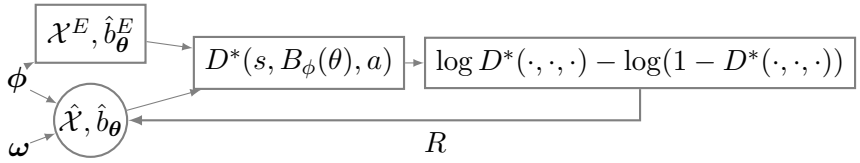


Figure E.1: Stochastic computation graph for the expectation: $\mathbb{E}_{\mathcal{X}^E, \hat{b}^E_{\boldsymbol{\theta}}}\Big[\log D^*\Big] - \mathbb{E}_{\boldsymbol{\pi}, \hat{b}_{\boldsymbol{\theta}}}\Big[\log(1 - D^*)\Big]$ where D^* represents the functional maximum over $D_{\boldsymbol{\alpha}}$. Notice that both the policy parameters ($\boldsymbol{\omega}$) and the belief parameters ($\boldsymbol{\phi}$) influence policy belief trajectories ($\hat{b}_{\boldsymbol{\theta}}$) - through environment interaction. Circles represent stochastic nodes while rectangles represent deterministic nodes

E.2 Proof of Theorem 1

We use max-norms $\|.\|$ over the sets of states, actions and next states. We refer to joint beliefs $b^t(\ldots) = \prod_i b^t_i(\ldots)$. As in Dec-AIRL, we distinguish interactive states (S_I) from non-interactive states (S_{NI}) where the reward function is given as:

$$R(s, \mathbf{a}, s') = \begin{cases} R(s, \mathbf{a}, s'), & \text{if } s \in S_I \\ \sum_i R_i(s_i, a_i, s'_i), & \text{if } s \in S_{NI} \end{cases} \tag{E.1}$$

Assumption 1 *The type transition kernel in the joint belief update acts as a contraction mapping, i.e.,*

$$\mathcal{D}_{TV}(b^{t+1}, b^t) \leq \delta \cdot \mathcal{D}_{TV}(b^t, b^{t-1}),$$

for some $0 \leq \delta < 1$*, where* $\mathcal{D}_{TV}$ *denotes the total variation distance.*

A useful consequence of this assumption is that $\frac{b^{t+1}(\theta|\ldots)}{b^t(\theta|\ldots)} \geq (1 - \delta/2), \forall \theta, t$. Note that this applies to both the expert and generated trajectories. Now, the estimates of joint rewards in non-interactive (NI) states is

$$\begin{aligned} R^t &= \sum_i R^t_i \\ &= \sum_i \log\left(\frac{D^t_i}{1 - D^t_i}\right) \end{aligned}$$

Similarly, the expert rewards in non-interactive states are $R^E = \sum_i \log\left(\frac{D_i^E}{1-D_i^E}\right)$. Now, the i-th agent's discriminator error is $\|D_i^t - D_i^E\| \leq \epsilon$, and the i-th expert's discriminator value [1] is 0.5, therefore

$$\begin{aligned}
\|R^t - R^E\| &= \|\sum_i \log\left(\frac{D_i^t}{1-D_i^t}\right) - \sum_i \log\left(\frac{D_i^E}{1-D_i^E}\right)\| \\
&= \|\sum_i \log\left(\frac{D_i^t}{D_i^E}\right) - \sum_i \log\left(\frac{1-D_i^t}{1-D_i^E}\right)\| \\
&\leq \left|\sum_i \log\left(\frac{1/2+\epsilon}{1/2-\epsilon}\right) - \sum_i \log\left(\frac{1/2-\epsilon}{1/2+\epsilon}\right)\right| \\
&= 2\sum_i \log\left(\frac{1/2+\epsilon}{1/2-\epsilon}\right) \\
&\leq 8N\epsilon \qquad \text{(E.2)}
\end{aligned}$$

when ϵ is small enough and N is the number of agents. For interactive states, the same arguments apply to the joint discriminator, but the bound corresponding to Eq. E.2 would just be 8ϵ. Therefore, we use Eq. E.2 as the dominant form of this bound.

The log-likelihood of a trajectory (X) given the reward estimate R^t is

$$\begin{aligned}
\log P(X|R^t) &\propto \sum_t \gamma^t \sum_\theta R^t(s^t, a^t, \theta) b^t(\theta|h^t) \\
&\geq \sum_t (\gamma(1-\delta/2))^t \sum_\theta R^t(\ldots, \theta)\rho_1(\theta) \\
&\geq \sum_t (\gamma(1-\delta/2))^t \sum_\theta \left|R^E(\ldots, \theta) - 8N\epsilon\right| \rho_1(\theta) \\
&\geq \log P(X|R^E) - \sum_t (\gamma(1-\delta/2))^t 8N\epsilon
\end{aligned}$$

Here, h^t is the history of states-actions preceding step t and ρ_1 is the initial belief. Thus, $LL(X|R^E) - LL(X|R^t) \leq \sum_t (\gamma(1-\delta/2))^t 8N\epsilon \leq \frac{8N\epsilon}{1-\gamma(1-\delta/2)}$. Accumulating this over a set of trajectories $\mathcal{X}$, we get $LL(\mathcal{X}|R^E) - LL(\mathcal{X}|R^t) \leq \frac{8|\mathcal{X}|N\epsilon}{1-\gamma(1-\delta/2)}$. Assuming identical priors, $P(R^t) = P(R^E)$, this also gives $LL(R^E|\mathcal{X}) - LL(R^t|\mathcal{X}) \leq \frac{8|\mathcal{X}|N\epsilon}{1-\gamma(1-\delta/2)}$.

[1] An expert's discriminator attempts to distinguish between expert trajectories and those produced by the expert's policy

References

Abbeel, P., & Ng, A. Y. (2004a). Apprenticeship learning via inverse reinforcement learning. *Proceedings of the Twenty-first International Conference on Machine Learning*, 1–8.

Abbeel, P., & Ng, A. Y. (2004b). Apprenticeship learning via inverse reinforcement learning. *Proceedings of the twenty-first international conference on Machine learning*, 1.

Achiam, J. (2018). Spinning Up in Deep Reinforcement Learning.

Adams, S., Cody, T., & Beling, P. A. (2022). A survey of inverse reinforcement learning. *Artificial Intelligence Review*, 1–40.

Albrecht, S. V., Christianos, F., & Schäfer, L. (2023). Multi-agent reinforcement learning: Foundations and modern approaches. *Massachusetts Institute of Technology: Cambridge, MA, USA*.

Albrecht, S. V., & Ramamoorthy, S. (2015). A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. *arXiv preprint arXiv:1506.01170*.

Alsaleh, R., & Sayed, T. (2021). Markov-game modeling of cyclist-pedestrian interactions in shared spaces: A multi-agent adversarial inverse reinforcement learning approach. *Transportation research part C: emerging technologies*, *128*, 103191.

Amato, C., Konidaris, G., Kaelbling, L. P., & How, J. P. (2019). Modeling and planning with macro-actions in decentralized pomdps. *Journal of Artificial Intelligence Research*, *64*, 817–859.

Amazigo, J. C., & Rubenfeld, L. A. (1980). Advanced calculus and its applications to the engineering and physical sciences. *(No Title)*.

Arjovsky, M., Chintala, S., & Bottou, L. (2017, August). Wasserstein generative adversarial networks. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 214–223, Vol. 70). PMLR. https://proceedings.mlr.press/v70/arjovsky17a.html

Arora, S., & Doshi, P. (2021a). A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, *297*, 103500.

Arora, S., & Doshi, P. (2021b). A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, *297*, 103500.

Arora, S., Doshi, P., & Banerjee, B. (2019). Online inverse reinforcement learning under occlusion. *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems*.

Arora, S., Doshi, P., & Banerjee, B. (2021). I2rl: Online inverse reinforcement learning under occlusion. *Autonomous Agents and Multi-Agent Systems*, *35*(1), 1–33.

Arora, S., Doshi, P., & Banerjee, B. (n.d.). Online inverse reinforcement learning with learned observation model. *6th Annual Conference on Robot Learning*.

Bartneck, C., Kulić, D., Croft, E., & Zoghbi, S. (2008). Godspeed questionnaire series. *International Journal of Social Robotics*.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. *International conference on machine learning*, 449–458.

Bertsekas, D. (2012). *Dynamic programming and optimal control: Volume i* (Vol. 4). Athena scientific.

Bertsekas, D. P. (2014). *Constrained optimization and lagrange multiplier methods*. Academic press.

Bogert, K. (2016). *Inverse Reinforcement Learning for Robotic Applications: Hidden Variables, Multiple Experts and Unknown Dynamics* [Doctoral dissertation, University of Georgia].

Bogert, K., & Doshi, P. (2015). Toward estimating others' transition models under occlusion for multi-robot irl. *Proceedings of the 24th International Conference on Artificial Intelligence*, 1867–1873.

Bogert, K., & Doshi, P. (2017). Scaling expectation-maximization for inverse reinforcement learning to multiple robots under occlusion. *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 522–529.

Bogert, K., Lin, J. F.-S., Doshi, P., & Kulic, D. (2016). Expectation-maximization for inverse reinforcement learning with hidden data. *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*, 1034–1042.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016a). Openai gym.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016b). Openai gym.

Brown, D. S., Goo, W., & Niekum, S. (2020). Better-than-demonstrator imitation learning via automatically-ranked demonstrations. *Conference on robot learning*, 330–359.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1–43.

Chandrasekaran, M., Eck, A., Doshi, P., & Soh, L. (2016). Individual planning in open and typed agent systems. *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, 82–91.

Chen, B., Chen, X., Timsina, A., & Soh, L.-K. (2015). Considering agent and task openness in ad hoc team formation. *AAMAS*, 1861–1862.

Chen, L., Paleja, R., & Gombolay, M. (2021). Learning from suboptimal demonstration via self-supervised reward regression. *Conference on robot learning*, 1262–1277.

Chen, M., Nikolaidis, S., Soh, H., Hsu, D., & Srinivasa, S. (2020). Trust-aware decision making for human-robot collaboration: Model learning and planning. *ACM Transactions on Human-Robot Interaction (THRI)*, *9*(2), 1–23.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Choi, J., & Kim, K.-E. (2011a). Map inference for bayesian inverse reinforcement learning. *Advances in Neural Information Processing Systems*, 1989–1997.

Choi, J., & Kim, K.-E. (2011b). Inverse reinforcement learning in partially observable environments. *Journal of Machine Learning Research*, *12*, 691–730.

Cohen, J., Dibangoye, J.-S., & Mouaddib, A.-I. (2017). Open decentralized pomdps. *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, 977–984.

Coumans, E., & Bai, Y. (2021). Pybullet quickstart guide.

Courcoubetis, C., & Weber, R. (2003). Lagrangian methods for constrained optimization. *Wiley-interscience series in systems and optimization*, 333.

Cover, T. M., Thomas, J. A., et al. (1991). Entropy, relative entropy and mutual information. *Elements of information theory*, *2*(1), 12–13.

Cziko, G. A. (1989). Unpredictability and indeterminism in human behavior: Arguments and implications for educational research. *Educational researcher*, *18*(3), 17–25.

Daley, B. (2021). Gym classics.

Eck, A., Shah, M., Doshi, P., & Soh, L.-K. (2020). Scalable decision-theoretic planning in open and typed multiagent systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(05), 7127–7134.

Eck, A., Soh, L.-K., & Doshi, P. (2023). Decision making in open agent systems. *AI Magazine*.

Erickson, Z., Gangaram, V., Kapusta, A., Liu, C. K., & Kemp, C. C. (2020). Assistive gym: A physics simulation framework for assistive robotics. *IEEE International Conference on Robotics and Automation (ICRA)*, *1*(None), 1–10.

Face, H. (n.d.). *Deep rl course*.

Ferber, J., & Weiss, G. (1999). *Multi-agent systems: An introduction to distributed artificial intelligence* (Vol. 1). Addison-wesley Reading.

Finn, C., Christiano, P., Abbeel, P., & Levine, S. (2016). A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*.

Finn, C., Levine, S., & Abbeel, P. (2016). Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization. *arXiv preprint*, *arXiv:1603.00448*, 0–10.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, *3*(4), 128–135.

Fu, J., Luo, K., & Levine, S. (2018). Learning robust rewards with adverserial inverse reinforcement learning. *International Conference on Learning Representations*, 1–10. https://openreview.net/forum?id=rkHywl-A-

Fu, M. C. (2006). Gradient estimation. *Handbooks in operations research and management science*, *13*, 575–616.

Garg, D., Chakraborty, S., Cundy, C., Song, J., & Ermon, S. (2021). Iq-learn: Inverse soft-q learning for imitation. *Advances in Neural Information Processing Systems*, *34*, 4028–4039.

Geng, S., Nassif, H., Manzanares, C., Reppen, M., & Sircar, R. (2020). Deep pqr: Solving inverse reinforcement learning using anchor actions. *International Conference on Machine Learning*, 3431–3441.

Ghasemipour, S. K. S., Zemel, R., & Gu, S. (2020a). A divergence minimization perspective on imitation learning methods. *Conference on Robot Learning*, 1259–1277.

Ghasemipour, S. K. S., Zemel, R., & Gu, S. (2020b). A divergence minimization perspective on imitation learning methods. *Conference on Robot Learning*, 1259–1277.

Gilbert, N. (2019). *Agent-based models*. Sage Publications.

Gleave, A., & Toyer, S. (2022). A primer on maximum causal entropy inverse reinforcement learning. *arXiv preprint arXiv:2203.11409*, *1*, arxiv.

Glynn, P. W. (1990). Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, *33*(10), 75–84.

Glynn, P. W., & Iglehart, D. L. (1989). Importance sampling for stochastic simulations. *Management science*, *35*(11), 1367–1392.

Gmytrasiewicz, P. J., & Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, *24*, 49–79.

Goldman, C. V., & Zilberstein, S. (2003). Optimizing information exchange in cooperative multi-agent systems. *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 137–144.

Goldman, C. V., & Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of artificial intelligence research*, *22*, 143–174.

Goldman, C. V., & Zilberstein, S. (2008). Communication-based decomposition mechanisms for decentralized mdps. *Journal of Artificial Intelligence Research*, *32*, 169–202.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.

Gopalan, N., & Tellex, S. (2015). Modeling and solving human-robot collaborative tasks using pomdps. *RSS Workshop on Model Learning for Human-Robot Communication*, *32*(4), 590–628.

Gordon, G., & Tibshirani, R. (2012). Karush-kuhn-tucker conditions. *Optimization*, *10*(725/36), 725.

Gruver, N., Song, J., Kochenderfer, M. J., & Ermon, S. (2020). Multi-agent adversarial inverse reinforcement learning with latent variables. *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 1855–1857.

Guan, Z., Xu, T., & Liang, Y. (2021). When will generative adversarial imitation learning algorithms attain global convergence. *International Conference on Artificial Intelligence and Statistics*, 1117–1125.

Haarnoja, T., Tang, H., Abbeel, P., & Levine, S. (2017). Reinforcement learning with deep energy-based policies. *International Conference on Machine Learning*, 1352–1361.

He, K., Doshi, P., & Banerjee, B. (2023). Latent interactive a2c for improved rl in open many-agent systems. *arXiv preprint arXiv:2305.05159*.

Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. *Advances in neural information processing systems*, *29*, None.

Hoang, T. N., & Low, K. H. (2013). Interactive pomdp lite: Towards practical planning to predict and exploit intentions for interacting with self-interested agents. *arXiv preprint arXiv:1304.5159*.

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, *50*(2), 1–35.

Jain, V., Doshi, P., & Banerjee, B. (19). Model-free irl using maximum likelihood estimation. *AAAI Conference on Artificial Intelligence*, 3951–3958.

Jaynes, E. T. (1957). Information theory and statistical mechanics. *Phys. Rev.*, *106*, 620–630.

Jeon, W., Barde, P., Nowrouzezahrai, D., & Pineau, J. (2020). Scalable multi-agent inverse reinforcement learning via actor-attention-critic. *arXiv preprint arXiv:2002.10525*, *None*(None), None.

Jiang, R., Zhang, X., Liu, Y., Xu, Y., Zhang, X., & Zhuang, Y. (2024). Multi-agent cooperative strategy with explicit teammate modeling and targeted informative communication. *Neurocomputing*, *586*, 127638.

Jumadinova, J., Dasgupta, P., & Soh, L.-K. (2013). Strategic capability-learning for improved multiagent collaboration in ad hoc environments. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *44*(8), 1003–1014.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, *101*(1-2), 99–134.

Kakarlapudi, A., Anil, G., Eck, A., Doshi, P., & Soh, L.-K. (2022). Decision-theoretic planning with communication in open multiagent systems. *Uncertainty in Artificial Intelligence*, 938–948.

Kamar, E., Gal, Y., & Grosz, B. J. (2009). Incorporating helpful behavior into collaborative planning. *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Kassam, S. A., & Poor, H. V. (1985). Robust techniques for signal processing: A survey. *Proceedings of the IEEE*, *73*(3), 433–481.

Kim, B., & Pineau, J. (2016). Socially adaptive path planning in human environments using inverse reinforcement learning. *International Journal of Social Robotics*, *8*(1), 51–66.

Koul, A. (2019). Ma-gym: Collection of multi-agent environments based on openai gym.

Krajewski, R., Bock, J., Kloeker, L., & Eckstein, L. (2018). The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2118–2125.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, *22*(1), 79–86.

Kwon, M., Biyik, E., Talati, A., Bhasin, K., Losey, D. P., & Sadigh, D. (2020). When humans aren't optimal: Robots that collaborate with risk-aware humans. *Proceedings of the 2020 ACM/IEEE international conference on human-robot interaction*, 43–52.

Lagrange, J. L. (1853). *Mécanique analytique* (Vol. 1). Mallet-Bachelier.

Larsen, T. N., Teigen, H. Ø., Laache, T., Varagnolo, D., & Rasheed, A. (2021). Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters. *Frontiers in Robotics and AI*, *8*, 738113.

Leslie, A. M., Friedman, O., & German, T. P. (2004). Core mechanisms in 'theory of mind'. *Trends in cognitive sciences*, *8*(12), 528–533.

Levine, S., Popović, Z., & Koltun, V. (2010). Feature construction for inverse reinforcement learning. *Proceedings of the 23rd International Conference on Neural Information Processing Systems*, 1342–1350.

Lin, X., Beling, P. A., & Cogill, R. (2017). Multiagent inverse reinforcement learning for two-person zero-sum games. *IEEE Transactions on Games*, *10*(1), 56–68.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994* (pp. 157–163). Elsevier.

Liu, M., Sivakumar, K., Omidshafiei, S., Amato, C., & How, J. P. (2017). Learning for multi-robot cooperation in partially observable stochastic environments with macro-actions. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1853–1860.

Liu, M., Zhou, M., Zhang, W., Zhuang, Y., Wang, J., Liu, W., & Yu, Y. (2020). Multi-agent interactions modeling with correlated policies. *arXiv preprint arXiv:2001.03415*, *None*(None), None.

Maddison, C. J., Mnih, A., & Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Mahadevan, S., Marchalleck, N., Das, T. K., & Gosavi, A. (1997). Self-improving factory simulation using continuous-time average-reward reinforcement learning. *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, 202–210.

Mai, T., Nguyen, Q. P., Low, K. H., & Jaillet, P. (2019). Inverse reinforcement learning with missing data. *arXiv preprint arXiv:1911.06930*.

Mao, A., Mohri, M., & Zhong, Y. (2023). Cross-entropy loss functions: Theoretical analysis and applications. *International Conference on Machine Learning*, 23803–23828.

Marinescu, R., Dechter, R., & Ihler, A. T. (2014). And/or search for marginal map. *UAI*, 563–572.

McGovern, A., Precup, D., Ravindran, B., Singh, S., & Sutton, R. S. (1998). Hierarchical optimal control of mdps. *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems*, 186–191.

McGugan, W. (2007). *Beginning game development with python and pygame: From novice to professional*. Apress.

Melo, F. S., & Veloso, M. (2011). Decentralized mdps with sparse interactions. *Artificial Intelligence*, *175*(11), 1757–1789.

Menéndez, M., Pardo, J., Pardo, L., & Pardo, M. (1997). The jensen-shannon divergence. *Journal of the Franklin Institute*, *334*(2), 307–318.

Michini, B., & How, J. P. (2012). Bayesian nonparametric inverse reinforcement learning. *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part II 23*, 148–163.

Mirsky, R., Carlucho, I., Rahman, A., Fosong, E., Macke, W., Sridharan, M., Stone, P., & Albrecht, S. V. (2022). A survey of ad hoc teamwork: Definitions, methods, and open problems. *European Conference on Multiagent Systems*.

Mittal, M., Yu, C., Yu, Q., Liu, J., Rudin, N., Hoeller, D., Yuan, J. L., Singh, R., Guo, Y., Mazhar, H., Mandlekar, A., Babich, B., State, G., Hutter, M., & Garg, A. (2023). Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, *8*(6), 3740–3747. https://doi.org/10.1109/LRA.2023.3270034

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Monahan, G. E. (1982). State of the art—a survey of partially observable markov decision processes: Theory, models, and algorithms. *Management science*, *28*(1), 1–16.

Mukherjee, D., Gupta, K., Chang, L. H., & Najjaran, H. (2022). A survey of robot learning strategies for human-robot collaboration in industrial settings. *Robotics and Computer-Integrated Manufacturing*, *73*, 102231.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. *Icml*, *1*, 2.

Nikolaidis, S., Forlizzi, J., Hsu, D., Shah, J., & Srinivasa, S. (2017). Mathematical models of adaptation in human-robot collaboration. *arXiv preprint arXiv:1707.02586*.

Nikolaidis, S., Kwon, M., Forlizzi, J., & Srinivasa, S. (2018). Planning with verbal communication for human-robot collaboration. *ACM Transactions on Human-Robot Interaction (THRI)*, *7*(3), 1–21.

Nikolaidis, S., Lasota, P., Ramakrishnan, R., & Shah, J. (2015). Improved human–robot team performance through cross-training, an approach inspired by human team training practices. *The International Journal of Robotics Research*, *34*(14), 1711–1730.

Nikolaidis, S., Lasota, P., Rossano, G., Martinez, C., Fuhlbrigge, T., & Shah, J. (2013). Human-robot collaboration in manufacturing: Quantitative evaluation of predictable, convergent joint action. *IEEE ISR 2013*, 1–6.

Nikolaidis, S., Nath, S., Procaccia, A. D., & Srinivasa, S. (2017). Game-theoretic modeling of human adaptation in human-robot collaboration. *Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction*, 323–331.

Nikolaidis, S., Ramakrishnan, R., Gu, K., & Shah, J. (2015). Efficient model learning from joint-action demonstrations for human-robot collaborative tasks. *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 189–196.

Nikolaidis, S., & Shah, J. (2012). Human-robot teaming using shared mental models. *ACM/IEEE HRI*.

Nikolaidis, S., Zhu, Y. X., Hsu, D., & Srinivasa, S. (2017). Human-robot mutual adaptation in shared autonomy. *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 294–302.

Nikolaidis, S. Z. (2014). *Computational formulation, modeling and evaluation of human-robot team training techniques* [Doctoral dissertation, Massachusetts Institute of Technology].

Ong, S. C., Png, S. W., Hsu, D., & Lee, W. S. (2010). Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, *29*(8), 1053–1068.

Orlov Savko, L., Qian, Z., Gremillion, G., Neubauer, C., Canady, J., Unhelkar, V., & Neubauer, C. (2024). Rw4t dataset: Data of human-robot behavior and cognitive states in simulated disaster

response tasks. *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, 924–928.

Orlov-Savko, L., Jain, A., Gremillion, G. M., Neubauer, C. E., Canady, J. D., & Unhelkar, V. (2022). Factorial agent markov model: Modeling other agents' behavior in presence of dynamic latent decision factors. *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 982–990.

Orsini, M., Raichuk, A., Hussenot, L., Vincent, D., Dadashi, R., Girgin, S., Geist, M., Bachem, O., Pietquin, O., & Andrychowicz, M. (2021). What matters for adversarial imitation learning? *Advances in Neural Information Processing Systems*, *34*, 14656–14668.

Panella, A., & Gmytrasiewicz, P. (2016). Bayesian learning of other agents' finite controllers for interactive pomdps. *Proceedings of the AAAI Conference on Artificial Intelligence*, *30*(1).

Peternel, L., Tsagarakis, N., Caldwell, D., & Ajoudani, A. (2018). Robot adaptation to human physical fatigue in human–robot co-manipulation. *Autonomous Robots*, *42*, 1011–1021.

Poggio, T., Torre, V., & Koch, C. (1987). Computational vision and regularization theory. *Readings in computer vision*, 638–643.

Prasad, V., Kshirsagar, A., Stock-Homburg, D. K. R., Peters, J., & Chalvatzaki, G. (2024). Moveint: Mixture of variational experts for learning human-robot interactions from demonstrations. *IEEE Robotics and Automation Letters*.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming* (1st). John Wiley & Sons, Inc.

Qian, Z., Orlov Savko, L., Neubauer, C., Gremillion, G., & Unhelkar, V. (2024). Measuring variations in workload during human-robot collaboration through automated after-action reviews. *Companion of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, 852–856.

Raffin, A., Kober, J., & Stulp, F. (2022). Smooth exploration for robotic reinforcement learning. *Conference on Robot Learning*, 1634–1644.

Rahman, A., Carlucho, I., Höpner, N., & Albrecht, S. V. (2023). A general learning framework for open ad hoc teamwork using graph-based policy learning. *Journal of Machine Learning Research*, *24*(298), 1–74.

Ramachandran, D., & Amir, E. (2007a). Bayesian inverse reinforcement learning. *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, 2586–2591.

Ramachandran, D., & Amir, E. (2007b). Bayesian inverse reinforcement learning. *IJCAI*, *7*, 2586–2591.

Reddy, T. S., Gopikrishna, V., Zaruba, G., & Huber, M. (2012). Inverse reinforcement learning for decentralized non-cooperative multiagent systems. *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 1930–1935.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.

Reis, A. (n.d.). *Rl blog*.

Repast framework. (2023).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Russell, S. (1998). Learning agents for uncertain environments (extended abstract). *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, 101–103.

Sadigh, D., Sastry, S. S., Seshia, S. A., & Dragan, A. (2016). Information gathering actions over human internal state. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 66–73.

Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., & Whiteson, S. (2019). The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*.

Schulman, J., Heess, N., Weber, T., & Abbeel, P. (2015). Gradient estimation using stochastic computation graphs. *Advances in neural information processing systems*, *28*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. *International conference on machine learning*, 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, *None*(None), None.

Sengadu Suresh, P., Gui, Y., & Doshi, P. (2023). Dec-airl: Decentralized adversarial irl for human-robot teaming. *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 1116–1124.

Seo, S., & Unhelkar, V. (2024). Idil: Imitation learning of intent-driven expert behavior. *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 1673–1682.

Seo, S., & Unhelkar, V. V. (2022). Semi-supervised imitation learning of team policies from suboptimal demonstrations. *arXiv preprint arXiv:2205.02959*.

Settles, B. (2009). Active learning literature survey. *Machine Learning*.

Shadow Robot Company [Accessed: February 19, 2024]. (n.d.).

Shahryari, S., & Doshi, P. (2017). Inverse reinforcement learning under noisy observations. *arXiv preprint arXiv:1710.10116*.

Shiarlis, K., Messias, J., & Whiteson, S. (2016). Inverse reinforcement learning from failure.

Shlens, J. (2014). Notes on kullback-leibler divergence and likelihood. *arXiv preprint arXiv:1404.2000*.

Shoham, Y., & Leyton-Brown, K. (2008). *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

Silver, D. (n.d.). *Rl lectures*.

Simão, J. S., & Demazeau, Y. (2001). On social reasoning in multi-agent systems. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, *5*(13), 1–10.

Simon, H. A. (1990). Bounded rationality. *Utility and probability*, 15–18.

Soans, N., Asali, E., Hong, Y., & Doshi, P. (2020). Sa-net: Robust state-action recognition for learning from observations. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2153–2159.

Song, J., Ren, H., Sadigh, D., & Ermon, S. (2018). Multi-agent generative adversarial imitation learning. *Advances in neural information processing systems*, *31*, None.

Stone, P., Kaminka, G., Kraus, S., & Rosenschein, J. (2010). Ad hoc autonomous agent teams: Collaboration without pre-coordination. *Proceedings of the AAAI Conference on Artificial Intelligence*, *24*(1), 1504–1509.

Sugiyama, M., & Kawanabe, M. (2012). *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press.

Suresh, P., Romeres, D., Doshi, P., & Jain, S. (n.d.). Open human-robot collaboration systems (ohrcs): A research perspective.

Suresh, P. S., & Doshi, P. (2022). Marginal map estimation for inverse rl under occlusion with observer noise. *The 38th Conference on Uncertainty in Artificial Intelligence*, None.

Suresh, P. S., Jain, S., Doshi, P., & Romeres, D. (2024). Open human-robot collaboration using decentralized inverse reinforcement learning. *arXiv preprint arXiv:2410.01790*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1-2), 181–211.

Tao, S., Xiang, F., Shukla, A., Qin, Y., Hinrichsen, X., Yuan, X., Bao, C., Lin, X., Liu, Y., Chan, T.-k., Gao, Y., Li, X., Mu, T., Xiao, N., Gurha, A., Huang, Z., Calandra, R., Chen, R., Luo, S., & Su, H. (2024). Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*.

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ international conference on intelligent robots and systems*, 5026–5033.

Torabi, F., Warnell, G., & Stone, P. (2018). Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*.

Trivedi, M., & Doshi, P. (2018). Inverse learning of robot behavior for collaborative planning. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1–9.

Unhelkar, V. V., Li, S., & Shah, J. A. (2020a). Decision-making for bidirectional communication in sequential human-robot collaborative tasks. *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, 329–341.

Unhelkar, V. V., Li, S., & Shah, J. A. (2020b). Semi-supervised learning of decision-making models for human-robot collaboration. *Conference on Robot Learning*, 192–203.

Unhelkar, V. V. (2020). *Effective information sharing for human-robot collaboration* [Doctoral dissertation, Massachusetts Institute of Technology].

van der Spaa, L., Kober, J., & Gienger, M. (2024). Simultaneously learning intentions and preferences during physical human-robot cooperation. *Autonomous Robots*, *48*(4), 11.

Vaserstein, L. N. (1969). Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, *5*(3), 64–72.

Venuto, D. (2020). *Robust adversarial inverse reinforcement learning with temporally extended actions*. McGill University (Canada).

Vicentini, F. (2021). Collaborative robotics: A survey. *Journal of Mechanical Design*, *143*(4), 040802.

Villani, V., Pini, F., Leali, F., & Secchi, C. (2018). Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, *55*, 248–266.

Visioli, A. (2006). *Practical pid control*. Springer Science & Business Media.

Vroman, M. C. (2014). *Maximum likelihood inverse reinforcement learning* [Doctoral dissertation, Rutgers University-Graduate School-New Brunswick].

Wang, C., Pérez-D'Arpino, C., Xu, D., Fei-Fei, L., Liu, K., & Savarese, S. (2022). Co-gail: Learning diverse strategies for human-robot collaboration. *Conference on Robot Learning*, 1279–1290.

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2023). Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7464–7475.

Wei, E., Wicke, D., & Luke, S. (2019). Multiagent adversarial inverse reinforcement learning. *AAMAS*, 2265–2266.

Whitney, D., Rosen, E., MacGlashan, J., Wong, L. L., & Tellex, S. (2017). Reducing errors in object-fetching interactions through social feedback. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 1006–1013.

Wikipedia. (2024a). *Comparison of key reinforcement learning algorithms*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Reinforcement_learning#Comparison_of_key_algorithms

Wikipedia. (2024b). *Imporance sampling*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Importance_sampling

Wikipedia. (2024c). *Regularization in machine learning*. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Regularization_(mathematics)

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*, 229–256.

Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. *2017 IEEE international conference on image processing (ICIP)*, 3645–3649.

Wulfmeier, M., & Posner, I. (2015). Maximum Entropy Deep Inverse Reinforcement Learning. *arXiv preprint*.

Xuan, P., Lesser, V., & Zilberstein, S. (2000). S. *Proceedings Fourth International Conference on MultiAgent Systems*, 467–468.

Yang, F., Vereshchaka, A., Chen, C., & Dong, W. (2020). Bayesian multi-type mean field multi-agent imitation learning. *Advances in Neural Information Processing Systems*, *33*, 2469–2478.

Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., & Wu, Y. (2022). The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, *35*, 24611–24624.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., & Wu, Y. (2021). The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, *1*(None), None.

Yu, L., Song, J., & Ermon, S. (2019). Multi-agent adversarial inverse reinforcement learning. *International Conference on Machine Learning*, 7194–7201.

Yuan, L., Wang, J., Zhang, F., Wang, C., Zhang, Z., Yu, Y., & Zhang, C. (2022). Multi-agent incentive communication via decentralized teammate modeling. *Proceedings of the AAAI Conference on Artificial Intelligence*, *36*, 9466–9474.

Zellner, A., & Highfield, R. A. (1988). Calculation of maximum entropy distributions and approximation of marginalposterior distributions. *Journal of Econometrics*, *37*(2), 195–209.

Zhang, Z., Yuan, L., Li, L., Xue, K., Jia, C., Guan, C., Qian, C., & Yu, Y. (2023). Fast teammate adaptation in the presence of sudden policy change. *arXiv preprint arXiv:2305.05911*.

Zheng, J., Liu, S., & Ni, L. M. (2014). Robust bayesian inverse reinforcement learning with sparse behavior noise. *Proceedings of the AAAI Conference on Artificial Intelligence*, *28*.

Ziebart, B. (2010, December). *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy* [Doctoral dissertation, Carnegie Mellon University].

Ziebart, B. D., Bagnell, J. A., & Dey, A. K. (2010). Modeling interaction via the principle of maximum causal entropy. *ICML*.

Ziebart, B. D., Maas, A., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, 1433–1438.