Building Assistive Sensorimotor Interfaces through Human-in-the-Loop Machine Learning

by

Siddharth Reddy

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Anca Dragan, Co-chair
Associate Professor Sergey Levine, Co-chair
Assistant Professor Gopala Krishna Anumanchipalli
Dr. Jan Leike

Spring 2022

Building Assistive Sensorimotor Interfaces through Human-in-the-Loop Machine Learning

Abstract

Building Assistive Sensorimotor Interfaces through Human-in-the-Loop Machine Learning

by

Siddharth Reddy

Doctor of Philosophy in Computer Science

University of California, Berkeley

Associate Professor Anca Dragan, Co-chair

Associate Professor Sergey Levine, Co-chair

One of the outstanding challenges in the field of human-computer interaction is building assistive interfaces that help users with the perception and control of complex systems, such as cars, quadcopters, and prosthetic limbs. In this thesis, we propose machine learning algorithms for automatically designing personalized, adaptive interfaces that improve users' performance on sequential decision-making tasks. First, we present work that uses theory of mind to model irrational user behavior as rational with respect to incorrect internal beliefs about how the world works, and leverages this assumption to assist users by modifying their observations and actions. Second, we present work that uses model-free reinforcement learning from human feedback to fine-tune user actions, with minimal assumptions about user behavior. We demonstrate the effectiveness of our methods through experiments with human participants, in which users play the Lunar Lander video game, perform simulated navigation tasks, and land a quadcopter.

To my family

Contents

List of Figures

List of Tables

Acknowledgments

Many people supported me along my Ph.D. journey.

Thanks to my parents, Madhavi and Sundeep Reddy, for raising me with love and encouraging me to pursue my dream of becoming a computer scientist.

Thanks to my Ph.D. advisors, Anca Dragan and Sergey Levine, for teaching me how to be a researcher and steering me in productive directions. I still remember our first meeting in March 2017 during Berkeley EECS Visit Days. I was new to robotics and reinforcement learning, but when you pitched the shared autonomy project, it clicked, and I was 100% certain that I had to come to Berkeley to work with you.

Thanks to my collaborators, Jensen Gao, Sean Chen, Glen Berseth, Nicholas Hardy, Nikhilesh Natraj, Karunesh Ganguly, Shane Legg, Jan Leike, and Gokul Swamy. It has been a blast working with you on algorithms for brain-computer interfaces, AI alignment, and robotic teleoperation.

Thanks to everyone in InterACT, RAIL, and BAIR, for being awesome labmates and creating an inspiring scenius at Berkeley.

Thanks to my undergraduate research advisors at Cornell—Thorsten Joachims, Igor Labutov, and Siddhartha Banerjee—and my summer internship mentors at Knewton—Chaitanya Ekanadham and Kevin Wilson—for introducing me to the world of computer science research.

Thanks to Preethi Gunaratne, David Wheeler, and Townley Chisholm, for mentoring me through my very first research experiences in high school and encouraging me to become a scientist.

Thanks to Jonathon Cai, for being a great roommate and an exemplar of independent, critical thinking.

Thanks to Curran Reddy and the boys, for our weekly excursions in Elysium during the pandemic.

Thanks to Zachary Marshall Young, Akilesh Potti, Vipin Sharma, Amal Puri, and Alok Singh, for our friendship and adventures together. Bless the Maker and His Water.

This work was supported by a Berkeley EECS Department Fellowship for first-year Ph.D. students, NVIDIA Graduate Fellowship, NSF IIS-1700696, AFOSR FA9550-17-1-0308, NSF NRI 1734633, Berkeley DeepDrive, and computational resource donations from Amazon.

Chapter 1

Introduction

I can calculate the motion of heavenly bodies, but not the madness of people.

—Sir Isaac Newton

The idea of combining human and machine intelligence in a shared-control system goes back to the early days of computing, with Ray Goertz's remote manipulator in 1949 [37], Ralph Mosher's Hardiman exoskeleton in 1968 [76], and Marvin Minsky's call for telepresence in 1980 [73]. After decades of research in robotics, human-computer interaction, and artificial intelligence, interfacing between a human operator and a remote-controlled robot remains a challenge. According to a review of the 2015 DARPA Robotics Challenge [6], "the most cost effective research area to improve robot performance is Human-Robot Interaction....The biggest enemy of robot stability and performance in the DRC was operator errors. Developing ways to avoid and survive operator errors is crucial for real-world robotics. Human operators make mistakes under pressure, especially without extensive training and practice in realistic conditions."

Designing interfaces that enable humans to seamlessly control machines can be a hard problem, especially for emerging applications like teleoperating robots through virtual reality headsets, sensing the world through augmented reality glasses, controlling prosthetic limbs through brain implants, driving trucks remotely, and performing robotic surgery. The challenge for the user is that their behavior can be suboptimal due to cognitive biases, limited reaction time, and other human factors. The challenge for the machine is that the user may not be able to directly specify the desired task in a machine-readable format, so the machine must infer the user's intent from the user's behavior. In this setting, *can we train machines to automatically assist users who exhibit suboptimal behavior?*

To answer this question, consider the model of human decision-making in Figure 1.1. First, the user observes an image from the quadcopter's camera. Then, given the observations they have received, and their internal beliefs about how those observations were generated by the world, they infer the current position and orientation of the quadcopter as well as the

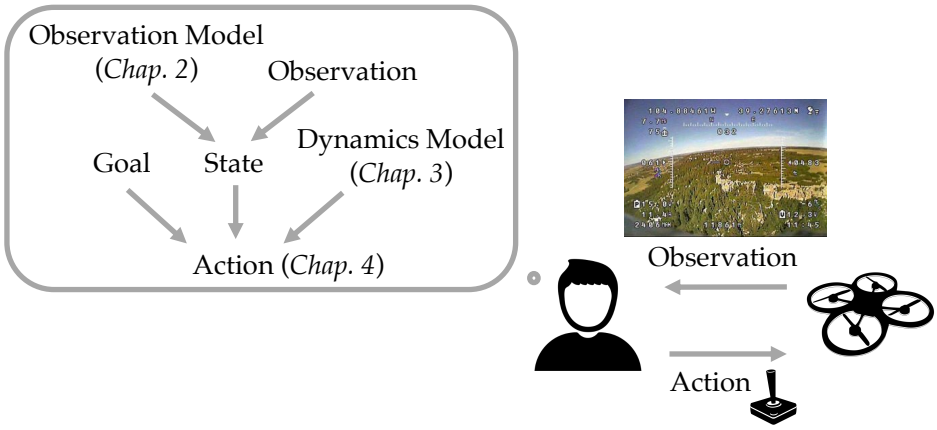

Figure 1.1: Consider a human operator that wants to locate a herd of animals using a joystick-controlled quadcopter equipped with a camera. We assume that when the user observes an image from the camera, they update their internal estimate of the state of the world (e.g., the position and orientation of the quadcopter, and the number of animals present in the area), then take an action that aims to achieve their goal (e.g., pushing the joystick forward with the intent of surveying more of the area in search of the herd). When the user updates their beliefs about the current state, we assume that they rely on an internal observation model that evaluates the probability of observing a given image from a hypothetical state. When the user plans their next action, we assume that they do so using an internal dynamics model that predicts the state transition caused by a hypothetical action. An inaccurate observation model or inaccurate dynamics model can lead to suboptimal actions.

number of animals in the area. Then, given their goal of locating the herd, and their internal beliefs about the physics of the quadcopter, they decide how to move their joystick.

Why is this decision sometimes suboptimal? Prior work assumes that the user's actions can be noisy, but are generally close to optimal (e.g., see Section 3.2). Based on this assumption, we could infer the user's desired goal from their actions, then automatically finish going to the goal for them. The problem with this assumption is that it doesn't take into account cognitive biases that cause the user's actions to deviate systematically from the optimal actions. When there are systematic biases that violate the assumption of noisy, near-optimality, we can no longer correctly infer the goal and no longer effectively assist in reaching the user's true goal [20]. Furthermore, since cognitive science researchers are still in the process of cataloguing these biases [22], specifying a unified model of user behavior that captures all possible biases is infeasible.

Our insight is that, instead of assuming that the user's behavior follows a known model of near-optimality or systematic suboptimality with respect to the real world, we assume that the user's behavior is near-optimal *with respect to the user's internal beliefs about how the world works*, and infer these internal beliefs from recordings of the user's behavior in the real world. If these internal beliefs are inaccurate, then behavior that is near-optimal with

respect to these beliefs can be systematically suboptimal in the real world.

Where can inaccurate beliefs about how the world works enter the user's decision-making process? The first place is perception (Chapter 2). In particular, the user's observation model about how the world generates observations may be inaccurate (see Figure 1.1). This can cause the user to inaccurately estimate the current state of the world, and, as a result, take the wrong action. The second place where bias can enter is planning (Chapter 3). For example, the user's internal model of the state transition dynamics may be inaccurate (see Figure 1.1), causing the user to take the wrong action.

We formalize this model of user behavior as

$$p(\mathbf{a}_t|\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}; \theta, \phi) = \int_{\mathcal{S}} \frac{\exp\left(Q(\mathbf{s}_t, \mathbf{a}_t)\right)}{\sum_{\mathbf{a}\in\mathcal{A}} \exp\left(Q(\mathbf{s}_t, \mathbf{a})\right)} b_\theta(\mathbf{s}_t|\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}) d\mathbf{s}_t, \tag{1.1}$$

where b_θ is the user's internal state estimation procedure, $\mathbf{a}$ is the user's action, $\mathbf{o}$ is the user's observation, $\mathbf{s}$ is the environment state, and Q is the user's soft Q function. We assume that Q satisfies the soft Bellman equation [119],

$$Q(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1}\sim T_\phi(\cdot|\mathbf{s}_t, \mathbf{a}_t)} \left[R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) + \gamma \log \left(\sum_{\mathbf{a}_{t+1}\in\mathcal{A}} \exp\left(Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right) \right) \right], \tag{1.2}$$

where T_ϕ is the user's internal dynamics model, and R is the user's reward function. The key idea in this model is that T_ϕ may differ from the true state transition dynamics, and b_θ may differ from the optimal Bayes filter for state estimation. See Equations 2.4 and 3.1 for details.

Chapters 2 and 3 discuss the systematic biases in perception and planning that users can have when they make decisions, how to learn about those biases from demonstrations of suboptimal user behavior, and how to leverage the learned internal models (b_θ and T_ϕ) to assist the user in overcoming bias.

What if we don't know, or don't want to make assumptions about, the kinds of biases the user may have? Chapter 4 addresses this question by using model-free reinforcement learning from human feedback to fine-tune user actions, with minimal assumptions about user behavior.

Chapter 2

Optimizing Observations to Communicate State

Chew, if only you could see what I've seen with your eyes.

—Roy Batty, *Blade Runner (1982)*

In this chapter, we aim to help users estimate the state of the world in tasks like robotic teleoperation and navigation with visual impairments, where users may have systematic biases that lead to suboptimal behavior: they might struggle to process observations from multiple sensors simultaneously, receive delayed observations, or overestimate distances to obstacles. While we cannot directly change the user's internal beliefs or their internal state estimation process, our insight is that we can still assist them by modifying the user's observations. Instead of showing the user their true observations, we synthesize new observations that lead to more accurate internal state estimates when processed by the user.

We refer to this method as *assistive state estimation* (ASE): an automated assistant uses the true observations to infer the state of the world, then generates a modified observation for the user to consume (e.g., through an augmented reality interface), and optimizes the modification to induce the user's new beliefs to match the assistant's current beliefs. To predict the effect of the modified observation on the user's beliefs, ASE learns a model of the user's state estimation process: after each task completion, it searches for a model that would have led to beliefs that explain the user's actions.

We evaluate ASE in a user study with 12 participants who each perform four tasks: two tasks with known user biases (bandwidth-limited image classification and a driving video game with observation delay), and two with unknown biases that our method has to learn (guided 2D navigation and a lunar lander teleoperation video game). ASE's general-purpose approach to synthesizing informative observations enables a different assistance strategy to emerge in each domain, such as quickly revealing informative pixels to speed up image

classification, using a dynamics model to undo observation delay in driving, identifying nearby landmarks for navigation, and exaggerating a visual indicator of tilt in the lander game. The results show that ASE substantially improves the task performance of users with bandwidth constraints, observation delay, and other unknown biases.

2.1 Introduction

People cannot directly access the state of the world, and must instead estimate it from sensory observations [61]. Unfortunately, systematic biases in the user's state estimation process can lead to inaccurate beliefs and suboptimal actions. For example, the user may not be able to keep track of many different sensors simultaneously while flying a plane [78], or navigate with a visual impairment while listening to a smartphone guide exhaustively list all nearby objects [83]. Tasks performed over a network, like teleoperating space robots [33], may require the user to compensate for unintuitive, intermittent delays in observations. Lens distortions can cause drivers to overestimate distances to obstacles: the warning, "objects in mirror may be closer than they appear," is engraved into the side mirrors of cars.

Short of intervening in human cognition through brain stimulation, or training users to overcome their biases, how can we assist users with performing more accurate perception? The key idea in this paper is that an automated assistant can intervene in human perception by modifying the observations the user receives. Given the user's biases, different observations lead to different state estimates. We invert this process to 'trick' the person into arriving at the correct estimate: we modify observations so that, when processed by the biased user, they induce an accurate state estimate.

Figure 2.1 describes our method: the assistant collects observations from the environment, performs state estimation unencumbered by cognitive biases, then shows the user a synthetic, optimized observation that induces accurate beliefs when processed by their biased perception system. These synthetic observations could be constrained to augment real observations—for example, in an augmented reality interface [118]—or could completely replace them—for instance, by replacing the user's video feed for teleoperating a robot. Crucially, this approach does not require knowing the current task the user is performing: rather than inducing the optimal action for the task, we induce accurate state estimates, so that the user can then decide on task-appropriate actions.

The main challenge is that in order to determine how informative a synthetic observation will be to the user, we need a model of the user's state estimation process. For instance, we might not know the user's bandwidth constraint, observation delay, or even what kinds of biases they have. We introduce an approach for training this model online: we start assisting with an initial model, and collect data of the user suboptimally performing tasks (e.g., navigating to different goals). We assume that upon task completion, the assistant gets to know what the task was (e.g., which goal the user was trying to reach) and can compute a near-optimal policy for that task (e.g., using reinforcement learning). We then look in hindsight at the user's actions, and optimize model parameters that lead to state

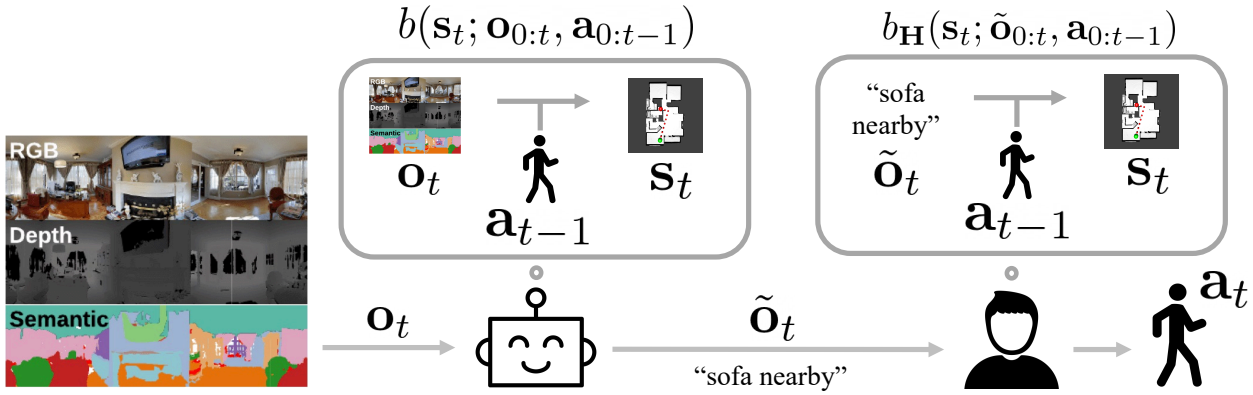

Figure 2.1: The assistant processes observations $\mathbf{o}_t$ generated by the environment on behalf of the user $\mathbf{H}$, updates its belief distribution over the current state $b(\mathbf{s}_t; \mathbf{o}_{0:t}, \mathbf{a}_{0:t-1})$, then synthesizes an observation $\tilde{\mathbf{o}}_t$ that will induce accurate beliefs $b_{\mathbf{H}}(\mathbf{s}_t; \tilde{\mathbf{o}}_{0:t-1}, \tilde{\mathbf{o}}_t, \mathbf{a}_{0:t-1})$ when shown to the user, enabling the user to make better decisions $\mathbf{a}_t$. For example, the assistant may use a smartphone camera and speaker to guide a visually-impaired user through an indoor environment: the assistant observes the user's egocentric scene through the camera, uses an object detector to determine nearby objects, then tells the user about one of them through the speaker. If the user's mental map of the environment includes object locations, the user can then infer their position and orientation: they must be in one of the states from which the mentioned object is visible. Enumerating all visible objects may overwhelm the user, so we assume the user is 'bandwidth-constrained' to hearing about just one object at a time. Hence, the assistant's challenge is to select the single object that will be most informative to the user (e.g., a landmark that is only visible from the user's current state).

estimates which make the observed actions seem near-optimal. Intuitively, we ask what the user must have believed, and what state estimation process would have led to those beliefs given the observations they received. The experiments in Appendix 6.1 show that the quality of assistance improves as we collect more data and the maximum-likelihood model gets closer to the user's internal state estimation process.

Our primary contribution is the *assistive state estimation* (ASE) algorithm for optimizing synthetic observations to induce accurate beliefs about the current state in the user. We evaluate ASE through a user study with 12 participants who each perform four tasks: two where the user's bias is known—image classification from bandwidth-limited input, and a driving video game with observation delay—and two where the bias is unknown—navigating a 2D simulation with limited vision where the user only remembers the locations of certain objects but not others, and a lunar lander teleoperation video game. The lander experiment is particularly interesting: the assistant learns to modify the tilt indicator away from its real value; actually improving the user's task performance, possibly because users tend to underestimate tilt. Our user studies show that in all domains, ASE substantially improves the user's task performance, relative to a passive baseline that simply shows the user an ambient observation generated by the environment. In addition to the user study, we perform

experiments with simulated users that show ASE improves the accuracy of simulated users' internal beliefs, and that ASE is capable of learning an expressive, neural network model of the user's belief update given enough demonstration data.

2.2 Assisting Users by Optimizing Observations

We formulate the assistance problem as follows. We assume that the environment follows a partially observable Markov decision process (POMDP) [56] with state space $\mathcal{S}$, observation space Ω, initial state distribution $p^{\text{init}}(\mathbf{s}_0)$, state transition dynamics $p^{\text{dyn}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, observation model $p^{\text{obs}}(\mathbf{o}|\mathbf{s})$, and unknown reward function $R(\mathbf{s}, \mathbf{a})$. At each timestep t, the assistant samples an ambient observation $\mathbf{o}_t$ from the environment. The assistant then intervenes and provides the user with a different observation $\tilde{\mathbf{o}}_t \in \Omega$. Since the reward function is unknown, we cannot compute the optimal action and provide the user with an observation that will induce them to take the optimal action. Instead, we aim for a task-agnostic method that assists the user by providing them with an observation that efficiently communicates the current state.

Our approach to this problem is outlined in Figure 2.1. We assume that the user's state estimation process differs from the assistant's, and that this mismatch leads to suboptimal user behavior. We assist the user by showing them synthetic observations that induce accurate beliefs about the current state. In particular, the assistant first performs state estimation, then optimizes an observation to update the user's beliefs to match the assistant's beliefs. To improve the assistant, we learn a personalized model of the user's state estimation process from demonstrations of suboptimal user actions on known tasks.

Preliminaries: Assumptions about State Estimation

The standard, recursive Bayesian filter [106] performs state estimation using the belief update,

$$b(\mathbf{s}_t|\mathbf{o}_{0:t}, a_{0:t-1}) \propto p^{\text{obs}}(\mathbf{o}_t|\mathbf{s}_t) \int_{\mathcal{S}} p^{\text{dyn}}(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) b(\mathbf{s}_{t-1}|\mathbf{o}_{0:t-1}, \mathbf{a}_{0:t-2}) d\mathbf{s}_{t-1}. \tag{2.1}$$

In domains with a small, discrete state space $\mathcal{S}$, we compute exact belief updates using Equation 2.1. In domains with high-dimensional, continuous states, the belief update in Equation 2.1 may be intractable to compute. To address this issue, we represent the state estimation process in continuous domains as

$$b(\mathbf{s}_t|\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}) = \mathcal{N}(\mathbf{s}_t; \mu = f(\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}), \Sigma = I\sigma^2), \tag{2.2}$$

where f is a known state encoder that maps a sequence of observations and actions to a continuous, vector-valued state. Although this procedure does not necessarily perform Bayesian belief updates, it enables us to apply our method to domains where the true initial state distribution p^{init}, true dynamics model p^{dyn}, and true observation model p^{obs} are unknown, but a state encoder f is available.

Synthesizing Observations that Induce Accurate Beliefs

To assist the user, we synthesize an observation $\tilde{\mathbf{o}}_t$ such that, after the user observes $\tilde{\mathbf{o}}_t$ and updates their beliefs about the current state $\mathbf{s}_t$, the user's beliefs will match the assistant's beliefs. Formally, given a history $(\mathbf{o}_{0:t}, \tilde{\mathbf{o}}_{0:t-1}, \mathbf{a}_{0:t-1})$, the assistant decides which observation to provide to the user $\mathbf{H}$ by greedily minimizing the KL-divergence between the assistant's beliefs and the user's beliefs at the end of the current timestep:

$$\tilde{\mathbf{o}}_t \leftarrow \arg\min_{\tilde{\mathbf{o}}_t \in \Omega} D_{\mathrm{KL}}\Big(\underbrace{b(\mathbf{s}_t | \mathbf{o}_{0:t}, \mathbf{a}_{0:t-1})}_{\text{assistant's beliefs}} \,\Big\|\, \underbrace{\hat{b}_{\mathbf{H}}(\mathbf{s}_t | \tilde{\mathbf{o}}_{0:t-1}, \tilde{\mathbf{o}}_t, \mathbf{a}_{0:t-1})}_{\text{assistant's prediction of user's beliefs}}\Big), \tag{2.3}$$

where $\hat{b}_{\mathbf{H}}$ is the assistant's model of the user's state estimation process. The assistant's beliefs are fixed during this optimization, having already been conditioned on the most recent ambient observation $\mathbf{o}_t$ generated by the environment, while the user's beliefs are conditioned on the synthetic observation $\tilde{\mathbf{o}}_t$ and can thus be optimized. The experiments in Section 2.4 and Appendix 6.1 illustrate how different assistance strategies emerge from Equation 2.3, such as revealing informative pixels for image classification, undoing observation delay in driving by forward-predicting the current observation, identifying landmarks for navigation, and exaggerating indicators of dangerous states in a landing task.

Learning Personalized Models of State Estimation

To optimize the synthetic observation in Equation 2.3, we need to model how the user will update their beliefs in response to observations. We assume the user's unknown state estimation process $b_{\mathbf{H}}$ differs from the assistant's known process b described in Equations 2.1 and 2.2. In particular, we assume $b_{\mathbf{H}}$ lies in hypothesis space $\mathcal{B}$. The hypothesis space, which we parameterize as $\mathcal{B} = \{b_\theta : \theta \in \Theta\}$, captures our prior assumptions about possible user biases. If we want to make minimal assumptions about the user's biases, we could define θ to be the weights in a neural network state encoder f_θ that defines the belief update b_θ via Equation 2.2. If instead we assume that the user performs a Bayesian belief update on each new observation and action, but potentially ignores or misinterprets certain observations, we could define θ to be the observation probabilities $p_\theta^{\mathrm{obs}}(\mathbf{o}|\mathbf{s}) = \theta_{\mathbf{o},\mathbf{s}}$ in Equation 2.1. In each of our experiments, we make different assumptions about the user, leading to different choices of hypothesis space $\mathcal{B}$.

We search the hypothesis space for a model that best explains user behavior. We assume access to a dataset $\mathcal{D}$ of demonstrations of suboptimal user actions on known tasks. This dataset could be generated offline by the user without the assistant's help, or generated online while the assistant helps the user. After each demonstration episode, we ask the user what task they were trying to perform during that episode. The task could be specified, for example, through a goal state or a reward function. Let $\tau = (\mathbf{o}_{0:T-1}, \mathbf{a}_{0:T-1})$ denote a demonstration, where T is the episode length. We model the user's actions as rational with

respect to their beliefs about the current state:

$$p(\mathbf{a}_t|\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}; \theta) = \int_{\mathcal{S}} \pi(\mathbf{a}_t|\mathbf{s}_t) b_\theta(\mathbf{s}_t|\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}) d\mathbf{s}_t, \tag{2.4}$$

where π is the user's policy, which we assume to be near-optimal for their desired task. We compute π in hindsight after asking the user what task they were trying to demonstrate; e.g., by asking the user to write down the reward function, then doing maximum entropy reinforcement learning [67]. Note that we only need to know a near-optimal policy for the tasks in the demonstrations used to train the user model. We do not need to know the policy at test time when we synthesize observations to assist the user. We assume that the user's policy π for a given task and their belief update $b_{\mathbf{H}}$ do not change once the assistant begins modifying observations using Equation 2.3. In practice, even if the user adapts their policy or state estimation process to the assistant, this tends to improve performance, rather than hurt it.

We use gradient descent to compute the maximum-likelihood estimate,

$$\hat{\theta} \leftarrow \arg\max_{\theta} \sum_{\tau \in \mathcal{D}} \sum_{t} \log p(\mathbf{a}_t|\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}; \theta). \tag{2.5}$$

We select the maximum-likelihood model to be our model of the user's state estimation process: $\hat{b}_{\mathbf{H}} \leftarrow b_{\hat{\theta}}$. This model enables us to predict the effect of an observation on the user's beliefs.

Assistive State Estimation

Our assistive state estimation (ASE) method is summarized in Algorithm 1. We initialize the user model $\hat{b}_{\mathbf{H}}$ with an initial model b_{init}. In domains with a small, discrete state space $\mathcal{S}$, we assume knowledge of the initial state distribution, state transition dynamics, and observation model, in order to compute Bayesian belief updates using Equation 2.1. In domains with high-dimensional, continuous states, we instead assume knowledge of a state encoder, so we can estimate the current state using Equation 2.2. At the start of each timestep t, the assistant collects an observation $\mathbf{o}_t$ from the environment. The assistant then optimizes a synthetic observation $\tilde{\mathbf{o}}_t$ that, when shown to the user, will induce beliefs that match the assistant's (Equation 2.3). The user sees the synthetic observation $\tilde{\mathbf{o}}_t$, takes an action $\mathbf{a}_t$, and the environment generates the next state $\mathbf{s}_{t+1}$. At the end of each episode, we ask the user what task they were trying to perform, add the episode to the dataset $\mathcal{D}$, and re-train the user model $\hat{b}_{\mathbf{H}}$ using Equation 2.5.

2.3 Related Work

Modeling human beliefs, preferences, and behavior. Inverse planning [7] and inverse reinforcement learning [80] learn a model of the user's reward function from demonstrated

Algorithm 1 Assistive State Estimation (ASE)

Require $b_{\text{init}} \in \mathcal{B}$ ▷ initial model of user
if $\mathcal{S}$ is discrete **then**
  Require $p^{\text{init}}(\mathbf{s}_0), p^{\text{dyn}}(\mathbf{s}'|\mathbf{s},\mathbf{a}), p^{\text{obs}}(\mathbf{o}|\mathbf{s})$ ▷ for assistant's belief update in Equation 2.1
else if $\mathcal{S}$ is continuous **then**
  Require state encoder $f(\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1})$ ▷ for assistant's belief update in Equation 2.2
Initialize $\mathcal{D} \leftarrow \emptyset$ ▷ user demonstrations
Initialize $\hat{b}_{\mathbf{H}} \leftarrow b_{\text{init}}$ ▷ assistant's model of user
while true **do**
  $s_0 \sim p^{\text{init}}(\mathbf{s}_0)$
  for $t \in \{0, 1, 2, ..., T-1\}$ **do**
    $\mathbf{o}_t \sim p^{\text{obs}}(\mathbf{o}_t|\mathbf{s}_t)$ ▷ assistant sees true observation, updates beliefs
    $\tilde{\mathbf{o}}_t \leftarrow \arg\min_{\tilde{\mathbf{o}}_t \in \Omega} D_{\text{KL}}(b(\mathbf{s}_t|\mathbf{o}_t) \parallel \hat{b}_{\mathbf{H}}(\mathbf{s}_t|\tilde{\mathbf{o}}_t))$ ▷ assistant synthesizes observation
    $\mathbf{a}_t \sim p(\mathbf{a}_t|\tilde{\mathbf{o}}_{0:t}, \mathbf{a}_{0:t-1})$ ▷ user sees synthetic observation, updates beliefs, takes action
    $\mathbf{s}_{t+1} \sim p^{\text{dyn}}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\tilde{\mathbf{o}}_{0:T-1}, \mathbf{a}_{0:T-1})\}$
  $\hat{\theta} \leftarrow \arg\max_\theta \sum_{\tau \in \mathcal{D}} \sum_t \log p(\mathbf{a}_t|\tilde{\mathbf{o}}_{0:t}, \mathbf{a}_{0:t-1}; \theta)$ ▷ assistant learns model of user
  $\hat{b}_{\mathbf{H}} \leftarrow b_{\hat{\theta}}$

actions. These methods typically assume that user actions are near-optimal, and can be affected by random noise [120], risk sensitivity [70], or dynamics model misspecification [95]. The closest prior work learns a reward function or policy from demonstrations, using a behavioral model that allows for false beliefs about the current state [30, 99, 24, 54]. ASE differs in that we explicitly avoid trying to learn the user's task-specific reward function or policy. Instead, we provide the user with task-agnostic assistance by learning a model of the user's state estimation process, and supplying the user with informative observations.

Task-specific assistance via communication and visualization. [16] assist users by modeling their internal beliefs and communicating observations that induce optimal actions, but require knowledge of the user's reward function at test time, assume a discrete state space, and do not learn a personalized model of the user's internal state estimation process. ASE does not assume knowledge of the task rewards at test time, can be applied to domains with high-dimensional, continuous observations like images, and interactively learns a user model. [52] learn to visualize high-dimensional examples to assist users with one-step classification tasks, whereas we focus on sequential decision-making and make minimal assumptions about the desired task. [116] use a human-in-the-loop reinforcement learning method to train an agent to sequentially explain black-box model predictions to a human auditor, where the agent is rewarded for causing the user's mental model of the predictive model to match the actual predictive model. Our work differs in that it focuses on improving users' situational awareness in control tasks with partial observations, rather than improving model interpretability.

Assistive navigation for visually-impaired users. The closest prior work plans instructional guidance actions under uncertainty about how the user will respond to instructions [82]. We take a complementary approach to assistance that focuses on situational awareness: we help the user estimate their current state, so that they can make more informed decisions in general. In particular, ASE could be useful for systems that inform users about nearby objects and points of interest through haptic or audio feedback [110, 97]: as users build a mental map of their environment to support navigation [8, 42], ASE can learn a user model that captures differences between the mental map and the real environment, then prioritize information that enables the user to localize themselves and nearby obstacles, without overwhelming the user with too much information [83].

2.4 User Studies

In our experiments, we evaluate whether ASE can provide helpful assistance to users; both in the case where we have prior knowledge of their state estimation process, and where we do not have such knowledge and must learn the state estimation model in the loop. We conduct a user study with 12 participants who each perform four tasks: classifying MNIST images under bandwidth constraints [66], playing the Car Racing video game from the OpenAI Gym with observation delay, navigating a simulated 2D environment with limited vision, and playing the Lunar Lander video game from the OpenAI Gym with limited vision [15]. We also conduct experiments with simulated users to study our method under ideal assumptions, and measure the accuracy of the simulated users' internal beliefs (Appendix 6.1 contains details).

Assisting Users with Known Biases

Our first set of user studies seeks to answer **Q1**: can we assist users when we assume we know their state estimation process? We test this hypothesis on MNIST image classification, and the Car Racing video game from the OpenAI Gym.

MNIST Image Classification with a Bandwidth Constraint

In this experiment, we test ASE's ability to assist the user when the user cannot leverage their memory of past actions and their knowledge of the state transition dynamics to infer the current state, and must rely entirely on observations. To that end, we formulate a sequential image classification task in which the user's actions have no effect on the state. We take the standard MNIST digit classification problem and intentionally introduce a bandwidth constraint: at each timestep, the user is shown one row of 28 pixels in the 28x28 image, and must try to classify the image given only the pixels observed so far. The assistant observes the full image at the start of the episode, and aims to help the user classify the image as quickly as possible by showing the user informative pixels.

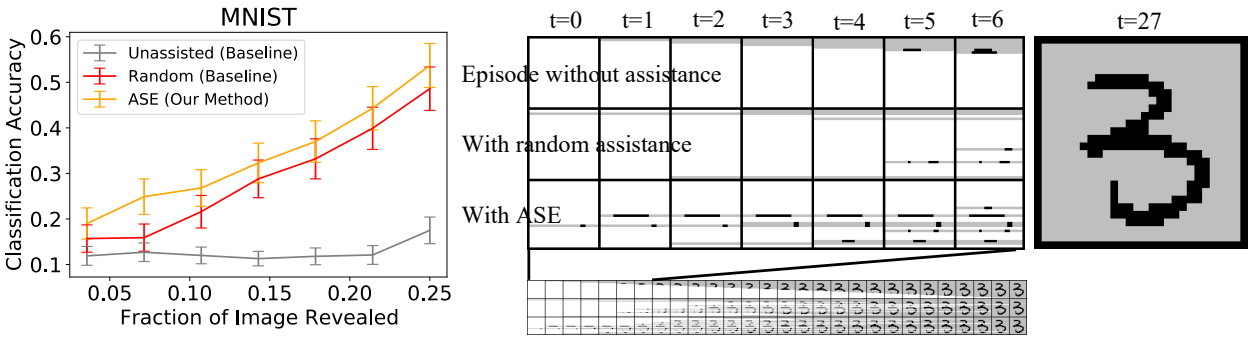

Figure 2.2: MNIST image classification experiments that address **Q1**—can we assist users when we assume we know their state estimation process?—by comparing our method (ASE), which synthesizes an informative observation under the assumption that the user's belief update is similar to the assistant's, to baselines that either use ambient observations generated by the environment (Unassisted) or randomly generate observations (Random). ASE tends to quickly reveal rows near the middle and rows with many non-zero pixels, enabling the user to more accurately guess the label earlier. In the unassisted condition, revealing rows in order from top to bottom is not as quick to reveal informative pixels. The random baseline tends to spread them out uniformly throughout the image, which is a good strategy in the long run but does not necessarily reveal informative pixels early in the episode. We measure standard error across 100 episodes.

The assistant uses a recurrent neural network state encoder f to compute the belief update b via Equation 2.2, where f is trained offline to reconstruct the full image given a sequence of pixel observations. We assume that the user's belief update $b_{\mathbf{H}}$ is equivalent to the assistant's belief update b, except that it can only process one row of pixels per timestep. We compute the optimal synthetic observation $\tilde{\mathbf{o}}_t$ by simply enumerating all rows of pixels that have not been shown to the user yet, and computing the KL-divergence (Equation 2.3) for each possible value of $\tilde{\mathbf{o}}_t$. Appendix 6.1 describes the experimental setup in more detail.

Manipulated factors. We evaluate (1) an unassisted baseline that reveals the pixel rows in order from top to bottom; (2) a random baseline that reveals a new pixel row sampled uniformly at random; and (3) ASE.

Dependent measures. We measure the user's classification accuracy at each timestep, in order to capture how quickly the user recognizes the image over the course of an episode.

Subject allocation. We recruited 11 male and 1 female participants, with an average age of 25. Each participant was provided with the rules of the task and example images, then labeled 25 different digits. Each digit was broken down into an episode of 28 partial images, yielding a total of 700 labels per user. To avoid the confounding effect of users learning to classify images more accurately and quickly over time, we randomly interleave episodes from each of the three conditions. For example, episode 1 is unassisted, episode 2 is assisted by ASE, episode 3 is assisted by the random baseline, etc.

Analysis. Figure 2.2 shows that ASE substantially outperforms the unassisted baseline (orange vs. gray curve), and enables the user to classify the digit using fewer timesteps (i.e.,

fewer pixels) than the random baseline (orange vs. red curve). We ran a one-way repeated measures ANOVA on the classification accuracy dependent measure from the random and ASE conditions, with the presence of ASE as a factor and the digit ID and fraction of image revealed as covariates, and found that $f(1, 5452) = 7.97, p < .01$. While the effect was not substantial—the assisted user's least-squares-mean accuracy was 74.2%, while the unassisted user's was 71.7%—the assisted user achieved significantly higher accuracy than the unassisted user. Although the uniform-random baseline happens to perform well on MNIST, it performs extremely poorly in simulation experiments with 2D navigation and Car Racing (Table 6.1 and Figure 6.3 in the appendix).

Car Racing Video Game with Observation Delay

In this experiment, we test ASE's ability to assist the user in a real-time driving game with delayed observations, where the user tends to react to outdated observations as if they are current. Our assistant sees the same delayed observations as the user, but instead of passing them to the user, replaces the user's video feed with synthetic images produced by a generative model. To optimize these images to induce the correct state beliefs in the user, the assistant forward-predicts the current state from the delayed observation and the user's most recent actions, then constructs an image observation representative of the predicted current state. By default, this environment emits a 64x64 RGB image observation with a top-down view of the car, and the user can steer left or right using their keyboard (Figure 2.3). To simulate intermittent observation delays, we set up the environment to alternate between a no-delay phase of emitting new observations immediately (for 5 timesteps) and a delay phase of repeatedly emitting the final observation from the previous no-delay phase (for 5 timesteps). Both the assistant and the user experience the same delay.

The assistant uses a recurrent neural network (RNN) state encoder f to compute the belief update b via Equation 2.2, and a variational auto-encoder (VAE) [59] to synthesize image observations from the hidden states of f. We assume that the user's belief update $b_{\mathbf{H}}$ is identical to the assistant's belief update b, except that $b_{\mathbf{H}}$ treats observations as if they are never delayed. In practice, although users can clearly tell there is a delay, they are incapable of adjusting to it and steer as if there is no delay. If the last d observations are delayed, a straightforward solution emerges from the assistant's belief-matching objective in Equation 2.3: replace the delayed observations $\mathbf{o}_{t-d+1:t}$ with recursively predicted, non-delayed observations $\hat{\mathbf{o}}_{t-d+1:t}$ from the RNN encoder f and VAE image decoder, and show the user the prediction of the current observation: $\tilde{\mathbf{o}}_t \leftarrow \hat{\mathbf{o}}_t$. If the last observation $\mathbf{o}_t$ was not delayed, then the assistant simply shows the ambient observation: $\tilde{\mathbf{o}}_t \leftarrow \mathbf{o}_t$. Appendix 6.1 describes the experimental setup in more detail.

Manipulated factors. We evaluate (1) an unassisted baseline that passively shows the ambient observation generated by the environment and (2) ASE.

Dependent measures. We measure performance using a reward function that penalizes going off road and gives bonuses for visiting new patches of the road.

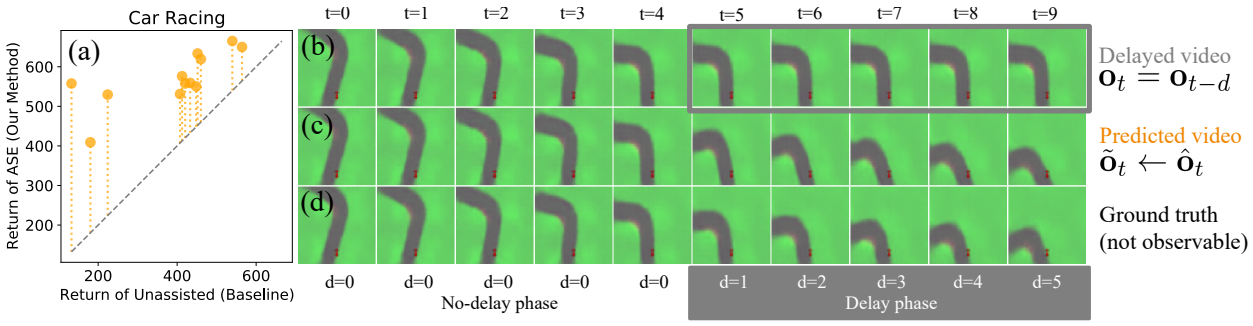

Figure 2.3: Car Racing video game experiments that address **Q1**—can we assist users when we assume we know their state estimation process?—by comparing our method (ASE), which synthesizes an informative observation under the assumption that the user's belief update is similar to the assistant's, to a baseline that always shows the ambient observation generated by the environment (Unassisted). **(a)** Each orange circle represents one of the 12 participants. The dashed gray line shows baseline-equivalent performance, and the dotted orange lines show the difference between assisted and unassisted performance. Per-user return is averaged across 3 episodes (50 seconds each). **(b-d)** Top-down views of approaching a left turn with observation delay d at time t: **(b)** outdated ambient observation $\mathbf{o}_t$, **(c)** forward-predicted observation representative of the current state $\hat{\mathbf{o}}_t$, and **(d)** the ground truth, which cannot be observed by either the user or the assistant. ASE shows the user the forward-predicted observation $\hat{\mathbf{o}}_t$, which is closer to the ground truth than the outdated ambient observation $\mathbf{o}_t$ that the user would see by default, especially when the delay is d is large.

Subject allocation. We recruited 11 male and 1 female participants, with an average age of 25. Each participant was provided with the rules of the task and a short practice period of 2 episodes to familiarize themselves with the controls and dynamics. Each user played in both conditions: unassisted, and assisted by ASE. To avoid the confounding effect of users learning to play the game better over time, we counterbalanced the order of the two conditions. Each condition lasted 3 episodes, with 1000 timesteps (50 seconds) per episode.

Analysis Plot (a) in Figure 2.3 shows that users are able to achieve substantially larger returns (i.e., drive on the road and stay off the grass more often) with the ASE assistant compared to the unassisted condition. ASE makes the user's video feed smoother by predicting the current observation when the true current observation is delayed, which makes real-time, closed-loop control of the car substantially easier. Users in the unassisted condition tended not to change their steering action when the true images were delayed, while assisted users were able to rapidly switch steering actions even during delay phases, by responding to the assistant's synthetic images. We ran a one-way repeated measures ANOVA on the returns from the unassisted and ASE conditions with the presence of ASE as a factor, and found that $f(1, 11) = 41.01, p < .001$. The assisted user achieved significantly higher returns than the unassisted user. The subjective evaluations in Table 6.2 in the appendix corroborate these results: users reported perceiving smaller delays and feeling more in control of the car when they were assisted. One reason that users may have perceived a small delay even in

the assisted condition is that the assistant uses an imperfect, learned state encoder f in its belief update. This suggests that even when the assistant has an imperfect state estimation process, ASE can still improve the user's task performance; the assistant's process just has to be more accurate than the user's.

Learning to Assist Users with Unknown Biases

Our second set of user studies seeks to answer **Q2**: can we assist users when we do not know their state estimation process, and must learn a model of it? We test this hypothesis first in a 2D navigation task, then in a variant of the Lunar Lander video game from the OpenAI Gym.

2D Navigation with Incomplete Mental Map of Object Locations

In this experiment, we intentionally introduce a bias into the user's perception (unknown to ASE), and test whether ASE can learn a user model that recovers this bias. Inspired by the assistive navigation systems discussed in Section 2.3, which inform visually-impaired users about nearby points of interest through audio feedback, we set up a simulated 2D navigation task in which the user cannot directly access their current position and orientation, but can infer them using text observations that describe nearby objects. To incept a controlled user bias, we intentionally do not include the locations of certain objects in the user's 'mental map,' which prevents the user from using observations of those objects to infer their current state as they navigate to a goal. To effectively assist the user, ASE must learn that the user ignores observations that mention these unknown objects. Figure 2.4 illustrates the 'mental map' of the 5x5 grid world shown to the user. At each timestep, the user is told about one of the objects directly in front of them. Some objects are unique, while other objects have multiple instances that exist in different locations (e.g., one computer vs. multiple plants). The objects are divided into 3 categories: (a) unique but unknown, (b) not unique but known, and (c) both unique and known.

The assistant knows the locations of all objects, and can observe all objects in front of the user simultaneously. Following Section 2.2, we parameterize the user model b_θ as a Bayesian belief update (Equation 2.1) that uses observation model $p_\theta^{\text{obs}}(\mathbf{o}|\mathbf{s})$. The parameter $\theta \in [0, 1]$ weights the observation probabilities of objects in category (a): $p_\theta^{\text{obs}}(\mathbf{o}|\mathbf{s}) \propto \theta \cdot p^{\text{obs}}(\mathbf{o}|\mathbf{s})$ for all objects $\mathbf{o}$ in category (a). Because we intentionally make category (a) objects unknown to the user, we know the true value: $\theta = 0$. We would like ASE to learn this value from the user's behavior. Appendix 6.1 describes the experimental setup in more detail.

Manipulated factors. We evaluate (1) an unassisted baseline that passively shows the ambient observation generated by the environment; (2) a naïve version of ASE that does not train the user model, and instead continues using the initial model b_{init} where $\theta = 1$; and (3) ASE. In ASE, we learn θ from the episodes collected in conditions 1 and 2. In practice, we pool the data from the first k participants to train the model for the k-th participant, since

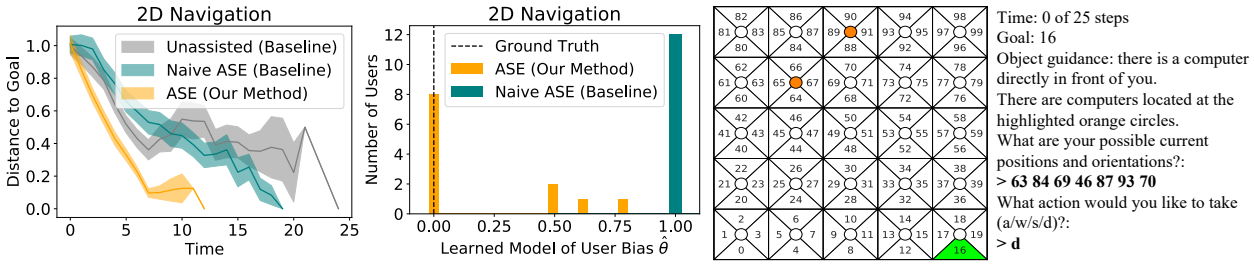

Figure 2.4: 2D navigation experiments that address **Q2**—can we assist users when we do not know their state estimation process, and must learn a model of it?—by comparing our method (ASE), which learns a model of the user's belief update then synthesizes observations that are informative under the learned model, to baselines that either use ambient observations generated by the environment (Unassisted) or assume the user's belief update is similar to the assistant's and do not learn a model (Naïve ASE). We measure standard error across 55 episodes (5 episodes per user). The results show that ASE is able to learn the user's bias parameter θ, which enables the personalized assistant to give the user more informative observations than the naive assistant. The user's console interface shows them the goal state (green) and the locations of the currently-observed object in their mental map (orange circles). The user can choose to move forward (w), turn 90 degrees (a/d), or stay still and wait for another observation (s).

the small amount of data collected for each individual user is too noisy to learn an accurate model from, and because the true model does not vary between users.

Dependent measures. We measure the distance from the user's current position to their goal position (normalized by distance from initial position to goal position) at each timestep, in order to capture how quickly the user moves toward the goal throughout the episode.

Subject allocation. We recruited 11 male and 1 female participants, with an average age of 25. Each participant was provided with the rules of the task and a short practice period of 3 episodes to familiarize themselves with the controls and dynamics. Each user played in all three conditions: unassisted, assisted by naïve ASE, and assisted by ASE. We counterbalanced the order of the unassisted and naïve ASE conditions. We could not counterbalance the order of the ASE condition to control for the learning effect, since ASE learns $\hat{\theta}$ from the data collected in the unassisted and naïve ASE conditions. Figure 6.4 in the appendix shows that the introduction of the ASE assistant sharply improves the user's performance across episodes, suggesting the learning effect was not a substantial confounder. Each condition lasted 5 episodes, with 25 timesteps per episode.

Analysis. Figure 2.4 shows that users are able to move toward the goal substantially faster with the ASE assistant compared to the unassisted condition. Furthermore, learning a model of the user's observation model substantially improved the assistant's performance compared to the naïve assistant. In the unassisted condition, the user receives many observations of objects in category (b), which are known but relatively uninformative since they have multiple known locations. In the naïve condition, the user receives many observations of objects in categories (a), which are unique but unknown. ASE learns that objects in category

(a) are unknown (i.e., $\hat{\theta} = 0$), so it only shows the unique and known objects in category (c). We ran a one-way repeated measures ANOVA on the time-to-goal dependent measure from the unassisted and ASE conditions with the presence of ASE as a factor, and found that $f(1, 11) = 18.02, p < .01$. The assisted user reached the goal significantly faster than the unassisted user. The subjective evaluations in Table 6.3 in the appendix corroborate these results: users reported finding the observations more helpful in the ASE condition compared to the unassisted condition.

Lunar Lander Video Game with Limited Vision

In this experiment, we evaluate whether ASE can learn a personalized model of naturally-occurring user biases in the Lunar Lander game, in which users tend to land at an unsafe angle. We conjecture that this suboptimal user behavior is caused by underestimating the lander's tilt, and that the assistant might learn to help the user by showing them an image in which the lander's tilt is exaggerated beyond the ground truth. At each timestep, the environment emits an image of the lander, and the user can fire the left or right thruster using their keyboard (plot (b) in Figure 2.5). The objective is to make sure the lander stays level as it descends, using the thrusters to prevent it from tilting left or right. The image includes a visual indicator of the lander's tilt, which is separate from the body of the lander (plot (e) in Figure 2.5). The assistant is capable of freely changing the angle of this tilt indicator in the image observation shown to the user, but cannot change any other aspect of the image (e.g., the lander body itself).

To simplify our model of the user, we focus on one feature: the angle of the lander. In the user model, an observation is characterized by the angle of the tilt indicator: the observation space is $\Omega = [-\pi, \pi]$. A state is characterized by the lander's angle: the state space is $\mathcal{S} = [-\pi, \pi]$. By default, the angle of the tilt indicator is equal to the lander's angle: $p^{\text{obs}}(\mathbf{o}|\mathbf{s}) = \mathbb{1}[\mathbf{o} = \mathbf{s}]$. We assume the user's suboptimality stems from incorrectly inferring the lander's tilt from the angle of the tilt indicator: it is easy to tell when the lander is severely tilted, but harder to tell when the lander is only slightly tilted. This is a problem for the user, since keeping the lander level requires detecting tilt early when it is still small, so that the thrusters have enough time to force the lander upright. We represent the user model b_θ using a simple logistic model: $b_\theta(\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}) = -\pi + 2\pi \cdot \sigma(\theta_0 + \theta_1 \cdot \mathbf{o}_t)$, where σ is the sigmoid function. The optimal synthetic observation anticipates the user's internal distortion: $\tilde{\mathbf{o}}_t \leftarrow b_{\hat{\theta}}^{-1}(\mathbf{o}_t)$. Appendix 6.1 describes the experimental setup in more detail.

Manipulated factors. We evaluate (1) an unassisted baseline that rotates the tilt indicator to exactly match the lander's angle and (2) ASE. In ASE, we learn θ from the episodes collected in the unassisted condition, as well as 5 assisted episodes generated iteratively using Algorithm 1.

Dependent measures. We measure the absolute value of the lander's angle $|\mathbf{s}_t|$ (i.e., 'tilt') at each timestep, in order to capture how well the user is able to stabilize the lander throughout the episode.

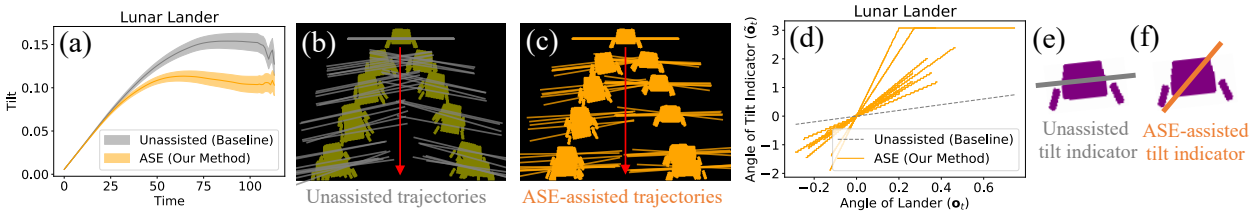

Figure 2.5: Lunar Lander experiments that address **Q2**—can we assist users when we do not know their state estimation process, and must learn a model of it?—by comparing our method (ASE), which learns a model of the user's belief update then synthesizes observations that are informative under the learned model, to a baseline that always shows the ambient observation generated by the environment (Unassisted). **(a)** We measure standard error across 120 episodes (10 episodes per user). **(b)** Sample of unassisted trajectories from the user studies. **(c)** With assistance, the user keeps the lander more level. **(d-f)** ASE tends to exaggerate the tilt indicator (orange vs. gray line), and personalizes the exaggeration to the user (each orange line in **(d)** corresponds to a different user).

Subject allocation. We recruited 11 male and 1 female participants, with an average age of 25. Each participant was provided with the rules of the task and a short practice period of 5 episodes to familiarize themselves with the controls and dynamics. Each user played in both conditions: unassisted, then assisted by ASE. We could not counterbalance the order of the two conditions to control for the learning effect, since ASE learns $\hat{\theta}$ from the data collected in the unassisted condition. Figure 6.4 in the appendix shows that the introduction of the ASE assistant sharply improves the user's performance across episodes, suggesting the learning effect was not a substantial confounder. Each condition lasted 10 episodes, with 150 timesteps (10 seconds) per episode.

Analysis. Plot (a) in Figure 2.5 shows that users are able to substantially decrease the tilt of the lander throughout the episode with the ASE assistant compared to the unassisted condition. The assistant learns that the user infers a smaller lander angle than the observed tilt indicator's angle. This leads to an assistance policy that exaggerates observations by rotating the tilt indicator to exceed the lander's true angle. Furthermore, plot (d) in Figure 2.5 shows that the learned distortion model varies across users. We ran a one-way repeated measures ANOVA on the average tilt dependent measure from the unassisted and ASE conditions with the presence of ASE as a factor, and found that $f(1, 11) = 6.30, p < .05$. The assisted user's average tilt was significantly smaller than the unassisted user's. The subjective evaluations in Table 6.4 in the appendix corroborate these results: users reported finding it easier to tell when the lander was tilted in the ASE condition compared to the unassisted condition.

2.5 Discussion

Summary. We propose the assistive state estimation (ASE) algorithm for helping users with perception in partially observable Markov decision processes. The key idea is to synthesize observations that induce more accurate beliefs about the current state than the ambient observations. In our first set of user studies, we show that ASE can assist users with limited sensor bandwidth by identifying subsets of informative pixels for image classification, and that ASE can assist users with observation delay in a driving video game by using a dynamics model to predict the current state of the world and constructing a hypothetical current observation. In our second set of user studies, we show that ASE can assist irrational users with navigating a 2D world by informing them about nearby landmarks, and detecting and minimizing tilt in the Lunar Lander game by exaggerating a visual indicator of the lander's tilt. These experiments broadly illustrate how assisting users with state estimation while making minimal assumptions about the desired task can improve real users' task performance. In addition to the user studies, we run simulation experiments on indoor navigation and MNIST digit classification that show (1) ASE not only improves users' task performance, but also improves the accuracy of simulated users' internal beliefs; and (2) the quality of assistance increases with the number of user-in-the-loop episodes collected.

Limitations and future work. ASE assumes that we can solve for a near-optimal state estimator in order to compute the assistant's belief update b via Equations 2.1 or 2.2, and can solve for near-optimal policies π in order to model user actions via Equation 2.4, which may not be feasible in real-world domains. Fortunately, recent work has demonstrated substantial improvements in learning approximate state estimators and policies in complex environments with image observations, unknown dynamics, and other challenges [117, 45]. While our proof of concept does not make use of these advances, incorporating them into a more practical assistive state estimation system is a promising direction for future work. Furthermore, our user studies are limited in that we do not know if the improvement in users' task performance is caused by more accurate internal beliefs, or by some other feature of the assistance condition, since we cannot directly measure those beliefs in real users (only in simulated users). Our ultimate goal is for the user to make better decisions when assisted, and the results show that the belief-matching objective in Equation 2.3 accomplishes this in four different domains. Even so, one direction for future work is to design experiments that more directly measure the user's internal beliefs during decision-making.

Chapter 3

Inferring Beliefs about Dynamics from Behavior

There was time only to see the lights of the Aurors' curses arrowing down at him, slightly angle the broomstick to avoid them, realize that the broomstick was simply continuing on with mostly the same momentum instead of going in the direction he pointed it, and activate the wordless concepts
crap
and
Newton

—Eliezer Yudkowsky, *Harry Potter and the Methods of Rationality (2010)*

Inferring intent from observed behavior has been studied extensively within the frameworks of Bayesian inverse planning and inverse reinforcement learning. These methods infer a goal or reward function that best explains the actions of the observed agent, typically a human demonstrator. Another agent can use this inferred intent to predict, imitate, or assist the human user. However, a central assumption in inverse reinforcement learning is that the demonstrator is close to optimal. While models of suboptimal behavior exist, they typically assume that suboptimal actions are the result of some type of random noise or a known cognitive bias, like temporal inconsistency.

In this chapter, we take an alternative approach, and model suboptimal behavior as the result of internal model misspecification: the reason that user actions might deviate

from near-optimal actions is that the user has an incorrect set of beliefs about the rules—the dynamics—governing how actions affect the environment. Our insight is that while demonstrated actions may be suboptimal in the real world, they may actually be near-optimal with respect to the user's *internal* model of the dynamics. By estimating these internal beliefs from observed behavior, we arrive at a new method for inferring intent.

We demonstrate in simulation and in a user study with 12 participants that this approach enables us to more accurately model human intent, and can be used in a variety of applications, including offering assistance in a shared autonomy framework and inferring human preferences.

3.1 Introduction

Characterizing the drive behind human actions in the form of a goal or reward function is broadly useful for predicting future behavior, imitating human actions in new situations, and augmenting human control with automated assistance—critical functions in a wide variety of applications, including pedestrian motion prediction [121], virtual character animation [84], and robotic teleoperation [77]. For example, remotely operating a robotic arm to grasp objects can be challenging for a human user due to unfamiliar or unintuitive dynamics of the physical system and control interface. Existing frameworks for assistive teleoperation and shared autonomy aim to help users perform such tasks [77, 55, 100, 14, 94]. These frameworks typically rely on existing methods for intent inference in the sequential decision-making context, which use Bayesian inverse planning or inverse reinforcement learning to learn the user's goal or reward function from observed control demonstrations. These methods typically assume that user actions are near-optimal, and deviate from optimality due to random noise [120], specific cognitive biases in planning [30, 29, 9], or risk sensitivity [70].

The key insight in this paper is that suboptimal behavior can also arise from a mismatch between the dynamics of the real world and the user's internal beliefs of the dynamics, and that a user policy that appears suboptimal in the real world may actually be near-optimal with respect to the user's internal dynamics model. As resource-bounded agents living in an environment of dazzling complexity, humans rely on intuitive theories of the world to guide reasoning and planning [36, 46]. Humans leverage internal models of the world for motor control [115, 57, 26, 72, 107], goal-directed decision making [12], and representing the mental states of other agents [87]. Simplified internal models can systematically deviate from the real world, leading to suboptimal behaviors that have unintended consequences, like hitting a tennis ball into the net or skidding on an icy road. For example, a classic study in cognitive science shows that human judgments about the physics of projectile motion are closer to Aristotelian impetus theory than to true Newtonian dynamics—in other words, people tend to ignore or underestimate the effects of inertia [18]. Characterizing the gap between internal models and reality by modeling a user's internal predictions of the effects of their actions allows us to better explain observed user actions and infer their intent.

The main contribution of this paper is a new algorithm for intent inference that first estimates a user's internal beliefs of the dynamics of the world using observations of how

they act to perform known tasks, then leverages the learned internal dynamics model to infer intent on unknown tasks. In contrast to the closest prior work [50, 38], our method scales to problems with high-dimensional, continuous state spaces and nonlinear dynamics. Our internal dynamics model estimation algorithm assumes the user takes actions with probability proportional to their exponentiated soft Q-values. We fit the parameters of the internal dynamics model to maximize the likelihood of observed user actions on a set of tasks with known reward functions, by tying the internal dynamics to the soft Q function via the soft Bellman equation. At test time, we use the learned internal dynamics model to predict the user's desired next state given their current state and action input.

We run experiments first with simulated users, testing that we can recover the internal dynamics, even in MDPs with a continuous state space that would otherwise be intractable for prior methods. We then run a user study with 12 participants in which humans play the Lunar Lander game (screenshot in Figure 3.1). We recover a dynamics model that explains user actions better than the real dynamics, which in turn enables us to assist users in playing the game by transferring their control policy from the recovered internal dynamics to the real dynamics.

3.2 Background

Inferring intent in sequential decision-making problems has been heavily studied under the framework of inverse reinforcement learning (IRL), which we build on in this work. The aim of IRL is to learn a user's reward function from observed control demonstrations. IRL algorithms are not directly applicable to our problem of learning a user's beliefs about the dynamics of the environment, but they provide a helpful starting point for thinking about how to extract hidden properties of a user from observations of how they behave.

In our work, we build on the maximum causal entropy (MaxCausalEnt) IRL framework [119, 11, 93, 79, 50]. In an MDP with a discrete action space $\mathcal{A}$, the human demonstrator is assumed to follow a policy π that maximizes an entropy-regularized reward $R(s, a, s')$ under dynamics $T(s'|s, a)$. Equivalently,

$$\pi(a|s) \triangleq \frac{\exp\left(Q(s,a)\right)}{\sum_{a' \in \mathcal{A}} \exp\left(Q(s,a')\right)}, \tag{3.1}$$

where Q is the soft Q function, which satisfies the soft Bellman equation [119],

$$Q(s,a) = \mathbb{E}_{s' \sim T(\cdot|s,a)}\left[R(s,a,s') + \gamma V(s')\right], \tag{3.2}$$

with V the soft value function,

$$V(s) \triangleq \log\left(\sum_{a \in \mathcal{A}} \exp\left(Q(s,a)\right)\right). \tag{3.3}$$

Prior work assumes T is the true dynamics of the real world, and fits a model of the reward R that maximizes the likelihood (given by Equation 3.1) of some observed demonstrations of the user acting in the real world. In our work, we assume access to a set of training tasks for which the rewards R are known, fit a model of the internal dynamics T that is allowed to deviate from the real dynamics, then use the recovered dynamics to infer intent (e.g., rewards) in new tasks.

3.3 Internal Dynamics Model Estimation

We split up the problem of intent inference into two parts: learning the internal dynamics model from user demonstrations on known tasks (the topic of this section), and using the learned internal model to infer intent on unknown tasks (discussed later in Section 3.4). We assume that the user's internal dynamics model is stationary, which is reasonable for problems like robotic teleoperation when the user has some experience practicing with the system but still finds it unintuitive or difficult to control. We also assume that the real dynamics are known ex-ante or learned separately.

Our aim is to recover a user's implicit beliefs about the dynamics of the world from observations of how they act to perform a set of tasks. The key idea is that, when their internal dynamics model deviates from the real dynamics, we can no longer simply fit a dynamics model to observed state transitions. Standard dynamics learning algorithms typically assume access to (s, a, s') examples, with (s, a) features and s' labels, that can be used to train a classification or regression model $p(s'|s, a)$ using supervised learning. In our setting, we instead have (s, a) pairs that indirectly encode the state transitions that the user expected to happen, but did not necessarily occur, because the user's internal model predicted different outcomes s' than those that actually occurred in the real world. Our core assumption is that the user's policy is near-optimal with respect to the unknown internal dynamics model. To this end, we propose a new algorithm for learning the internal dynamics from action demonstrations: inverse soft Q-learning.

Inverse Soft Q-Learning

The key idea behind our algorithm is that we can fit a parametric model of the internal dynamics model T that maximizes the likelihood of observed action demonstrations on a set of training tasks with known rewards by using the soft Q function as an intermediary.[1] We tie the internal dynamics T to the soft Q function via the soft Bellman equation (Equation 3.2), which ensures that the soft Q function is induced by the internal dynamics T. We tie the soft Q function to action likelihoods using Equation 3.1, which encourages the soft Q function to explain observed actions. We accomplish this by solving a constrained optimization problem

[1] Our algorithm can in principle learn from demonstrations even when the rewards are unknown, but in practice we find that this relaxation usually makes learning the correct internal dynamics too difficult.

in which the demonstration likelihoods appear in the objective and the soft Bellman equation appears in the constraints.

Formulating the optimization problem. Assume the action space $\mathcal{A}$ is discrete.[2] Let $i \in \{1, 2, ..., n\}$ denote the training task, $R_i(s, a, s')$ denote the known reward function for task i, T denote the unknown internal dynamics, and Q_i denote the unknown soft Q function for task i. We represent Q_i using a function approximator $Q_{\boldsymbol{\theta_i}}$ with parameters $\boldsymbol{\theta_i}$, and the internal dynamics using a function approximator $T_{\boldsymbol{\phi}}$ parameterized by $\boldsymbol{\phi}$. Note that, while each task merits a separate soft Q function since each task has different rewards, all tasks share the same internal dynamics.

Recall the soft Bellman equation (Equation 3.2), which constrains Q_i to be the soft Q function for rewards R_i and internal dynamics T. An equivalent way to express this condition is that Q_i satisfies $\delta_i(s, a) = 0 \; \forall s, a$, where δ_i is the soft Bellman error:

$$\delta_i(s, a) \triangleq Q_i(s, a) - \int_{s' \in \mathcal{S}} T(s'|s, a) \left(R_i(s, a, s') + \gamma V_i(s')\right) \mathrm{d}s'. \tag{3.4}$$

We impose the same condition on $Q_{\boldsymbol{\theta_i}}$ and $T_{\boldsymbol{\phi}}$, i.e., $\delta_{\boldsymbol{\theta_i},\boldsymbol{\phi}}(s, a) = 0 \; \forall s, a$. We assume our representations are expressive enough that there exist values of $\boldsymbol{\theta_i}$ and $\boldsymbol{\phi}$ that satisfy the condition. We fit parameters $\boldsymbol{\theta_i}$ and $\boldsymbol{\phi}$ to maximize the likelihood of the observed demonstrations while respecting the soft Bellman equation by solving the constrained optimization problem

$$\begin{aligned} \underset{\{\boldsymbol{\theta_i}\}_{i=1}^n, \boldsymbol{\phi}}{\mathit{minimize}} \quad & \sum_{i=1}^{n} \sum_{(s,a) \in \mathcal{D}_i^{\text{demo}}} - \log \pi_{\boldsymbol{\theta_i}}(a|s) \\ \text{subject to} \quad & \delta_{\boldsymbol{\theta_i},\boldsymbol{\phi}}(s, a) = 0 \; \forall i \in \{1, 2, ..., n\}, s \in \mathcal{S}, a \in \mathcal{A}, \end{aligned} \tag{3.5}$$

where $\mathcal{D}_i^{\text{demo}}$ are the demonstrations for task i, and $\pi_{\boldsymbol{\theta_i}}$ denotes the action likelihood given by $Q_{\boldsymbol{\theta_i}}$ and Equation 3.1.

Solving the optimization problem. We use the penalty method [10] to approximately solve the constrained optimization problem described in Equation 3.5, which recasts the problem as unconstrained optimization of the cost function

$$c(\boldsymbol{\theta}, \boldsymbol{\phi}) \triangleq \sum_{i=1}^{n} \sum_{(s,a) \in \mathcal{D}_i^{\text{demo}}} - \log \pi_{\boldsymbol{\theta_i}}(a|s) + \frac{\rho}{2} \sum_{i=1}^{n} \int_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} (\delta_{\boldsymbol{\theta_i},\boldsymbol{\phi}}(s, a))^2 \mathrm{d}s, \tag{3.6}$$

where ρ is a constant hyperparameter, $\pi_{\boldsymbol{\theta_i}}$ denotes the action likelihood given by $Q_{\boldsymbol{\theta_i}}$ and Equation 3.1, and $\delta_{\boldsymbol{\theta_i},\boldsymbol{\phi}}$ denotes the soft Bellman error, which relates $Q_{\boldsymbol{\theta_i}}$ to $T_{\boldsymbol{\phi}}$ through Equation 3.4.

For MDPs with a discrete state space $\mathcal{S}$, we minimize the cost as is. MDPs with a continuous state space present two challenges: (1) an intractable integral over states in the

[2] We assume a discrete action space to simplify our exposition and experiments. Our algorithm can be extended to handle MDPs with a continuous action space using existing sampling methods [44].

sum over penalty terms, and (2) integrals over states in the expectation terms of the soft Bellman errors δ (recall Equation 3.4). To tackle (1), we resort to constraint sampling [17]; specifically, randomly sampling a subset of state-action pairs $\mathcal{D}_i^{\text{samp}}$ from rollouts of a random policy in the real world. To tackle (2), we choose a deterministic model of the internal dynamics $T_{\boldsymbol{\phi}}$, which simplifies the integral over next states in Equation 3.4 to a single term[3].

In our experiments, we minimize the objective in Equation 3.6 using Adam [58]. We use a mix of tabular representations, structured linear models, and relatively shallow multi-layer perceptrons to model $Q_{\boldsymbol{\theta_i}}$ and $T_{\boldsymbol{\phi}}$. In the tabular setting, $\boldsymbol{\theta_i}$ is a table of numbers with a separate entry for each state-action pair, and $\boldsymbol{\phi}$ can be a table with an entry between 0 and 1 for each state-action-state triple. For linear and neural network representations, $\boldsymbol{\theta_i}$ and $\boldsymbol{\phi}$ are sets of weights.

Regularizing the Internal Dynamics Model

One issue with our approach to estimating the internal dynamics is that there tend to be multiple feasible internal dynamics models that explain the demonstration data equally well, which makes the correct internal dynamics model difficult to identify. We propose two different solutions to this problem: collecting demonstrations on multiple training tasks, and imposing a prior on the learned internal dynamics that encourages it to be similar to the real dynamics.

Multiple training tasks. If we only collect demonstrations on $n = 1$ training tasks, then at any given state s and action a, the recovered internal dynamics may simply assign a likelihood of one to the next state s' that maximizes the reward function $R_1(s, a, s')$ of the single training task. Intuitively, if our algorithm is given user demonstrations on only one task, then the user's actions can be explained by an internal dynamics model that always predicts the best possible next state for that one task (e.g., the target in a navigation task), no matter the current state or user action. We can mitigate this problem by collecting demonstrations on $n > 1$ training tasks, which prevents degenerate solutions by forcing the internal dynamics to be consistent with a diverse set of user policies.

Action intent prior. In our experiments, we also explore another way to regularize the learned internal dynamics: imposing the prior that the learned internal dynamics $T_{\boldsymbol{\phi}}$ should be similar to the known real dynamics T^{real} by restricting the support of $T_{\boldsymbol{\phi}}(\cdot|s, a)$ to states s' that are reachable in the real dynamics. Formally,

$$T_{\boldsymbol{\phi}}(s'|s, a) \triangleq \sum_{a^{\text{int}} \in \mathcal{A}} T^{\text{real}}(s'|s, a^{\text{int}}) f_{\boldsymbol{\phi}}(a^{\text{int}}|s, a) \tag{3.7}$$

where a is the user's action, a^{int} is the user's intended action, and $f_{\boldsymbol{\phi}} : \mathcal{S} \times \mathcal{A}^2 \to [0, 1]$ captures the user's 'action intent'—the action they would have taken if they had perfect knowledge of the real dynamics. This prior changes the structure of our internal dynamics model to predict the user's intended action with respect to the real dynamics, rather than

[3] Another potential solution is sampling states to compute a Monte Carlo estimate of the integral.

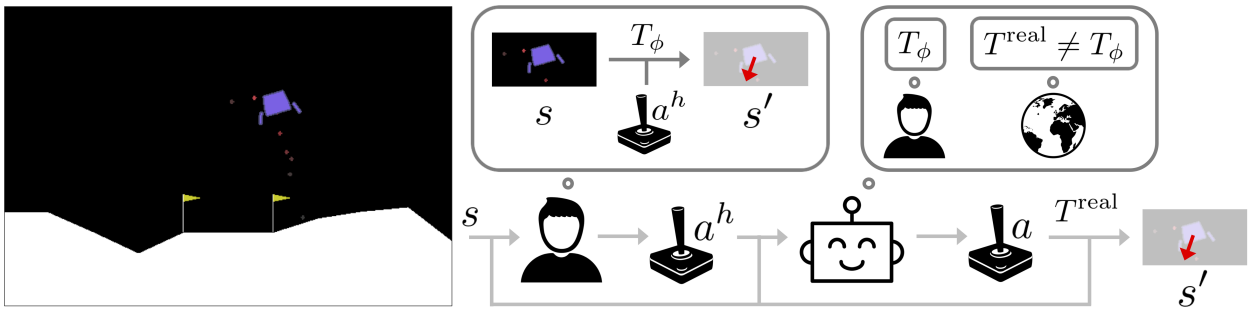

Figure 3.1: A high-level schematic of our internal-to-real dynamics transfer algorithm for shared autonomy, which uses the internal dynamics model learned by our method to assist the user with an unknown control task; in this case, landing the lunar lander between the flags. The user's actions are assumed to be consistent with their internal beliefs about the dynamics T_ϕ, which differ from the real dynamics T^{real}. Our system models the internal dynamics to determine where the user is trying to go next, then acts to get there.

directly predicting their intended next state. Note that, when we use this action intent prior, T_ϕ is no longer directly modeled. Instead, we model f_ϕ and use Equation 3.7 to compute T_ϕ.

In our experiments, we examine the effects of employing multiple training tasks and imposing the action intent prior, together and in isolation.

3.4 Using Learned Internal Dynamics Models

The ability to learn internal dynamics models from demonstrations is broadly useful for intent inference. In our experiments, we explore two applications: (1) shared autonomy, in which a human and robot collaborate to solve a challenging real-time control task, and (2) learning the reward function of a user who generates suboptimal demonstrations due to internal model misspecification. In (1), intent is formalized as the user's desired next state, while in (2), the user's intent is represented by their reward function.

Shared Autonomy via Internal-to-Real Dynamics Transfer

Many control problems involving human users are challenging for autonomous agents due to partial observability and imprecise task specifications, and are also challenging for humans due to constraints such as bounded rationality [105] and physical reaction time. Shared autonomy combines human and machine intelligence to perform control tasks that neither can on their own, but existing methods have the basic requirement that the machine either needs a description of the task or feedback from the user, e.g., in the form of rewards [55, 14, 94]. We propose an alternative algorithm that assists the user without knowing their reward function by leveraging the internal dynamics model learned by our method. The key idea is formalizing the user's intent as their desired next state. We use the learned internal

dynamics model to infer the user's desired next state given their current state and control input, then execute an action that will take the user to the desired state under the real dynamics; essentially, transferring the user's policy from the internal dynamics to the real dynamics, akin to simulation-to-real transfer for robotic control [23]. See Figure 3.1 for a high-level schematic of this process.

Equipped with the learned internal dynamics model T_{ϕ}, we perform internal-to-real dynamics transfer by observing the user's action input, computing the induced distribution over next states using the internal dynamics, and executing an action that induces a similar distribution over next states in the real dynamics. Formally, for user control input a_t^h and state s_t, we execute action a_t, where

$$a_t \triangleq \underset{a \in A}{arg\,min}\, D_{\mathrm{KL}}(T_{\phi(s_{t+1}}|s_t, a_t^h) \parallel T^{\mathrm{real}}(s_{t+1}|s_t, a)) \tag{3.8}$$

Learning Rewards from Misguided User Demonstrations

Most existing inverse reinforcement learning algorithms assume that the user's internal dynamics are equivalent to the real dynamics, and learn their reward function from near-optimal demonstrations. We explore a more realistic setting in which the user's demonstrations are suboptimal due to a mismatch between their internal dynamics and the real dynamics. Users are 'misguided' in that their behavior is suboptimal in the real world, but near-optimal with respect to their internal dynamics. In this setting, standard IRL algorithms that do not distinguish between the internal and the real dynamics learn incorrect reward functions. Our method can be used to learn the internal dynamics, then explicitly incorporate the learned internal dynamics into an IRL algorithm's behavioral model of the user.

In our experiments, we instantiate prior work with MaxCausalEnt IRL [119], which inverts the behavioral model from Equation 3.1 to infer rewards from demonstrations. We adapt it to our setting, in which the real dynamics are known and the internal dynamics are either learned (separately by our algorithm) or assumed to be the same as the known real dynamics. MaxCausalEnt IRL cannot learn the user's reward function from misguided demonstrations when it makes the standard assumption that the internal dynamics are equal to the real dynamics, but can learn accurate rewards when it instead uses the learned internal dynamics model produced by our algorithm.

3.5 User Study and Simulation Experiments

The purpose of our experiments is two-fold: (1) to test the correctness of our algorithm, and (2) to test our core assumption that a human user's internal dynamics can be different from the real dynamics, and that our algorithm can learn an internal dynamics model that is useful for assisting the user through internal-to-real dynamics transfer. To accomplish (1), we perform three simulated experiments that apply our method to shared autonomy (see Section 3.4) and to learning rewards from misguided user demonstrations (see Section 3.4).

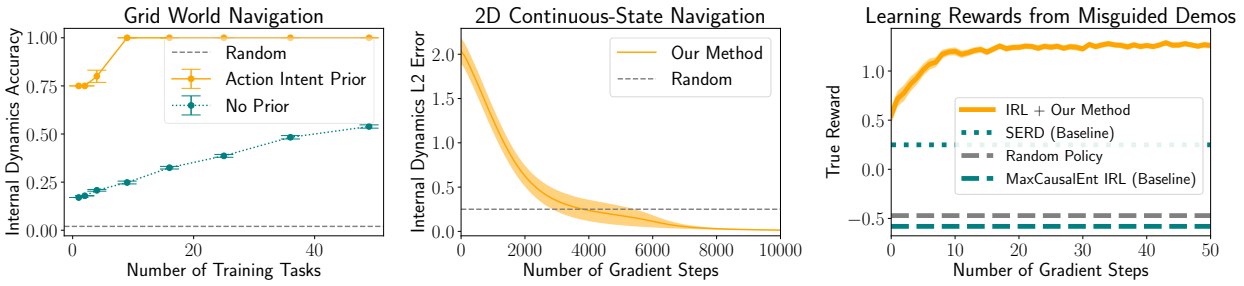

Figure 3.2: **Left, Center:** Error bars show standard error on ten random seeds. Our method learns accurate internal dynamics models, the regularization methods in Section 3.3 increase accuracy, and the approximations for continuous-state MDPs in Section 3.3 do not compromise accuracy. **Right:** Error regions show standard error on ten random tasks and ten random seeds each. Our method learns an internal dynamics model that enables MaxCausalEnt IRL to learn rewards from misguided user demonstrations.

In the shared autonomy experiments, we first use a tabular grid world navigation task to sanity-check our algorithm and analyze the effects of different regularization choices from Section 3.3. We then use a continuous-state 2D navigation task to test our method's ability to handle continuous observations using the approximations described in Section 3.3. In the reward learning experiment, we use the grid world environment to compare the performance of MaxCausalEnt IRL [119] when it assumes the internal dynamics are the same as the real dynamics to when it uses the internal dynamics learned by our algorithm. To accomplish (2), we conduct a user study in which 12 participants play the Lunar Lander game (see Figure 3.1) with and without internal-to-real dynamics transfer assistance. We summarize these experiments in Sections 3.5 and 3.5. Further details are provided in Section 6.2 of the appendix.

Simulation Experiments

Shared autonomy. The grid world provides us with a domain where exact solutions are tractable, which enables us to verify the correctness of our method and compare the quality of the approximation in Section 3.3 with an exact solution to the learning problem. The continuous task provides a more challenging domain where exact solutions via dynamic programming are intractable. In each setting, we simulate a user with an internal dynamics model that is severely biased away from the real dynamics of the simulated environment. The simulated user's policy is near-optimal with respect to their internal dynamics, but suboptimal with respect to the real dynamics. Figure 3.2 (left and center plots) provides overall support for the hypothesis that our method can effectively learn tabular and continuous representations of the internal dynamics for MDPs with discrete and continuous state spaces. The learned internal dynamics models are accurate with respect to the ground truth internal dynamics, and internal-to-real dynamics transfer successfully assists the simulated users. The

learned internal dynamics model becomes more accurate as we increase the number of training tasks, and the action intent prior (see Section 3.3) increases accuracy when the internal dynamics are similar to the real dynamics. These results confirm that our approximate algorithm is correct and yields solutions that do not significantly deviate from those of an exact algorithm. Further results and experimental details are discussed in Sections 6.2 and 6.2 of the appendix.

Learning rewards from misguided user demonstrations. Standard IRL algorithms, such as MaxCausalEnt IRL [119], can fail to learn rewards from user demonstrations that are 'misguided', i.e., systematically suboptimal in the real world but near-optimal with respect to the user's internal dynamics. Our algorithm can learn the internal dynamics model, and we can then explicitly incorporate the learned internal dynamics into the MaxCausalEnt IRL algorithm to learn accurate rewards from misguided demonstrations. We assess this method on a simulated grid world navigation task. Figure 3.2 (right plot) supports our claim that standard IRL is ineffective at learning rewards from misguided user demonstrations. After using our algorithm to learn the internal dynamics and explicitly incorporating the learned internal dynamics into an IRL algorithm's model of the user, we see that it's possible to recover accurate rewards from these misguided demonstrations. Additional information on our experimental setup is available in Section 6.2 of the appendix.

In addition to comparing to the standard MaxCausalEnt IRL baseline, we also conducted a comparison (shown in Figure 3.2) with a variant of the Simultaneous Estimation of Rewards and Dynamics (SERD) algorithm [50] that simultaneously learns rewards and the internal dynamics instead of assuming that the internal dynamics are equivalent to the real dynamics. This baseline performs better than random, but still much worse than our method. This result is supported by the theoretical analysis in Armstrong et al. [4], which characterizes the difficulty of simultaneously deducing a human's rationality—in our case, their internal dynamics model—and their rewards from demonstrations.

User Study on the Lunar Lander Game

Our previous experiments were conducted with simulated expert behavior, which allowed us to control the corruption of the internal dynamics. However, it remains to be seen whether this model of suboptimality effectively reflects real human behavior. We test this hypothesis in the next experiment, which evaluates whether our method can learn the internal dynamics accurately enough to assist real users through internal-to-real dynamics transfer.

Task description. We use the Lunar Lander game from OpenAI Gym [15] (screenshot in Figure 3.1) to evaluate our algorithm with human users. The objective of the game is to land on the ground, without crashing or flying out of bounds, using two lateral thrusters and a main engine. The action space $\mathcal{A}$ consists of six discrete actions. The state $s \in \mathbb{R}^9$ encodes position, velocity, orientation, and the location of the landing site, which is one of nine values corresponding to $n = 9$ distinct tasks. The physics of the game are forward-simulated by a black-box function that takes as input seven hyperparameters, which include engine power and game speed. We manipulate whether or not the user receives internal-to-real

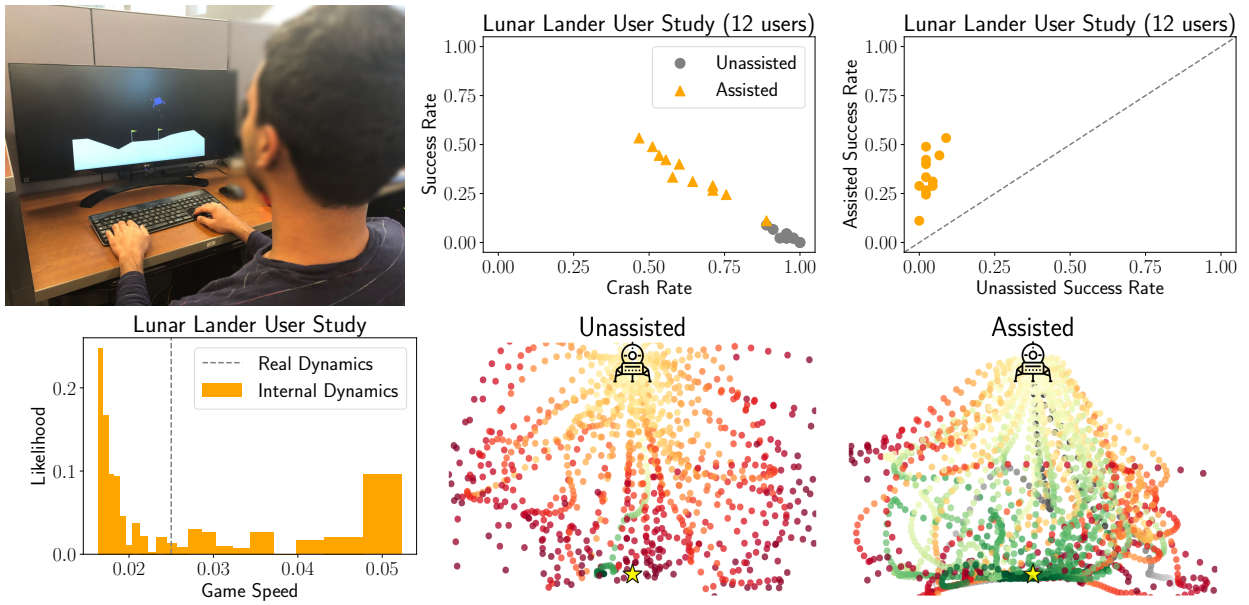

Figure 3.3: Human users find the default game environment—the real dynamics—to be difficult and unintuitive, as indicated by their poor performance in the unassisted condition (top center and right plots) and their subjective evaluations (in Table 3.1). Our method observes suboptimal human play in the default environment, learns a setting of the game physics under which the observed human play would have been closer to optimal, then performs internal-to-real dynamics transfer to assist human users in achieving higher success rates and lower crash rates (top center and right plots). The learned internal dynamics has a slower game speed than the real dynamics (bottom left plot). The bottom center and right plots show successful (green) and failed (red) trajectories in the unassisted and assisted conditions.

dynamics transfer assistance using an internal dynamics model trained on their unassisted demonstrations. The dependent measures are the success and crash rates in each condition. The task and evaluation protocol are discussed further in Section 6.2 of the appendix.

Analysis. In the default environment, users appear to play as though they underestimate the strength of gravity, which causes them to crash into the ground frequently (see the supplementary videos). Figure 3.3 (bottom left plot) shows that our algorithm learns an internal dynamics model characterized by a slower game speed than the real dynamics, which makes sense since a slower game speed induces smaller forces and slower motion—conditions under which the users' action demonstrations would have been closer to optimal. These results support our claim that our algorithm can learn an internal dynamics model that explains user actions better than the real dynamics.

When unassisted, users often crash or fly out of bounds due to the unintuitive nature of the thruster controls and the relatively fast pace of the game. Figure 3.3 (top center and right plots) shows that users succeed significantly more often and crash significantly less often when assisted by internal-to-real dynamics transfer (see Section 6.2 of the appendix for hypothesis

Table 3.1: Subjective evaluations of the Lunar Lander user study from 12 participants. Means reported below for responses on a 7-point Likert scale, where 1 = Strongly Disagree, 4 = Neither Disagree nor Agree, and 7 = Strongly Agree. p-values from a one-way repeated measures ANOVA with the presence of assistance as a factor influencing responses.

	p-value	Unassisted	Assisted
I enjoyed playing the game	$< .001$	3.92	**5.92**
I improved over time	$< .0001$	3.08	**5.83**
I didn't crash	$< .001$	1.17	**3.00**
I didn't fly out of bounds	$< .05$	1.67	**3.08**
I didn't run out of time	$> .05$	5.17	6.17
I landed between the flags	$< .001$	1.92	**4.00**
I understood how to complete the task	$< .05$	6.42	**6.75**
I intuitively understood the physics of the game	$< .01$	4.58	**6.00**
My actions were carried out	$> .05$	4.83	5.50
My intended actions were carried out	$< .01$	2.75	**5.25**

tests). The assistance makes the system feel easier to control (see the subjective evaluations in Table 3.1), less likely to tip over (see the supplementary videos), and move more slowly in response to user actions (assistance led to a 30% decrease in average speed). One of the key advantages of assistance was its positive effect on the rate at which users were able to switch between different actions: on average, unassisted users performed 18 actions per minute (APM), while assisted users performed 84 APM. Quickly switching between firing various thrusters enabled assisted users to better stabilize flight. These results demonstrate that the learned internal dynamics can be used to effectively assist the user through internal-to-real dynamics transfer, which in turn gives us confidence in the accuracy of the learned internal dynamics. After all, we cannot measure the accuracy of the learned internal dynamics by comparing it to the ground truth internal dynamics, which is unknown for human users.

3.6 Related Work

The closest prior work in intent inference and action understanding comes from inverse planning [7] and inverse reinforcement learning [80], which use observations of a user's actions to estimate the user's goal or reward function. We take a fundamentally different approach to intent inference: using action observations to estimate the user's beliefs about the world dynamics.

The simultaneous estimation of rewards and dynamics (SERD) instantiation of Max-CausalEnt IRL [50] aims to improve the sample efficiency of IRL by forcing the learned real dynamics model to explain observed state transitions as well as actions. The framework includes terms for the demonstrator's beliefs of the dynamics, but the overall algorithm and experiments of Herman et al. [50] constrain those beliefs to be the same as the real

dynamics. Our goal is to learn an internal dynamics model that may deviate from the real dynamics. To this end, we propose two new internal dynamics regularization techniques, multi-task training and the action intent prior (see Section 3.3), and demonstrate their utility for learning an internal dynamics model that differs from the real dynamics (see Section 3.5). We also conduct a user experiment that shows human actions in a game environment can be better explained by a learned internal dynamics model than by the real dynamics, and that augmenting user control with internal-to-real dynamics transfer results in improved game play. Furthermore, the SERD algorithm is well-suited to MDPs with a discrete state space, but intractable for continuous state spaces. Our method can be applied to MDPs with a continuous state space, as shown in Sections 3.5 and 3.5.

Golub et al. [38] propose an internal model estimation (IME) framework for brain-machine interface (BMI) control that learns an internal dynamics model from control demonstrations on tasks with linear-Gaussian dynamics and quadratic reward functions. Our work is (1) more general in that it places no restrictions on the functional form of the dynamics or the reward function, and (2) does not assume sensory feedback delay, which is the fundamental premise of using IME for BMI control.

Rafferty et al. [92, 90, 91] use an internal dynamics learning algorithm to infer a student's incorrect beliefs in online learning settings like educational games, and leverage the inferred beliefs to generate personalized hints and feedback. Our algorithm is more general in that it is capable of learning continuous parameters of the internal dynamics, whereas the cited work is only capable of identifying the internal dynamics given a discrete set of candidate models.

Modeling human error has a rich history in the behavioral sciences. Procrastination and other time-inconsistent human behaviors have been characterized as rational with respect to a cost model that discounts the cost of future action relative to that of immediate action [2, 60]. Systematic errors in human predictions about the future have been partially explained by cognitive biases like the availability heuristic and regression to the mean [108]. Imperfect intuitive physics judgments have been characterized as approximate probabilistic inferences made by a resource-bounded observer [46]. We take an orthogonal approach in which we assume that suboptimal behavior is primarily caused by incorrect beliefs of the dynamics, rather than uncertainty or biases in planning and judgment.

Humans are resource-bounded agents that must take into account the computational cost of their planning algorithm when selecting actions [41]. One way to trade-off the ability to find high-value actions for lower computational cost is to plan using a simplified, low-dimensional model of the dynamics [49, 34]. Evidence from the cognitive science literature suggests humans find it difficult to predict the motion of objects when multiple information dimensions are involved [88]. Thus, we arrive at an alternative explanation for why humans may behave near-optimally with respect to a dynamics model that differs from the real dynamics: even if users have perfect knowledge of the real dynamics, they may not have the computational resources to plan under the real dynamics, and instead choose to plan using a simplified model.

3.7 Discussion

Limitations. Although our algorithm models the soft Q function with arbitrary neural network parameterizations, the internal dynamics parameterizations we use are smaller, with at most seven parameters for continuous tasks. Increasing the number of dynamics parameters would require a better approach to regularization than those proposed in Section 3.3.

Summary. We contribute an algorithm that learns a user's implicit beliefs about the dynamics of the environment from demonstrations of their suboptimal behavior in the real environment. Simulation experiments and a small-scale user study demonstrate the effectiveness of our method at recovering a dynamics model that explains human actions, as well as its utility for applications in shared autonomy and inverse reinforcement learning.

Future work. The ability to learn internal dynamics models from demonstrations opens the door to new directions of scientific inquiry, like estimating young children's intuitive theories of physics and psychology without eliciting verbal judgments [114, 32, 39]. It also enables applications that involve intent inference, including adaptive brain-computer interfaces for prosthetic limbs [19, 104] that help users perform control tasks that are difficult to fully specify.

Chapter 4

Shared Autonomy via Deep Reinforcement Learning

There are times in every commander's life when he must yield the stick of authority to a subordinate. Sometimes the reason is one of expertise, when the subordinate has skills the commander lacks. Sometimes it is positional, when the subordinate is in the right place at the right [time] and the commander is not. Often it is anticipated there will be loss of direct communication, which means the subordinate may be given general instructions but must then carry them out on his own initiative as the situation flows around him. No commander enjoys those moments. Most subordinates fear them, as well. Those who do not fear already betray the overconfidence that nearly always leads to disaster. But the moments must be faced. And all will learn from them, whether to satisfaction or to sorrow.

—Timothy Zahn, *Thrawn (2017)*

In shared autonomy, user input is combined with semi-autonomous control to achieve a common goal. The goal is often unknown ex-ante, so prior work enables agents to infer the goal from user input and assist with the task. Such methods tend to assume some combination of knowledge of the dynamics of the environment, the user's policy given their goal, and the set of possible goals the user might target, which limits their application to real-world scenarios.

In this chapter, we propose a deep reinforcement learning framework for model-free shared autonomy that lifts these assumptions. We use human-in-the-loop reinforcement learning with neural network function approximation to learn an end-to-end mapping from environmental observation and user input to agent action values, with task reward as the only form of supervision. This approach poses the challenge of following user commands closely enough to provide the user with real-time action feedback and thereby ensure high-quality user input, but also deviating from the user's actions when they are suboptimal. We balance these two needs by discarding actions whose values fall below some threshold, then selecting the remaining action closest to the user's input.

Controlled studies with users ($n = 12$) and synthetic pilots playing a video game, and a pilot study with users ($n = 4$) flying a real quadrotor, demonstrate the ability of our algorithm to assist users with real-time control tasks in which the agent cannot directly access the user's private information through observations, but receives a reward signal and user input that both depend on the user's intent. The agent learns to assist the user without access to this private information, implicitly inferring it from the user's input. This enables the assisted user to complete the task more effectively than the user or an autonomous agent could on their own.

4.1 Introduction

Imagine the task of flying a quadrotor to a safe landing site. This problem is challenging for both humans and robots, but in different ways. For a human, controlling many degrees of freedom at once while dealing with unfamiliar quadrotor dynamics is hard. For a robot, understanding what makes a good landing location can be difficult, especially when the human has a future task in mind that might influence where they want the quadrotor to land now.

Shared autonomy [37, 1] aims to address this problem by combining user input with automated assistance. We focus on an area of shared autonomy in which information about the user's intent is hidden from the robot, in which prior work [77, 55, 85, 64, 47] has proposed approaches that infer the user's goal from their input and autonomously act to achieve it. These approaches tend to assume (1) a known dynamics model of the world, (2) a known goal representation (a set of possible goals), and (3) a known user policy given a goal.

For many real-world tasks, these assumptions constrain the adaptability and generality of the system. (1) Fitting an accurate global dynamics model can be more difficult than learning to perform the task. (2) Assuming a fixed representation of the user's goal (e.g., a discrete

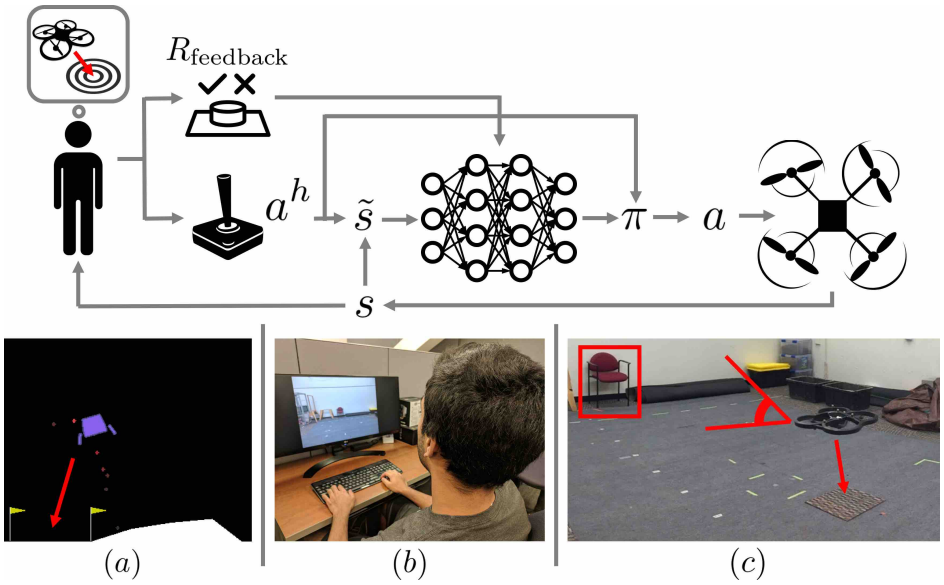

Figure 4.1: An overview of our method for assisting humans with real-time control tasks using model-free shared autonomy and deep reinforcement learning. We empirically evaluate our method on simulated pilots and real users playing the Lunar Lander game (a) and flying a quadrotor (b,c).

set of graspable objects) reduces the flexibility of the system to perform tasks in which the users' desires are difficult to specify but easy to evaluate (e.g., goal regions, or success defined directly on raw pixel input). (3) User input can exhibit systematic suboptimality that prevents standard goal inference algorithms from recovering user intent by inverting a generative model of behavior.

Our goal is to devise a shared autonomy method that lifts these assumptions, and our primary contribution is a model-free deep reinforcement learning algorithm for shared autonomy that represents a step in this direction. The key idea is that training an end-to-end mapping from environmental observation and user input to agent action values, with task reward as the only form of supervision, removes the need for known dynamics, a particular goal representation, and even a user behavior model. From the agent's perspective, the user acts like a prior policy that can be fine-tuned, and an additional sensor generating observations from which the agent can implicitly decode the user's private information. From the user's perspective, the agent behaves like an adaptive interface that learns a personalized mapping from user commands to actions that maximizes task reward.

One of the core challenges in this work lies in adapting standard deep reinforcement learning techniques to leverage input from a human without significantly interfering in their real-time 'feedback control loop'—the user's ability to observe the consequences of their own actions, and adjust their inputs accordingly. Consistently ignoring the user's input can prevent them from using action feedback to improve the quality of their input. To address this issue, we use human-in-the-loop deep Q-learning to learn an approximate state-action

value function that computes the expected future return of an action given the current environmental observation and the user's control input. Rather than taking the highest-value action, our assistive agent executes the closest high-value action to the user's input, balancing the need to take optimal actions with the need to preserve the user's feedback control loop. This approach also enables the user to directly modulate the level of assistance through the parameter $\alpha \in [0, 1]$, which sets the threshold of the system's tolerance for suboptimal user actions.

Standard deep reinforcement learning algorithms pose another challenge for human-in-the-loop training: they typically require a large number of interactions with environment, which can be a burden on users. We approach this problem by decomposing the agent's reward function into two parts: known terms computed for every state, and a terminal reward provided by the user upon succeeding or failing at the task. This decomposition enables the system to learn efficiently from a dense reward signal that captures generally useful behaviors like not crashing, and also adapt to individual users through feedback. It also enables pretraining the agent in simulation without a user in the loop, then later fine-tuning—instead of learning from scratch—with user feedback. To further improve sample efficiency, our method is capable of incorporating inferred goals into the agent's observations when the goal space and user model are known.

We apply our method to two real-time assistive control problems: the Lunar Lander game and a quadrotor landing task (see Figure 4.1). Our studies with both human and simulated pilots suggest that our method can successfully improve pilot performance. We find that our method is capable of adapting to the unique types of suboptimality exhibited by different simulated pilots, and that by varying a hyperparameter that controls our agent's tolerance for suboptimal pilot controls, we are able to help simulated pilots who need different amounts of assistance. With human pilots, our method substantially improves task success and reduces catastrophic failure. Finally, we show that when the user policy or goal representation are known, our method can be combined with adaptations of existing techniques to exploit this knowledge.

4.2 Related Work

Robotic teleoperation. We build on shared autonomy work in which the system is initially unaware of the user's goal [40, 28, 77, 55, 85, 64, 47] and explore problem statements with unknown dynamics, unknown user policy, and unknown goal representation. The parallel autonomy [100] and outer-loop stabilization [14] frameworks approach shared-control teleoperation from a different angle: instead of predicting user intent, they minimally adjust user input to achieve safe trajectories for tasks like semi-autonomous driving. Our agent's policy of executing a near-optimal action closest to the human's suggestion is inspired by this approach. Existing work in parallel autonomy requires analytic descriptions of the environment, such as the explicit locations of road boundaries and a model of the behavior of other cars. Outer-loop stabilization requires knowledge of the user's goal. Our method

is analogous, but for environments in which we do not have a dynamics model or a goal representation.

Brain-computer interfaces. A large body of work in brain-machine interfaces uses optimal control and reinforcement learning algorithms to implement closed-loop decoder adaptation [103] for applications like prosthetic limb controllers that respond to neural signals from myoelectric sensors [86]. These algorithms typically track desired motion, whereas we focus on tasks with long-horizon goals.

Reinforcement learning with human feedback. Shared autonomy enables a semi-autonomous agent to interpret user input at test time. In contrast, human-in-the-loop reinforcement learning frameworks leverage human feedback to train autonomous agents that operate independently of the user at test time [112, 62, 63, 69]. These frameworks are applicable to settings where the agent has access to all task-relevant information (e.g., goals), but the reward function is initially unknown or training can be sped up by human guidance. We focus on the orthogonal setting where the agent does not have direct access to the information that is private to the user and relevant to the task, and will always need to leverage user input to accomplish the task; even after training. This is also the key difference between our method and inverse reinforcement learning [80] and learning from demonstration [3], which generally require user interaction during training time but not at test time.

Adaptive HCI. While the bulk of the shared autonomy research discussed here exists in the context of the robotics literature, adaptive human-computer interfaces have been explored in computer graphics for animating virtual characters using motion capture data from humans [27], in natural language processing for learning to act on natural language instructions from humans [111, 13], and in formal methods for verification of semi-autonomous systems [101]. By not assuming a known user policy, our work also enables agents to adapt to a user's style of giving input.

4.3 Background

We first recap the reinforcement learning and shared autonomy problem statements on which we build in our method.

Reinforcement Learning

Consider a Markov decision process (MDP) with states $\mathcal{S}$, actions $\mathcal{A}$, transitions $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1]$. In cases where the state is not fully observable, we can extend this definition to a partially-observable MDP (POMDP) in which there is an additional set of possible observations Ω and observation function $O : \mathcal{S} \times \Omega \to [0, 1]$. The expected future discounted return of taking action a in state s with policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is expressed by the state-action value function $Q^\pi(s, a)$, and the goal in RL is to learn a policy π^* that maximizes expected future discounted return.

One algorithm for solving this problem is Q-learning [113], which minimizes the Bellman error of the Q function,

$$Q(s,a) - \gamma \mathbb{E}_{s' \sim T(\cdot|s,a)} \left[R(s,a,s') + \max_{a' \in \mathcal{A}} Q(s',a') \right],$$

as a proxy for maximizing return. We will build on this method to implement model-free shared autonomy.

Shared Autonomy

Prior work has formalized shared autonomy as a POMDP [55]. The reward function, known to both the user and agent, depends on a goal $g \in \mathcal{G}$ known to the user but unknown to the agent. The set of candidate goals $\mathcal{G}$ is known to the agent. The user follows a goal-conditioned policy $\pi_h : \mathcal{S} \times \mathcal{G} \times \mathcal{H} \to [0,1]$ known to the agent, where $\mathcal{H}$ is the space of possible user inputs—if the user suggests actions, then $\mathcal{H} = \mathcal{A}$. The transition distribution T is known to the agent. The agent's uncertainty in the goal can be formalized as partial observability, which leads to the following POMDP: the state space $\tilde{\mathcal{S}} = \mathcal{S} \times \mathcal{G}$ is augmented with the goal, the transition distribution $\tilde{T}((s_{t+1}, g) \mid s_t, g, a_t) = T(s_{t+1} \mid s_t, a_t)$ maintains a constant goal, and the observation distribution $O(s, a^h \mid s, g) = \pi_h(a^h \mid s, g)$ is given by the user policy where $a^h \in \mathcal{H}$ is the user input. Prior work assumes the goal space $\mathcal{G}$, user policy π_h, and environment dynamics T are known ex-ante to the agent, and solves the POMDP $(\tilde{\mathcal{S}}, \mathcal{A}, \tilde{T}, \tilde{R}, \mathcal{H}, O)$ using approximate methods like hindsight optimization [55]. In the following section, we introduce a different problem statement for shared autonomy which relaxes these assumptions.

4.4 Model-Free Shared Autonomy

We will relax the standard formulation in Section 4.3 to remove first the assumptions of known dynamics and the known observation model π_h for the user's private information, and then the known set of candidate goals $\mathcal{G}$. We introduce a model-free deep reinforcement learning method, with variants that can also take advantage of a known observation model and goal space when they do exist, but still provide assistance even when they are not available.

Problem Statement

In our problem formulation, the transition T, the user's policy π_h, and the goal space $\mathcal{G}$ are no longer all necessarily known to the robot. The reward function, which still depends on the user's private information, is decomposed as:

$$R(s,a,s') = \underbrace{R_{\text{general}}(s,a,s')}_{\text{known}} + \underbrace{R_{\text{feedback}}(s,a,s')}_{\text{unknown, but observed}}. \tag{4.1}$$

This captures a structure typically present in shared autonomy: there are some terms in the reward that are known, such as the need to avoid collisions. We capture these in $R_{general}$. R_{feedback} is a user-generated feedback that depends on their private information. We do not know this function. We merely assume the robot is informed when the user provides feedback (e.g., by pressing a button). In practice, the user might simply indicate once per trial whether the robot succeeded or not.

Known-User-Policy: Unknown dynamics, known goal space and user policy. In this setting, the transition T is unknown, but we have access to both $\mathcal{G}$ and the user's policy $\pi_h(a^h|s, g)$. Having access to $\mathcal{G}$ structures R_{feedback}, which is now parameterized by the goal according to $R_{\text{feedback}}(s, a, s'; g)$, and assigns high reward when $s' = g$, and 0 otherwise, without requiring manual indication from the user. We do not know g, but having access to π_h enables us to infer g via Bayesian inference.

Known-Goal-Space: Unknown dynamics and user policy, known goal space. We also consider a version of the problem where we know $\mathcal{G}$, but do not make assumptions about the user's policy π_h. In this case, R_{feedback} is still parameterized by the goal, but we must use a classification or regression model to predict the goal from the user's actions.

Min-Assumptions: Unknown dynamics, user policy, and goal space. Most of our experiments will be concerned with this setting, where we no longer assume a goal representation. This provides us with a maximally general approach, where the user might imagine whichever goal they prefer, without the need to explicitly define the space of goals in advance. In this case, we do not know the functional form of R_{feedback}, nor do we assume any parameterization for it, we merely assume the robot can observe it (evaluate it) as it takes actions. This is typically a sparse terminal reward that signals whether the task was completed successfully, and comes from the user.

Method Overview

Our method takes observations of the environment and the user's controls or inferred goal (when available) as input, and produces a high value action or control output that is as close as possible to the user's control. We learn state-action values via Q-learning with neural network function approximation. In this section, we will describe how the agent combines user input with environmental observations, motivate and describe our choice of deep Q-learning for training the agent, and describe how the agent shares control with the user.

Incorporating User Control

Because we do not know dynamics in any of our problems of interest, we use a deep reinforcement learning agent which maps observations from its sensors to actions (or Q values for each action). We incorporate information from the user as useful observations for the agent. Our method jointly embeds the agent's observation of the environment s_t with the information from the user u_t by simply concatenating them. The particular form of u_t

depends on the information that is available. Formally,

$$\tilde{s}_t = \begin{bmatrix} s_t \\ u_t \end{bmatrix}. \tag{4.2}$$

When we do not know $\mathcal{G}$, we use the user's actions a_t^h as u_t. When we know more about the possible user goals and policy, we set u_t to the inferred goal $\hat{g}_t$.

Known-User-Policy: Incorporating user control via Bayesian goal inference. When the user's policy is available, it can be used to infer the maximum a posteriori estimate of the goal $\hat{g}_t$. We can instantiate Bayesian goal inference by using maximum entropy inverse reinforcement learning [120] with a goal-parameterized Q function trained via Q-learning separately from our agent, analogously to prior work [55]. Each time step produces a better estimate of the goal $\hat{g}_t$, as additional actions reveal more about the user's intent.

Known-Goal-Space: Incorporating user control via supervised goal prediction. When we do not have a convenient model of the user's policy, we can use supervised prediction to compute the goal estimate $\hat{g}_t$. In this case, we use a separate recurrent LSTM network to predict the goal, conditioned on the sequence of states and user controls observed up to the current time t. Training data is collected from the user. As before, we concatenate $\hat{g}_t$ with the agent's observation of the environment s_t to get the combined observation $\tilde{s}_t$.

Min-Assumptions: Incorporating user control via raw action embedding. In this setting, which we use in the majority of our experiments, we do not use any explicit goal inference. Instead, the policy directly takes in the user's actions a_t^h and must learn to implicitly decode the user's intent and perform the task.[1] To our agent, the user is part of the external environment, and the user's control is yet another source of observations, much like the output of any of the agent's other sensors. Because deep neural networks are end-to-end trainable, our agent can discover arbitrary relationships between user controls and observations of the physical environment, rather than explicitly assuming the existence of a goal. Our method jointly embeds the agent's observation of the environment s_t with the user's control input a_t^h by simply concatenating them, henceforth referred to as "raw action embedding." In this setting, we set $u_t = a_t^h$.

Q-Learning with User Control

Model-free reinforcement learning with a human in the loop poses two challenges: (1) maintaining informative user input and (2) minimizing the number of interactions with the environment. (1) If the user input is a suggested control, consistently ignoring the suggestion and taking a different action can degrade the quality of user input, since humans rely on feedback from their actions to perform real-time control tasks [65]. Additionally, some user policies may already be approximately optimal and only require fine-tuning. (2) Many model-free reinforcement learning algorithms require a large number of interactions with

[1] In principle, the user's past actions are also informative of intent, and a recurrent policy could effectively integrate these. In practice, we found a reactive policy to be more effective for our tasks.

the environment, which may be impractical for human users. To mitigate these two issues, we use deep Q-learning [113] to learn an approximate state-action value function that can be used to select and evaluate actions. Specifically, we implement neural fitted Q-iteration (NFQI) [96] with experience replay [68], a periodically updated target network [74], and double Q-learning [109]. This gets around a practical problem with using vanilla deep Q-networks (DQN) [74] for human-in-the-loop learning: DQN performs a gradient update after each step, which can cause the task interface to lag and disrupts human control, whereas NFQI only performs gradient updates at the end of each episode. We chose Q-learning because (a) it is an off-policy algorithm, so we do not need to exactly follow the agent's policy and can explicitly trade off control between the user and agent, and (b) off-policy Q-learning tends to be more sample-efficient than policy gradient and Monte Carlo value-based methods [51].

Control Sharing

Motivated by the discussion of (1) and (a) in the previous section, we use the following behavior policy to select actions during and after Q-learning: select a feasible action closest to the user's suggestion, where an action is feasible if it isn't that much worse than the optimal action. Formally,

$$\pi_\alpha(a \mid \tilde{s}, a^h) = \delta\left(a = \underset{\{a: Q'(\tilde{s},a) \geq (1-\alpha) Q'(\tilde{s},a^*)\}}{\arg\max} f(a, a^h)\right), \tag{4.3}$$

where f is an action-similarity function and $Q'(\tilde{s}, a) = Q(\tilde{s}, a) - \min_{a' \in \mathcal{A}} Q(\tilde{s}, a')$ maintains a sane comparison for negative Q values: if $Q(\tilde{s}, a) < 0 \ \forall a$ and $0 < \alpha < 1$, then the set of feasible actions would be empty if we didn't subtract a baseline from the Q values. The constant $\alpha \in [0, 1]$ is a hyperparameter that controls the tolerance of the system to suboptimal human suggestions, or equivalently, the amount of assistance. The functional form of the action feasibility condition is motivated by the fact that it is invariant to affine scaling of Q values. The overall algorithm is summarized in Algorithm 2.

4.5 Simulation Experiments

We begin our experiments with an analysis of our method under different simulated users. To simplify terminology, we henceforth refer to the user as the *pilot* and the semi-autonomous agent as the *copilot*. Our central hypothesis is that our method can improve a pilot's performance despite not knowing the world's dynamics and the pilot's policy, or assuming a particular set of goals. Simulating pilots enables us to take a deeper dive into different aspects of our method (like the effects of the tolerance parameter α, and of training and testing on different types of input) before testing on real users—after all, simulated pilots do not run out of patience.

The Lunar Lander System. We use the Lunar Lander game from OpenAI Gym [15] (see the bottom-left panel of Figure 4.1) as our test platform for this part of our experiments.

Algorithm 2 Human-in-the-loop deep Q-learning

Initialize experience replay memory $\mathcal{D}$ to capacity N
Initialize Q-function with random or pretrained weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode $= 1, M$ **do**
  for $t = 1, T$ **do**
    Sample action $a_t \sim \pi_\alpha(a_t \mid \tilde{s}_t, a_t^h)$ using equation 4.3
    Execute action a_t and observe $(\tilde{s}_{t+1}, a_{t+1}^h, r_t)$
    Store transition $(\tilde{s}_t, a_t, r_t, \tilde{s}_{t+1})$ in $\mathcal{D}$
    if $\tilde{s}_{t+1}$ is terminal **then**
      for $k = 1$ to K **do** ▷ training loop
        Sample minibatch $(\tilde{s}_j, a_j, r_j, \tilde{s}_{j+1})$ from $\mathcal{D}$
        $y_j = r_j + \gamma \hat{Q}(\tilde{s}_{j+1}, \arg\max_{a'} Q(\tilde{s}_{j+1}, a'; \theta); \theta^-)$
        $\theta \leftarrow \theta - \eta \nabla_\theta \sum_j (y_j - Q(\tilde{s}_j, a_j; \theta))^2$
    Every C steps reset $\hat{Q} = Q$

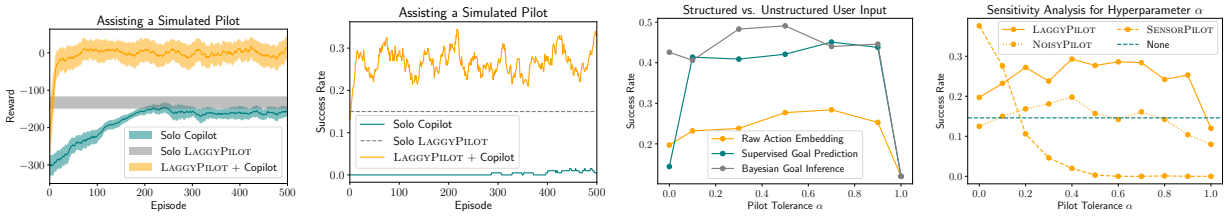

Figure 4.2: (1,2) A copilot that leverages input from the synthetic LaggyPilot outperforms the solo LaggyPilot and solo copilot. The colored bands illustrate the standard error of rewards and success rates for ten different random seeds. Rewards and success rates are smoothed using a moving average with a window size of 20 episodes. (3) The benefit of using Bayesian goal inference or supervised goal prediction depends on α. Each success rate is averaged over ten different random seeds and the last 100 episodes of training. (4) The effect of varying α depends on the user model. Each success rate is averaged over ten different random seeds and the last 100 episodes of training.

The objective of the game is to pilot the lunar lander vehicle to a specified landing site on the ground without crashing using two lateral thrusters and a main engine. Each episode lasts at most 1000 steps, and runs at 50 frames per second. An episode ends when the lander crashes, flies out of bounds, remains stationary on the ground, or time runs out. The action space $\mathcal{A}$ consists of six discrete actions that correspond to the {left, right, off} steering commands and {on, off} main engine settings. The state $s \in \mathbb{R}^8$ is an eight-dimensional vector that encodes the lander's position, velocity, angle, angular velocity, and indicators for contact between the legs of the vehicle and the ground. The x-coordinate of the landing site is selected uniformly at random at the beginning of each episode, and is not directly accessible to the agent through the state s. A human playing the game can see two flags demarcating the landing site, and can supply a suggested control $a^h \in \mathcal{A}$—depending on the user policy,

Table 4.1: Evaluation of simulated pilot-copilot teams on Lunar Lander: on ten different random seeds and the last 100 episodes of copilot training for teams with a copilot; on 100 episodes for teams without a copilot.

	Without Copilot		With Copilot			
Pilot	Success Rate	Crash Rate	Success Rate	Crash Rate	Training Episodes	α
None	-	-	0.026	0.156	742	0.0
Sensor	0.000	1.000	0.060	0.650	800	0.2
Laggy	0.150	0.750	0.287	0.186	236	0.8
Noisy	0.150	0.700	0.240	0.160	604	0.5
Optimal	0.720	0.030	-	-	-	-

a^h could be an approximately-optimal action, a signal that encodes the relative direction of the landing site, etc. Thus, in order to perform the task, the agent needs to leverage a^h to maneuver toward the landing site.

The agent uses a multi-layer perceptron with two hidden layers of 64 units each to approximate the Q function $\hat{Q} : \mathcal{S} \times \mathcal{A}^2 \to \mathbb{R}$. The action-similarity function $f(a, a^h)$ in the agent's behavior policy counts the number of dimensions in which actions a and a^h agree (e.g., $f((\text{left}, \text{on}), (\text{left}, \text{off})) = 1$). As discussed earlier in Section 4.4, the agent's reward function is composed of a hard-coded function R_{general} and a user-generated signal R_{feedback}. R_{general} penalizes speed and tilt, since moving fast and tipping over are generally dangerous for any pilot regardless of their intent. R_{feedback} emits a large positive reward at the end of the episode if the vehicle successfully lands at the intended site, or a large negative reward if it crashes or goes out of bounds.

Testing Unstructured Copilot Performance

We now test the central hypothesis that our method, model-free shared autonomy, improves a pilot's performance. We do this first in the Min-Assumptions setting, where the dynamics, user policy, and goal space are all unknown. We then test our ability to leverage this information when it exists in the next section.

Manipulated variables. We manipulate (1) the operator team composition: a solo pilot, a solo copilot, or our method—a pilot assisted by a copilot; and (2) the policy followed by the simulated pilot—a categorical variable that can take on four values: None (always executes a noop), LaggyPilot, NoisyPilot, and SensorPilot.

LaggyPilot is an optimal pilot except that it can't change actions quickly, which for a real human might be the result of poor reaction time. The LaggyPilot policy is trained as follows: augment the state vector with the landing site coordinates, train a reinforcement learning agent using vanilla DQN, and corrupt the trained policy by forcing it to repeat the previously executed action with fixed probability $p = 0.85$. This causes each action to repeat for a number of steps that follows a geometric distribution.

NoisyPilot is an optimal pilot except that it occasionally takes the wrong action, which for a real human might be the result of mistakenly pressing the wrong key. It uses the same

training procedure as LaggyPilot but follows an ϵ-greedy behavior policy at test time ($\epsilon = 0.3$).

SensorPilot tries to move toward the landing site by firing the appropriate lateral thruster, but is oblivious to gravity and doesn't use the main engine; these actions provide enough signal for an assistive copilot to deduce the location of the landing site, which may be all the human is willing to do.

Dependent measures. We measure reward, success rate, and crash rate.

Hypothesis. We hypothesize that a pilot-copilot team with a simulated pilot will perform better on the Lunar Lander game than a solo pilot or solo copilot.

Analysis. The results in Figure 4.2 (first two plots) show that a copilot which leverages input from LaggyPilot outperforms the solo LaggyPilot and solo copilot: the combined pilot-copilot team crashes and goes out of bounds less often, uses less fuel, follows stabler trajectories, and finds the landing site more often than the other two solo teams. The solo copilot and combined pilot-copilot teams learn from experience, whereas the solo LaggyPilot is pretrained and frozen; hence the stationarity of the gray curve. Table 4.1 shows that NoisyPilot and SensorPilot also benefit from assistance, although SensorPilot's success rate does not substantially increase.

To measure the sensitivity of the copilot's performance to the pilot tolerance hyperparameter α (recall Equation 4.3), we sweep different values of α while shaping the reward R_{feedback} to improve the performance of SensorPilot. The results in Figure 4.2 (bottom right) show the effects of varying α for different simulated pilot models: $\alpha = 0$ is optimal for SensorPilot, and $\alpha \approx 0.5$ is optimal for LaggyPilot and NoisyPilot.

The Benefit of Structure when Structure Exists

In some tasks, the user's private information will indeed be a goal, and we will indeed know the set of candidate goals and the policy that user follows given a goal. In this section, we show the adaptions of our method for Known-Goal-Space and Known-User-Policy from Section 4.4 can effectively leverage this information when it exists.

Manipulated variables. We manipulate (1) the input decoding mechanism—a categorical variable that can take on three values: Bayesian goal inference, supervised goal prediction, and raw action embedding; and (2) the pilot tolerance $\alpha \in [0, 1]$—a continuous variable sampled uniformly across the unit interval.

Hypothesis. We hypothesize that a copilot that uses Bayesian goal inference or supervised goal prediction to interpret user control inputs from LaggyPilot will outperform a copilot that uses raw action embedding.

Analysis. The results in Figure 4.2 (third plot) show that when the goal space and user model are known, Bayesian goal inference and supervised goal prediction outperform raw action embedding. Bayesian goal inference enables much better assistance when the user model is approximately correct: LaggyPilot behaves similarly enough to an optimal pilot that maximum entropy inverse reinforcement learning generates high-accuracy estimates of the landing site. As a result, Bayesian goal inference performs better than supervised

Table 4.2: Training and testing with different pilots on Lunar Lander. Success rates shown for 100 episodes.

	Evaluation Pilot			
Training Pilot	None	Sensor	Laggy	Noisy
None	0.02	0.02	0.29	0.04
Sensor	0.18	0.46	0.31	0.23
Laggy	0.00	0.00	0.31	0.23
Noisy	0.10	0.10	0.38	0.21

goal prediction and raw action embedding on LaggyPilot. We conclude that when an approximately correct user model is available, one should take advantage of it by using Bayesian goal inference instead of supervised goal prediction or raw action embedding. When the user model is unknown ex-ante, then one should use supervised goal prediction instead of raw action embedding.

Adapting to Diverse Users

Next, we investigate to what extent the copilot's learned policy is adapted to the pilot it assists at training time. User-specific adaptation is important because it would enable our method to generalize to tasks in which users display a range of behavior policies with distinct types of errors that cannot simultaneously be corrected by a general assistance feature.

Manipulated variables. We manipulate (1) the policy followed by the simulated pilot used to train the copilot and (2) the policy followed by the simulated pilot used to evaluate the copilot—both categorical variables that can each take on four values: None (always executes a noop), SensorPilot, LaggyPilot, and NoisyPilot.

Hypothesis. We hypothesize that the copilot learns an assistive policy that is personalized to the individual user, and that a copilot trained with one type of simulated pilot will perform better if evaluated with the same type of pilot than with a different pilot.

Analysis. The results in Table 4.2 hint that the copilot trained to assist SensorPilot acquires a relatively unique assistive policy, and that assisting SensorPilot requires something qualitatively different than assisting other pilots. A copilot trained with SensorPilot does not help other simulated pilots as well as it helps SensorPilot. Copilots trained with non-SensorPilot pilots do not assist SensorPilot as well as a copilot trained with SensorPilot. In contrast, a copilot evaluated with LaggyPilot or NoisyPilot performs equally well when trained with either of those two pilots. These results may be explained by the fact that SensorPilot implements a goal-signaling policy that is qualitatively distinct from LaggyPilot and NoisyPilot, which both implement policies based on perturbations of an optimal pilot.

Another takeaway from Table 4.2 is that a copilot trained without a pilot learns an assistive policy that is just as effective at helping LaggyPilot as a copilot policy trained with LaggyPilot in the loop. This suggests that the copilot can still learn useful assistive

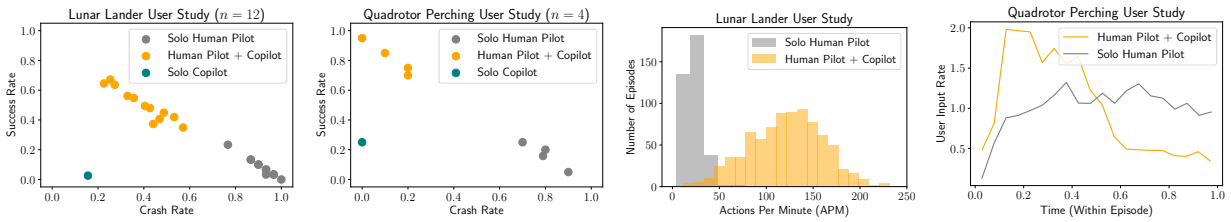

Figure 4.3: (1) Evaluation of real humans on Lunar Lander. Success and crash rates averaged over 30 episodes for teams with human pilots. (2) Evaluation of real humans on the quadrotor perching task. Success and crash rates averaged over 20 episodes. (3) Pilot-copilot teams in the Lunar Lander game are able to switch between actions more quickly than solo human pilots, which enables them to better stabilize flight. (4) On their own, users tend to provide input at a constant rate throughout an episode. When assisted by a copilot, users initially rotate the drone to orient the camera at the target object, then defer to the copilot to fly to the landing pad.

behaviors even when there is no pilot in the loop during training. Furthermore, it may enable us to save human pilots time by pretraining the copilot without the human pilot, then fine-tuning the pretrained copilot with the human pilot.

4.6 User Study with a Game Agent

We saw in simulation that our method can improve the performance of different kinds of pilots. Next, we test whether it can actually help real people in a teleoperation task.

Manipulated variables. We manipulated the team structure: solo copilot, solo human pilot, and our method—human pilot with a copilot. We use the same Lunar Lander environment for this part of the experiment. R_{feedback} is a terminal reward as before.

Dependent measures. Our objective measures are success rate and crash rate. We additionally introduce some informal subjective measures, where in each condition we ask participants about their experience, to help us understand their perception of the copilot.

Hypothesis. We hypothesize that a pilot-copilot team with a real human pilot will perform better than a solo human pilot or solo copilot.

Subject allocation. We recruited 11 male and 1 female participants, with an average age of 24. Each participant was provided with the rules of the game and a short practice period of 20 episodes to familiarize themselves with the controls and dynamics. To avoid the confounding effect of humans learning to play the game better over time, we counterbalanced the order of the two conditions that required human play (solo pilot, and assisted pilot). Each condition lasted 30 episodes.

To speed up learning, the copilot was pretrained without a pilot in the loop then fine-tuned on data collected from the human pilot. Pilot tolerance $\alpha = 0.6$ was chosen heuristically to match the difficulty of the game and the average human user's skill level. The default game environment is too challenging for human pilots, so it was modified to make the vehicle's legs

more resistant to crashing on impact with the ground. Additionally, pilot tolerance was set to $\alpha = 0$ when the human was not pressing any keys, and an additional key was introduced to enable the user to explicitly enter a noop input with $\alpha = 0.6$.

Analysis. Figure 4.3 (first plot) shows a clear quantitative and qualitative benefit to combining a real human pilot with a copilot. Humans follow a tortuous path with sudden drops and difficult course corrections, leading to fewer successes and more crashes. With a copilot, the human follows a smooth, gradual descent to the landing site, leading to significantly more successes and significantly fewer crashes than without a copilot for each of the participants. We ran a repeated measures ANOVA with the presence of the copilot as a factor influencing success and crash rates, and found that $f(1, 11) = 165.0001, p < 0.0001$ for the success rate and $f(1, 11) = 259.9992, p < 0.0001$ for the crash rate. The combined human pilot-copilot team succeeds significantly more often than the solo copilot, at the expense of crashing significantly more often. For each of the participants, we ran a binomial test comparing their success rate and crash rate in the combined pilot-copilot team to those of the solo copilot and found that $p < 0.01$ for all comparisons.

The subjective evaluations generally suggest that users benefited from the copilot. The assistive system was particularly helpful in avoiding crashing, but perceived to be somewhat inconsistent in its behavior and too aggressive in stabilizing flight at the expense of slowing down the lander's descent.

4.7 User Study with a Physical Robot: Quadrotor Perching

One of the drawbacks of analyzing Lunar Lander is that the game interface and physics do not reflect the complexity and unpredictability of a real-world robotic shared autonomy task. To evaluate our method in a more realistic environment, we formulate a "perching" task for a real human flying a real quadrotor: land the vehicle on a level, square landing pad at some distance from the initial take-off position, such that the drone's first-person camera is pointed at a specific object in the drone's surroundings, without flying out of bounds or running out of time. Perching a drone at an arbitrary vantage point enables it to be used as a mobile security camera for surveillance applications. Humans find it challenging to simultaneously point the camera at the desired scene and navigate to the precise location of a feasible landing pad under time constraints. An assistive copilot has little trouble navigating to and landing on the landing pad, but does not know where to point the camera because it does not know what the human wants to observe after landing. Together, the human can focus on pointing the camera and the copilot can focus on landing precisely on the landing pad.

Robot task. Figure 4.1 (b, c) illustrates the experimental setup. We fly the Parrot AR-Drone 2 in an indoor flight room equipped with a Vicon motion capture system to measure the position and orientation of the drone as well as the position of the landing pad. Users are only allowed to look through the drone's first-person camera to navigate, and are blocked from

getting a third-person view of the drone. Each episode lasts at most 30 seconds. An episode begins when the drone finishes taking off. An episode ends when the drone lands, flies out of bounds, or time runs out. The action space $\mathcal{A}$ consists of 18 discrete actions that correspond to moving left, right, forward, back, descending, or hovering in place and simultaneously rotating (yawing) clockwise, counter-clockwise, or not rotating. The state $s \in \mathbb{R}^{10}$ is a ten-dimensional vector that encodes the vehicle's position, velocity, angle, angular velocity, and the horizontal components of the difference between the landing pad position and the vehicle's position. At the beginning of each episode, the starting position and orientation of the drone are randomized and the user is told that their goal is to point the camera at an object selected randomly from a set of four in the vicinity: a red chair, a gray chair, white styrofoam boards, or a door. The agent's state does not include this target orientation, which is necessary for success. Success is defined as landing on the pad (evaluated automatically using motion tracking) while orienting the camera at the correct object, which is evaluated by the human experimenter with a button press at the end of the episode. Crashing is defined as landing outside the landing pad or going out of bounds.

As before, the agent uses a multi-layer perceptron with two hidden layers of 64 units each to approximate the Q function $\hat{Q} : \mathcal{S} \times \mathcal{A}^2 \to \mathbb{R}$. The action-similarity function $f(a, a^h)$ in the agent's behavior policy counts the number of dimensions in which actions a and a^h agree (e.g., $f((\text{left}, \text{rotate clockwise}), (\text{left}, \text{rotate counter-clockwise})) = 1$). As discussed earlier in Section 4.4, the agent's reward function is composed of a hard-coded function R_{general} and a user-generated signal R_{feedback}. R_{general} penalizes distance from the landing pad, since moving toward the pad is generally useful to all pilots regardless of their desired camera orientation. R_{feedback} emits a large positive reward at the end of the episode if the task was completed successfully, or a large negative reward in the event of a crash.

Manipulated variables. We manipulate the pilot-copilot team membership as before.

Dependent measures. Performance is measured using the dependent factors of success rate and crash rate. As before, we ask participants about their experience (see Table 2 in the supplementary material) to help us understand their perception of the copilot.

Hypothesis. We hypothesize that a pilot-copilot team with a real human pilot will perform better on the quadrotor perching task than a solo human pilot or a solo copilot.

Subject allocation. We recruited 3 male and 1 female participants, with an average age of 23. Each participant was provided with the rules of the game and a short practice period of 2 episodes to familiarize themselves with the controls and dynamics. To avoid the confounding effect of humans learning to play the game better over time, we counterbalanced the order of the two conditions that required human play (solo pilot, and assisted pilot). Each condition lasted 20 episodes.

To speed up learning, the copilot was pretrained in simulation without a pilot in the loop then fine-tuned on data collected from the human pilot. The pretraining simulation assumed an idealized physics model in which the drone is a point mass, there are no external forces, linear velocity commands are executed without any noise, and sensors have zero measurement error. In the pretraining simulation, a target angle (yaw) is randomly sampled for each episode to simulate the random choice of a target object for the camera in the real world. As

before, the agent cannot directly access this target angle through its state.

With a real human pilot in the real world, pilot tolerance was set to $\alpha = 0$ when the human was not pressing any keys, and otherwise set to $\alpha = 1$.

Analysis. Figure 4.3 (second plot) shows a clear quantitative and qualitative benefit to combining a real human pilot with a copilot. Humans are rarely able to arrive at the landing pad, leading to fewer successes and more crashes. With a copilot, the human consistently gets to the landing pad, leading to significantly more successes and significantly fewer crashes than without a copilot. The sample size of $n = 4$ participants is relatively small, so the evidence is mainly anecdotal and should be interpreted in the context of the larger simulation experiments and user study on the Lunar Lander game. With that in mind, we ran a repeated measures ANOVA with the presence of the copilot as a factor influencing success and crash rates, and found that $f(1, 3) = 44.1045, p < 0.01$ for the success rate and $f(1, 3) = 62.3151, p < 0.01$ for the crash rate. The combined human pilot-copilot team succeeds significantly more often than the solo copilot, at the expense of crashing significantly more often. For each of the participants, we ran a binomial test comparing their success rate and crash rate in the combined pilot-copilot team to those of the solo copilot and found that $p < 0.01$ for all comparisons. The subjective evaluations in Table 2 of the supplementary material generally suggest that users benefited from the copilot.

4.8 Discussion

In this paper, we contribute an algorithm for shared autonomy that uses model-free reinforcement learning to help human users with tasks with unknown dynamics, user policies, and goal representations. We introduce a behavioral policy for deep Q-learning that enables users to directly control the level of assistance, as well as a decomposition of the reward function that enables the system to quickly learn generally useful behaviors and also adapt to individual users. Our user studies with a virtual agent and a real robot suggest that this method can indeed be effective at improving user performance.

Several weaknesses and open questions remain to be addressed. Inferring user intent in general will require memory. Several existing techniques may accomplish this, including concatenating m previous frames with the current observation, or adding recurrent connections to the copilot policy architecture as in [48]. Finally, users will adapt to the robot's interface, and explicitly capturing this may improve copilot training and inform theoretical guarantees on convergence [81].

Chapter 5

Conclusions and Future Work

We're each of us alone, to be sure. What can you do but hold your hand out in the dark?

—Ursula K. Le Guin, *Nine Lives (1968)*

Humans generally know what they want, but can have trouble achieving it due to systematic biases in perception and planning that lead to suboptimal behavior. Machines generally don't know what their human operators want, but are often capable of sensing and controlling complex systems without the limitations of human cognitive biases. In this thesis, we presented various methods for building assistive interfaces that combine human and machine intelligence to perform challenging sequential decision-making tasks.

The most promising area for future work on these methods is brain-computer interfaces [19, 104], which have the potential to help patients who have lost motor function regain their independence and flourish. For example, the internal-to-real dynamics transfer assistance method from Chapter 3 and the human-in-the-loop Q-learning method from Chapter 4 could be used to improve the accuracy of decoders that predict the user's desired action given the user's brain activity. Furthermore, recent work on audio-visual speech entrainment [35], haptics [25], lingual nerve stimulation [75], and intracortical stimulation of somatosensory cortex [31] suggests that improving feedback is at least as important as improving action decoding accuracy for closed-loop brain-computer interfaces. The assistive state estimation algorithm from Chapter 2 could be used to optimize observations (e.g., haptic feedback patterns) for this purpose.

Another promising direction for future research is machine theory of mind [89] and AI alignment. Prior work has shown that, in order to accurately infer human preferences from demonstration data, one must make assumptions about the rationality of human behavior [5]. However, making the wrong assumptions (e.g., assuming humans are noisily-rational when they are actually systematically biased) can lead to inaccurate estimates of the human

reward function [20]. The training methods discussed in Chapters 2 and 3 could be useful for learning a model of human irrationality from demonstration data, akin to prior work [102].

Bibliography

[1] Peter Aigner and Brenan McCarragher. "Human integration into robot control utilising potential fields". In: *IEEE International Conference on Robotics and Automation*. 1997.

[2] George A Akerlof. "Procrastination and obedience". In: *The American Economic Review* (1991).

[3] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* (2009).

[4] Stuart Armstrong and Sören Mindermann. "Impossibility of deducing preferences and rationality from human policy". In: *arXiv preprint arXiv:1712.05812* (2017).

[5] Stuart Armstrong and Sören Mindermann. "Occam's razor is insufficient to infer the preferences of irrational agents". In: *Neural Information Processing Systems* (2018).

[6] Christopher G Atkeson et al. "What happened at the DARPA robotics challenge, and why". In: *DRC Finals Special Issue of the Journal of Field Robotics* (2016).

[7] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. "Action understanding as inverse planning". In: *Cognition* (2009).

[8] Nikola Banovic et al. "Uncovering information needs for independent spatial learning for users who are visually impaired". In: *ACM SIGACCESS Conference on Computers and Accessibility*. 2013.

[9] Leon Bergen, Owain Evans, and Joshua Tenenbaum. "Learning structured preferences". In: *Annual Meeting of the Cognitive Science Society*. 2010.

[10] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. 2014.

[11] Michael Bloem and Nicholas Bambos. "Infinite time horizon maximum causal entropy inverse reinforcement learning". In: *IEEE Conference on Decision and Control*. 2014.

[12] Matthew Botvinick and James An. "Goal-directed decision making in prefrontal cortex: a computational framework". In: *Neural Information Processing Systems*. 2009.

[13] Satchuthananthavale RK Branavan et al. "Reinforcement learning for mapping instructions to actions". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Association for Computational Linguistics. 2009.

[14] Alexander Broad, Todd David Murphey, and Brenna Dee Argall. "Learning models for shared control of human-machine systems with unknown dynamics". In: *Robotics: Science and Systems*. 2017.

[15] Greg Brockman et al. "OpenAI Gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[16] Moritz Bühler and Thomas Weisswange. "Theory of Mind based Communication for Human Agent Cooperation". In: (2020).

[17] Giuseppe Calafiore and Fabrizio Dabbene. *Probabilistic and randomized methods for design under uncertainty*. 2006.

[18] Alfonso Caramazza, Michael McCloskey, and Bert Green. "Naive beliefs in "sophisticated" subjects: Misconceptions about trajectories of objects". In: *Cognition* (1981).

[19] Jose M Carmena. "Advances in neuroprosthetic learning and control". In: *PLoS biology* (2013).

[20] Lawrence Chan, Andrew Critch, and Anca Dragan. "Human irrationality: both bad and good for reward inference". In: *arXiv preprint arXiv:2111.06956* (2021).

[21] Angel Chang et al. "Matterport3D: Learning from RGB-D Data in Indoor Environments". In: *International Conference on 3D Vision* (2017).

[22] Susan E. F. Chipman. *The Oxford handbook of cognitive science*. 2016.

[23] Mark Cutler and Jonathan P How. "Efficient reinforcement learning for robots using informative simulated priors". In: *IEEE International Conference on Robotics and Automation*. 2015.

[24] Saurabh Daptardar, Paul Schrater, and Xaq Pitkow. "Inverse Rational Control with Partially Observable Continuous Nonlinear Dynamics". In: *arXiv preprint arXiv:1908.04696* (2019).

[25] Darrel R Deo et al. "Effects of Peripheral Haptic Feedback on Intracortical Brain-Computer Interface Control and Associated Sensory Responses in Motor Cortex". In: *IEEE Transactions on Haptics* (2021).

[26] Michel Desmurget and Scott Grafton. "Forward modeling allows feedback control for fast reaching movements". In: *Trends in Cognitive Sciences* (2000).

[27] Mira Dontcheva, Gary Yngve, and Zoran Popović. "Layered acting for character animation". In: *ACM Transactions on Graphics*. 2003.

[28] Anca D Dragan and Siddhartha S Srinivasa. "A policy-blending formalism for shared control". In: *The International Journal of Robotics Research* (2013).

[29] Owain Evans and Noah D Goodman. "Learning the preferences of bounded agents". In: *NIPS Workshop on Bounded Optimality*. 2015.

[30] Owain Evans, Andreas Stuhlmüller, and Noah Goodman. "Learning the preferences of ignorant, inconsistent agents". In: *AAAI Conference on Artificial Intelligence*. 2016.

[31] Sharlene N Flesher et al. "A brain-computer interface that evokes tactile sensations improves robotic arm control". In: *Science* (2021).

[32] Jerry A Fodor. "A theory of the child's theory of mind." In: *Cognition* (1992).

[33] Terrence Fong et al. "Space telerobotics: unique challenges to human-robot collaboration in space". In: *Reviews of Human Factors and Ergonomics* (2013).

[34] David Fridovich-Keil et al. "Planning, Fast and Slow: A Framework for Adaptive Real-Time Safe Trajectory Planning". In: *arXiv preprint arXiv:1710.04731* (2017).

[35] Julius Fridriksson et al. "Speech entrainment enables patients with Broca's aphasia to produce fluent speech". In: *Brain* (2012).

[36] Tobias Gerstenberg and Joshua B Tenenbaum. "Intuitive theories". In: *Oxford Handbook of Causal Reasoning* (2017).

[37] Ray C Goertz. "Manipulators used for handling radioactive materials". In: *Human factors in technology* (1963).

[38] Matthew Golub, Steven Chase, and M Yu Byron. "Learning an internal dynamics model from control demonstration". In: *International Conference on Machine Learning*. 2013.

[39] Alison Gopnik and Henry M Wellman. "The theory theory". In: *Mapping the mind: Domain specificity in cognition and culture* (1994).

[40] Scott A Green et al. "Human-robot collaboration: A literature review and augmented reality approach in design". In: *International Journal of Advanced Robotic Systems* (2008).

[41] Thomas L Griffiths, Falk Lieder, and Noah D Goodman. "Rational use of cognitive resources: Levels of analysis between the computational and the algorithmic". In: *Topics in Cognitive Science* (2015).

[42] João Guerreiro et al. "Virtual navigation for blind people: Building sequential representations of the real-world". In: *ACM SIGACCESS Conference on Computers and Accessibility*. 2017.

[43] David Ha and Jürgen Schmidhuber. "Recurrent world models facilitate policy evolution". In: *Neural Information Processing Systems*. 2018.

[44] Tuomas Haarnoja et al. "Reinforcement learning with deep energy-based policies". In: *arXiv preprint arXiv:1702.08165* (2017).

[45] Danijar Hafner et al. "Dream to control: Learning behaviors by latent imagination". In: *arXiv preprint arXiv:1912.01603* (2019).

[46] Jessica Hamrick, Peter Battaglia, and Joshua B Tenenbaum. "Internal physics models guide probabilistic judgments about object dynamics". In: *Annual Conference of the Cognitive Science Society*. 2011.

[47] Kris Hauser. "Recognition, prediction, and planning for assisted teleoperation of freeform tasks". In: *Autonomous Robots* (2013).

[48] Matthew Hausknecht and Peter Stone. "Deep recurrent Q-learning for partially observable MDPs". In: *arXiv preprint arXiv:1507.06527* (2015).

[49] Sylvia L Herbert et al. "FaSTrack: a modular framework for fast and guaranteed safe motion planning". In: *arXiv preprint arXiv:1703.07373* (2017).

[50] Michael Herman et al. "Inverse reinforcement learning with simultaneous estimation of rewards and dynamics". In: *Artificial Intelligence and Statistics*. 2016.

[51] Matteo Hessel et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *arXiv preprint arXiv:1710.02298* (2017).

[52] Sophie Hilgard et al. "Learning Representations by Humans, for Humans". In: *arXiv preprint arXiv:1905.12686* (2019).

[53] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* (1997).

[54] Daniel Jarrett and Mihaela van der Schaar. "Inverse Active Sensing: Modeling and Understanding Timely Decision-Making". In: *arXiv preprint arXiv:2006.14141* (2020).

[55] Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. "Shared autonomy via hindsight optimization". In: *arXiv preprint arXiv:1503.07619* (2015).

[56] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* (1998).

[57] Mitsuo Kawato. "Internal models for motor control and trajectory planning". In: *Current Opinion in Neurobiology* (1999).

[58] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[59] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[60] Jon Kleinberg and Sigal Oren. "Time-inconsistent planning: a computational problem in behavioral economics". In: *ACM Conference on Economics and Computation*. 2014.

[61] David C Knill and Whitman Richards. *Perception as Bayesian inference*. 1996.

[62] W Bradley Knox and Peter Stone. "Interactively shaping agents via human reinforcement: The TAMER framework". In: *ACM International Conference on Knowledge Capture*. 2009.

[63] W Bradley Knox and Peter Stone. "Reinforcement learning from simultaneous human and MDP reward". In: *International Conference on Autonomous Agents and Multiagent Systems*. 2012.

[64] Hema S Koppula and Ashutosh Saxena. "Anticipating human activities using object affordances for reactive robotic response". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2016).

[65] Michael Laskey et al. "Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations". In: *IEEE International Conference on Robotics and Automation*. 2017.

[66] Yann LeCun. *The MNIST database of handwritten digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/.

[67] Sergey Levine. "Reinforcement learning and control as probabilistic inference: Tutorial and review". In: *arXiv preprint arXiv:1805.00909* (2018).

[68] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. Tech. rep. Carnegie Mellon University, 1993.

[69] Zhiyu Lin et al. "Explore, exploit or listen: Combining human feedback and policy model to speed up deep reinforcement learning in 3D worlds". In: *arXiv preprint arXiv:1709.03969* (2017).

[70] Anirudha Majumdar et al. "Risk-sensitive inverse reinforcement learning via coherent risk models". In: *Robotics: Science and Systems*. 2017.

[71] Manolis Savva* et al. "Habitat: A Platform for Embodied AI Research". In: *IEEE/CVF International Conference on Computer Vision*. 2019.

[72] Biren Mehta and Stefan Schaal. "Forward models in visuomotor control". In: *Journal of Neurophysiology* (2002).

[73] Marvin Minsky. "Telepresence". In: *Omni* (1980).

[74] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[75] Joel Adrian Moritz Jr. "Evaluation of electrical tongue stimulation for communication of audio information to the brain". PhD thesis. Colorado State University, 2016.

[76] Ralph S Mosher. "Handyman to Hardiman". In: *SAE Transactions* (1968).

[77] Katharina Muelling et al. "Autonomy infused teleoperation with application to brain computer interface controlled manipulation". In: *Autonomous Robots* (2017).

[78] Max Mulder. "Cybernetics of tunnel-in-the-sky displays". PhD thesis. Delft University of Technology, 1999.

[79] Gergely Neu and Csaba Szepesvári. "Apprenticeship learning using inverse reinforcement learning and gradient methods". In: *arXiv preprint arXiv:1206.5264* (2012).

[80] Andrew Y Ng and Stuart J Russell. "Algorithms for inverse reinforcement learning." In: *International Conference on Machine Learning*. 2000.

[81] Stefanos Nikolaidis et al. "Human-Robot Mutual Adaptation in Shared Autonomy". In: *arXiv preprint arXiv:1701.07851* (2017).

[82] Eshed Ohn-Bar, Kris Kitani, and Chieko Asakawa. "Personalized dynamics models for adaptive assistive navigation systems". In: *arXiv preprint arXiv:1804.04118* (2018).

[83] Sabrina A Panëels et al. "Listen to it yourself! evaluating usability of what's around me? for the blind". In: *SIGCHI Conference on Human Factors in Computing Systems*. 2013.

[84] Xue Bin Peng et al. "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills". In: *arXiv preprint arXiv:1804.02717* (2018).

[85] Claudia Pérez-D'Arpino and Julie A Shah. "Fast target prediction of human reaching motion for cooperative human-robot manipulation tasks using time series classification". In: *IEEE International Conference on Robotics and Automation*. 2015.

[86] Patrick M Pilarski et al. "Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning". In: *IEEE International Conference on Rehabilitation Robotics*. 2011.

[87] David Premack and Guy Woodruff. "Does the chimpanzee have a theory of mind?" In: *Behavioral and Brain Sciences* (1978).

[88] Dennis R Proffitt and David L Gilden. "Understanding natural dynamics". In: *Journal of Experimental Psychology: Human Perception and Performance* (1989).

[89] Neil Rabinowitz et al. "Machine theory of mind". In: *International Conference on Machine Learning*. 2018.

[90] Anna N Rafferty and Thomas L Griffiths. "Diagnosing Algebra Understanding via Inverse Planning". In: ().

[91] Anna N Rafferty, Rachel Jansen, and Thomas L Griffiths. "Using Inverse Planning for Personalized Feedback". In: *International Conference on Educational Data Mining*. 2016.

[92] Anna N Rafferty, Michelle M LaMar, and Thomas L Griffiths. "Inferring learners' knowledge from their actions". In: *Cognitive Science* (2015).

[93] Deepak Ramachandran and Eyal Amir. "Bayesian inverse reinforcement learning". In: *Urbana* (2007).

[94] Siddharth Reddy, Anca D Dragan, and Sergey Levine. "Shared autonomy via deep reinforcement learning". In: *Robotics: Science and Systems*. 2018.

[95] Siddharth Reddy, Anca D. Dragan, and Sergey Levine. "Where do you think you're going?: Inferring beliefs about dynamics from behavior". In: *Neural Information Processing Systems*. 2018.

[96] Martin Riedmiller. "Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method". In: *European Conference on Machine Learning*. 2005.

[97] Daisuke Sato et al. "Navcog3: An evaluation of a smartphone-based blind indoor navigation assistant with semantic features in a large-scale environment". In: *ACM SIGACCESS Conference on Computers and Accessibility*. 2017.

[98] Jürgen Schmidhuber. "Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments". In: (1990).

[99] Felix Schmitt et al. "I see what you see: Inferring sensor and policy models of human real-world motor behavior". In: *AAAI Conference on Artificial Intelligence*. 2017.

[100] Wilko Schwarting et al. "Parallel autonomy in automated vehicles: Safe motion generation with minimal intervention". In: *IEEE International Conference on Robotics and Automation*. 2017.

[101] Sanjit A Seshia, Dorsa Sadigh, and S Shankar Sastry. "Formal methods for semi-autonomous driving". In: *ACM Annual Design Automation Conference*. 2015.

[102] Rohin Shah et al. "On the feasibility of learning, rather than assuming, human biases for reward inference". In: *International Conference on Machine Learning*. 2019.

[103] Maryam M Shanechi, Amy L Orsborn, and Jose M Carmena. "Robust brain-machine interface design using optimal feedback control modeling and adaptive point process filtering". In: *PLoS computational biology* (2016).

[104] Krishna V Shenoy and Jose M Carmena. "Combining decoder design and neural adaptation in brain-machine interfaces". In: *Neuron* (2014).

[105] Herbert A Simon. "Bounded rationality and organizational learning". In: *Organization Science* (1991).

[106] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. "Probalistic robotics". In: (2005).

[107] Emanuel Todorov. "Optimality principles in sensorimotor control". In: *Nature Neuroscience* (2004).

[108] Amos Tversky and Daniel Kahneman. "Judgment under uncertainty: Heuristics and biases". In: *Science* (1974).

[109] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. 2016.

[110] Hsueh-Cheng Wang et al. "Enabling independent navigation for visually impaired people through a wearable vision-based feedback system". In: *IEEE International Conference on Robotics and Automation*. 2017.

[111] Sida I Wang, Percy Liang, and Christopher D Manning. "Learning Language Games through Interaction". In: *arXiv preprint arXiv:1606.02447* (2016).

[112] Garrett Warnell et al. "Deep TAMER: Interactive agent shaping in high-dimensional state spaces". In: *arXiv preprint arXiv:1709.10163* (2017).

[113] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* (1992).

[114] Friedrich Wilkening and Trix Cacchione. "Children's intuitive physics". In: *The Wiley-Blackwell Handbook of Childhood Cognitive Development, Second edition* (2010).

[115] Daniel M Wolpert, Zoubin Ghahramani, and Michael I Jordan. "An internal model for sensorimotor integration". In: *Science* (1995).

[116] Arnold YS Yeung et al. "Sequential Explanations with Mental Model-Based Policies". In: *arXiv preprint arXiv:2007.09028* (2020).

[117] Marvin Zhang et al. "SOLAR: Deep structured representations for model-based reinforcement learning". In: *International Conference on Machine Learning*. 2019.

[118] Yuhang Zhao et al. "Designing and Evaluating a Customizable Head-mounted Vision Enhancement System for People with Low Vision". In: *ACM Transactions on Accessible Computing* (2019).

[119] Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. "Modeling interaction via the principle of maximum causal entropy". In: *International Conference on Machine Learning*. 2010.

[120] Brian D Ziebart et al. "Maximum Entropy Inverse Reinforcement Learning". In: *AAAI Conference on Artificial Intelligence*. 2008.

[121] Brian D Ziebart et al. "Planning-based prediction for pedestrians". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009.

Chapter 6

Appendices

6.1 Optimizing Observations to Communicate State

Simulation Experiments

One of the drawbacks of running a user study with human participants is that, while we can measure task performance, we cannot directly measure the accuracy of users' internal beliefs about the current state. Studying how ASE scales with the amount of observation delay, training data, and other factors would also require a prohibitive number of human-in-the-loop experiments. To that end, we run experiments with simulated users on indoor navigation and MNIST digit classification.

Improving Accuracy of Users' Internal Beliefs

Our third experiment seeks to answer **Q3**: can we improve the accuracy of simulated users' internal beliefs? We test this hypothesis in an indoor navigation task with a more realistic environment than the 5x5 layout from the user study: we take one floor of a 3D house from the Matterport3D dataset [21], and discretize it into a navigable 2D grid using the Habitat framework [71]. The simulated user's belief update $b_{\mathbf{H}}$ is identical to the assistant's belief update b, except that it ignores any observation that consists of more than one object. The assistant knows the simulated user's belief update $b_{\mathbf{H}}$. Note that this differs from the 5x5 experiment in Section 2.4, in which the user's belief update was not only bandwidth-constrained, but also tainted by a misspecified observation model due to the presence of unknown objects whose locations were not plotted in the user's mental map. The purpose of this experiment is not to test if ASE can learn a user model, but rather to test the accuracy of the user's induced beliefs. Appendix 6.1 describes the experimental setup in more detail.
Manipulated factors. We evaluate (1) an unassisted baseline that passively shows the ambient observation generated by the environment and (2) ASE.
Dependent measures. We measure success rate, distance to the goal at the end of the episode, number of steps taken to reach the goal, and the user's internal log-likelihood of the true state.
Analysis. Table 6.1 show that ASE substantially outperforms the unassisted and random baselines in improving the accuracy of the user's internal beliefs. ASE tends to inform the user of landmark objects that are more likely to be seen from the current state than in other states – like gym equipment, paintings, and showers – enabling the user to infer the current state more accurately. In the unassisted condition, the user tends to receive observations of objects that are common not only in the current state but also common across states – like walls, floors, and ceilings – which makes it difficult to identify the current state. This result illustrates that ASE can be used to improve situational awareness, independent of the user's desired task.

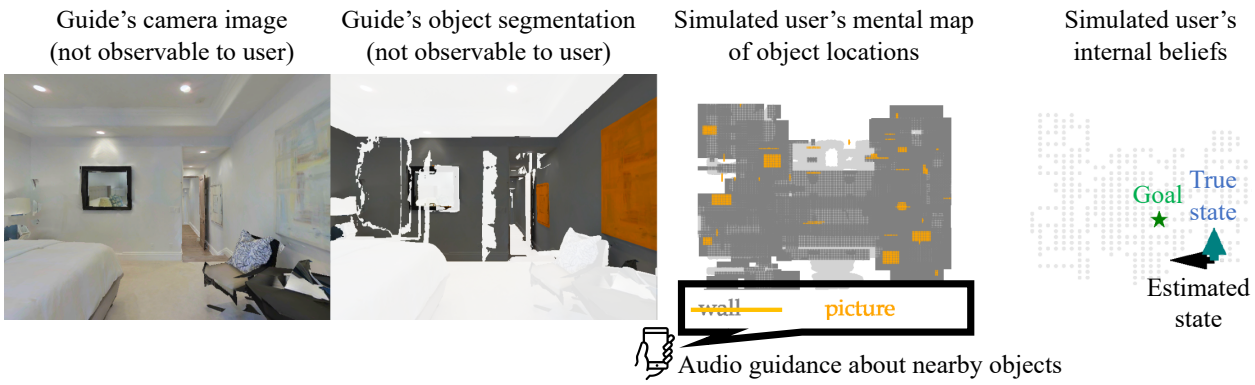

Figure 6.1: A simulated blind user navigating an indoor environment, using audio guidance about nearby objects to estimate position and orientation (the same problem setting as Figure 2.1). The assistant sees the RGB camera image, uses the semantic mesh from the dataset to determine the list of visible objects, then replaces the ambient observation (gray), which was sampled uniformly at random from the list of visible objects, with an optimized observation (orange) that minimizes KL-divergence (Equation 2.3). The simulated user knows the locations of all objects, and can use this mental map to infer their current position and orientation given observations of nearby objects and memory of past movements.

	Success Rate	Distance to Goal	Time to Goal	Belief in True State
Unassisted (Baseline)	0.73 ± 0.04	0.10 ± 0.02	70.87 ± 2.72	-1.70 ± 0.02
Random (Baseline)	0.02 ± 0.01	1.00 ± 0.04	99.99 ± 0.71	-20.44 ± 0.08
ASE (Our Method)	1.00 ± 0.00	0.00 ± 0.00	38.55 ± 1.60	-0.72 ± 0.02

Table 6.1: Habitat navigation experiments that address **Q3** – can we improve the accuracy of simulated users' internal beliefs? – by comparing our method (ASE), which synthesizes an informative observation that fits within the simulated user's sensor bandwidth, to baselines that either use ambient observations generated by the environment (Unassisted) or randomly generate observations (Random). The results show that our method (ASE) substantially outperforms the baselines (Unassisted and Random). The simulated user's internal beliefs are represented as log-likelihoods. We measure standard error across 100 evaluation episodes.

Scaling to Multivariate User Models

Our fourth experiment seeks to answer **Q4**: given enough demonstration data, can ASE learn complex models of the user's state estimation process? We test this hypothesis in the MNIST domain from Section 2.4. We simulate a user by training an LSTM sequence model to predict the image label given a sequence of pixel observations. We define the simulated user's belief update $b_{\mathbf{H}}$ using Equation 2.2, where the state encoder $f_{\mathbf{H}}$ maps a sequence of observations to a 32-dimensional hidden state. This hidden state is distinct from the hidden state produced by the assistant's state encoder f, due to the difference between the assistant's reconstruction

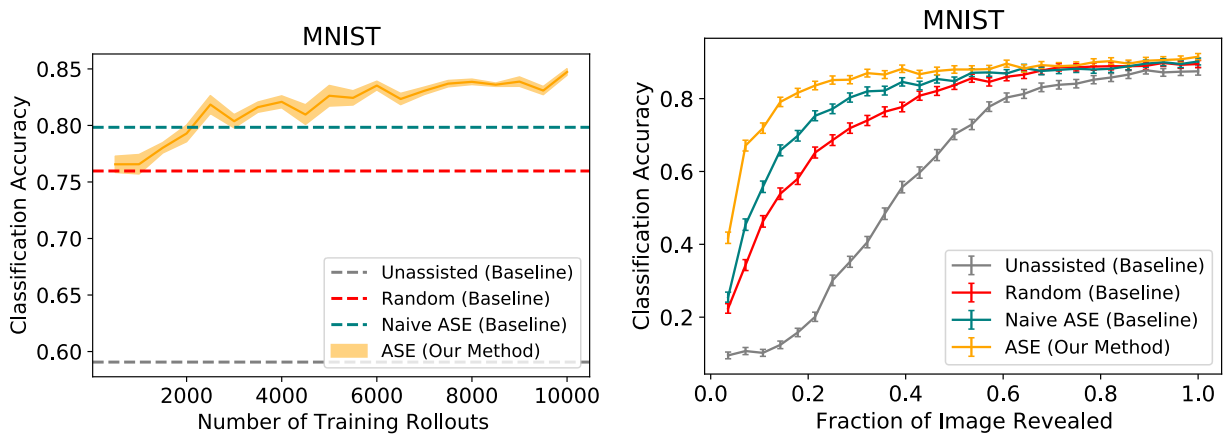

Figure 6.2: MNIST experiments that address **Q4** – given enough demonstration data, can ASE learn complex models of the user's state estimation process? – by comparing our method (ASE), which learns a model of the simulated user, to a baseline variant of our method that does not learn a model (Naïve ASE). The results show that with enough training data, the personalized assistant outperforms the naïve assistant by more accurately predicting the effect of a given observation on the simulated user, and thus providing more informative observations to the simulated user. We measure standard error across 5 random seeds and 1000 evaluation episodes.

objective (described in Section 2.4) and the simulated user's classification objective. ASE represents the user's state encoder as a recurrent neural network f_θ with 32 hidden units, and defines the user model b_θ via Equation 2.2. Appendix 6.1 describes the experimental setup in more detail.

Manipulated factors. We evaluate (1) an unassisted baseline that passively shows the ambient observation generated by the environment; (2) a naïve version of ASE that does not train the user model; and (3) ASE, where we learn θ from episodes generated iteratively using Algorithm 1. The naïve assistant incorrectly assumes the user's state encoder $f_{\mathbf{H}}$ is equivalent to the assistant's state encoder f, except that it can only process one row of pixels per timestep. We vary the number of training episodes $|\mathcal{D}|$ in the ASE condition.

Dependent measures. We measure per-timestep classification accuracy, as in Section 2.4.

Analysis. Figures 6.2 shows that with enough training episodes in $\mathcal{D}$, ASE can learn a model of the simulated user that enables it to outperform a naïve version of ASE that assumes the simulated user's belief update uses the same state encoder as the assistant's. This result demonstrates that ASE can scale to training an expressive, recurrent neural network model of the user's belief update b_θ.

Scaling to Longer Observation Delays

Our fifth experiment seeks to answer **Q5**: does ASE still improve the user's performance when observations are severely delayed? We test this hypothesis with simulated users in

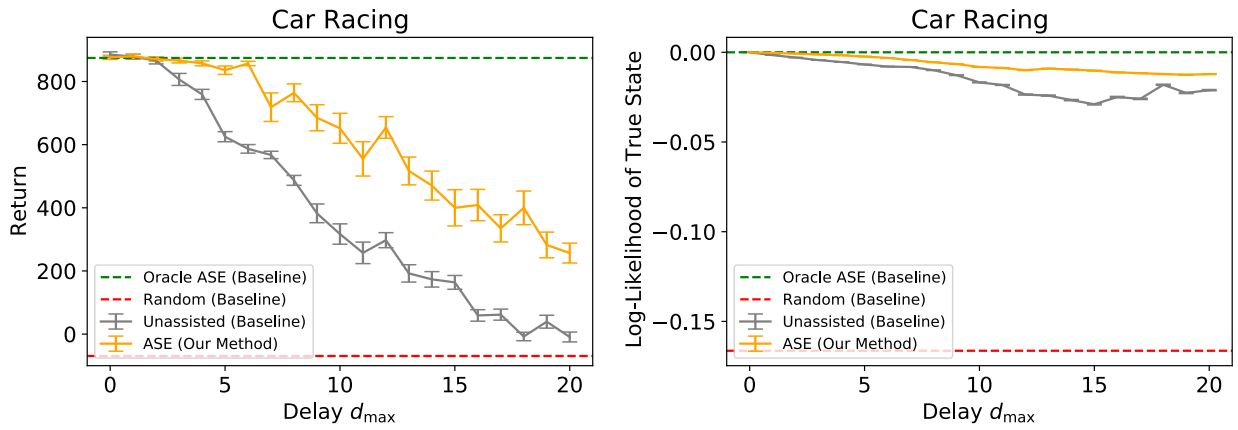

Figure 6.3: Car Racing experiments that address **Q5** – does ASE still improve the user's performance when observations are severely delayed? – by comparing our method (ASE), which tries to 'undo' the observation delay $d_{\max}$ by predicting the current state and showing the user an observation representative of the predicted current state, to baselines that either show the human the outdated ambient observation generated by the environment (Unassisted) or randomly generate observations (Random). The results show that ASE substantially improves the simulated user's task performance (left plot) and the simulated user's internal state estimation accuracy (right plot), especially when the delay $d_{\max}$ is high. We measure standard error across 20 evaluation episodes. The gap between ASE and the oracle can be attributed to imperfections in the assistant's learned dynamics model, which is used to define its state encoder f.

the Car Racing domain from Section 2.4. We simulate the user using an expert policy trained via the model-based reinforcement learning method described in [43]. In addition to manipulating the assistance condition as in Section 2.4, we also manipulate the observation delay $d_{\max} \in \{0, 1, 2, ..., 20\}$. The delay $d_{\max}$ controls the length of the no-delay and delay phases. For example, in the user study in Section 2.4, the no-delay and delay phases each lasted 5 timesteps, which corresponds to $d_{\max} = 5$. In addition to measuring the task return, we also measure the simulated user's internal log-likelihood of the true state at each timestep.

Figure 6.3 shows that ASE substantially outperforms the unassisted and random baselines (orange vs. gray and red curves) in assisting simulated users. ASE helps the simulated user by predicting the current state given outdated observations, then showing the user an observation representative of the predicted current state. These predictions are not perfect, as shown by the gap between ASE and the oracle (orange vs. green curve), but still align the simulated user's beliefs more closely with the true state. As the delay $d_{\max}$ increases, the assistant's dynamics model – which is used to define its state encoder f (see Section 2.4) – is not able to accurately predict the current state. Hence, both assisted and unassisted performance decrease as the delay increases.

Implementation Details

We use Adam [58] to perform gradient descent on the objective in Equation 2.5.
MNIST. There are $T = 28$ timesteps per episode, and the user can change their label at each timestep. Each observation consists of zero or more row indices and corresponding pixel values: $\Omega = ([28] \times \mathbb{R}^{28})^*$, where $[28] = \{1, 2, ..., 28\}$ denotes the set of row indices, and $*$ denotes the Kleene star. Let $\mathbf{I}_{1:28,1:28} \in \mathbb{R}^{28\times 28}$ denote the full image. The environment initially emits the full image observation $\mathbf{o}_0 = ((1, \mathbf{I}_{1,1}, \mathbf{I}_{1,2}, ..., \mathbf{I}_{1,28}), (2, \mathbf{I}_{2,1}, ...), ..., (28, \mathbf{I}_{28,1}, ...))$, and the assistant observes it. We define the assistant's belief update b using Equation 2.2, where the state encoder f is an LSTM sequence model [53] trained to reconstruct the full image given a sequence of pixel observations. We assume that the user's state estimation process lies in a singleton hypothesis space $\mathcal{B} = \{b_{\mathbf{H}}\}$, where $b_{\mathbf{H}}$ is identical to the assistant's belief update b, except that it ignores any observation that consists of more than one row of pixels.

Under these assumptions, the optimal synthetic observation $\tilde{\mathbf{o}}_t$ consists of exactly one row of pixels. Furthermore, $\tilde{\mathbf{o}}_t$ minimizes the Euclidean distance between the user's latent state $f(\tilde{\mathbf{o}}_{0:t-1}, \tilde{\mathbf{o}}_t)$ after observing the partial image, and the assistant's latent state $f(\mathbf{o}_0)$ after observing the full image (taking $\sigma^2 \to 0$ in the assistant's belief update in Equation 2.2 simplifies the KL-divergence to the Euclidean distance between mean states). To compute the assistant's belief state after observing the full image at time $t = 0$, we break up the full image into an arbitrary sequence of pixel row observations, and feed each observation to the RNN state encoder f one by one. Note that all of this occurs immediately after the assistant observes the full image at time $t = 0$: by the time the assistant computes the optimal observation for the user at time $t = 0$, the assistant has already processed the entire sequence of observations. In the simulation experiment in Appendix 6.1, we train a policy π_θ (to model user actions via Equation 2.4) end to end with the state estimation model b_θ.
Car Racing. There are a maximum of $T = 1000$ timesteps per episode. Each observation consists of an image and a binary feature that indicates whether the image is delayed: $\Omega = \mathbb{R}^{64x64} \times \{0, 1\}$, where 0 indicates no delay and 1 indicates delay. We define the assistant's belief update b using Equation 2.2, where the state encoder f is composed of a recurrent neural network (RNN) dynamics model [98] and a variational auto-encoder model of image observations (VAE) [59] trained on random trajectories [43]. The state space $\mathcal{S}$ is the 256-dimensional latent space of the RNN f. The VAE uses a 32-dimensional latent space to model 64x64 RGB image observations. We assume that the user's state estimation process lies in a singleton hypothesis space $\mathcal{B} = \{b_{\mathbf{H}}\}$, where $b_{\mathbf{H}}$ is identical to the assistant's belief update b, except that it ignores the binary delay indicator in the observations and simply assumes all observations are not delayed.

Under these assumptions, the optimal synthetic observation $\tilde{\mathbf{o}}_t$ minimizes the Euclidean distance between the user's latent state $f(\tilde{\mathbf{o}}_{0:t-1}, \tilde{\mathbf{o}}_t, \mathbf{a}_{0:t-1})$ and the assistant's latent state $f(\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1})$ (as in MNIST, taking $\sigma^2 \to 0$ in the assistant's belief update in Equation 2.2 simplifies the KL-divergence to the Euclidean distance between mean states). If the last d observations are delayed, we approximate this solution using a prediction of a current, non-delayed observation: $\tilde{\mathbf{o}}_t \leftarrow \hat{\mathbf{o}}_t$. This prediction is made by replacing the delayed observations $\mathbf{o}_{t-d+1:t}$ with

recursively predicted, non-delayed observations $\hat{\mathbf{o}}_{i>t-d} = (g(f(\mathbf{o}_{0:t-d}, \hat{\mathbf{o}}_{t-d+1:i-1}, a_{0:i-1})), 0)$ from the RNN state encoder f and VAE image decoder g, where the 0 indicates that the predicted observation is not delayed.

2D navigation. The states are arranged in a 5x5 grid, and the number of states is $|\mathcal{S}|= 100$. Each state (x, y, ϕ) contains a discrete position $x, y \in \mathbb{N}$ and discrete orientation $\phi \in \{\text{N}, \text{S}, \text{E}, \text{W}\}$. The actions, $\mathcal{A} = \{\text{turn left}, \text{turn right}, \text{move forward}\}$, change the user's orientation or position deterministically. There are a maximum of $T = 25$ timesteps per episode. The environment contains a discrete set of 78 objects (26 in each of the 3 categories): $\mathcal{X} = \{\text{chair, window, bathtub, painting, ...}\}$. The observation space is the power set of the set of objects: $\Omega = \mathcal{P}(\mathcal{X})$. The observation model $p^{\text{obs}}(\mathbf{o}|\mathbf{s})$ is a delta function on the subset of all objects $\mathbf{o}$ visible from state $\mathbf{s}$. The assistant knows the locations of all objects, and can observe all objects in front of the user simultaneously: in other words, the assistant performs Bayesian belief updates (Equation 2.1) using the true observation model p^{obs}. We compute the optimal synthetic observation $\tilde{\mathbf{o}}_t$ by simply enumerating the singleton sets of objects in Ω and computing the KL-divergence (Equation 2.3) for each possible value of $\tilde{\mathbf{o}}_t$. We set the (goal-conditioned) policy $\pi(a|s; g) \propto \exp(Q(s, a; g))$, which is used to model user actions in Equation 2.4, to take the shortest path to the goal: the value function Q is computed using tabular soft Q-iteration [113, 11] with a reward function that gives a constant negative penalty for each state transition that does not reach the goal.

Lunar Lander. We blend the color of the lander's body with the background to make it difficult for the user to see, making the tilt indicator more prominent. Each episode ends when the lander contacts the ground, which typically occurs at $T = 115$ timesteps. We define the assistant's belief update b using Equation 2.2, where the state encoder f simply passes through the most recent observation: $f(\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}) = \mathbf{o}_t$. We hardcode the optimal policy π used to model user actions in Equation 2.4: if the angle $s_t > 0$, then fire the right thruster to counter-rotate the vehicle counter-clockwise; or if the angle $s_t < 0$, then fire the left thruster to counter-rotate the vehicle clockwise. To simplify the integral in Equation 2.4, we take $\sigma^2 \to 0$ in the model of the user's belief update b_θ (Equation 2.2).

Habitat navigation. The number of states is $|\mathcal{S}|= 1640$. The number of objects is $|\mathcal{X}|= 34$. The initial state distribution is uniform: $s_0 \sim \text{Unif}(\mathcal{S})$. At the beginning of the episode, the user has a uniform belief distribution over possible initial states. For each episode, we sample a goal state uniformly at random from $\mathcal{S}$. There are a maximum of $T = 100$ timesteps per episode.

Table 6.2: Car Racing User Study

	p-value	Unassisted	ASE
I was able to keep the car on the road	$<$ **.0001**	1.67	**3.75**
I could anticipate the consequences of my steering actions	$<$ **.001**	2.25	**4.17**
I could tell when the car was about to go off road	$<$ **.01**	3.08	**4.33**
I could tell when I needed to steer to keep the car on the road	$<$ **.05**	3.17	**4.83**
I was often able to determine the car's current position using the picture on the screen	$<$ **.05**	3.50	**4.75**
I could tell that the picture on the screen was sometimes delayed	$<$ **.001**	6.83	**4.25**
The delay made it harder to perform the task	$<$ **.01**	6.58	**4.83**

Subjective evaluations from 12 participants. Means reported below for responses on a 7-point Likert scale, where 1 = Strongly Disagree, 4 = Neither Disagree nor Agree, and 7 = Strongly Agree. p-values from a one-way repeated measures ANOVA with the presence of assistance as a factor influencing responses.

Table 6.3: 2D Navigation User Study

		p-value	Unassisted	Assisted
Naive ASE	I was often able to infer my current position and orientation	$>$ .05	5.50	5.67
	I was often able to move toward the goal	$>$ .05	5.50	5.58
	I often found the guidance helpful	$>$ .05	6.00	5.50
	I often forgot which position and orientation I believed was in	$>$ .05	3.17	3.25
ASE	I was often able to infer my current position and orientation	$<$ **.01**	5.50	**6.83**
	I was often able to move toward the goal	$<$ **.01**	5.50	**6.83**
	I often found the guidance helpful	$<$ **.01**	6.00	**6.92**
	I often forgot which position and orientation I believed was in	$<$ **.01**	3.17	**1.42**

Subjective evaluations from 12 participants. Means reported below for responses on a 7-point Likert scale, where 1 = Strongly Disagree, 4 = Neither Disagree nor Agree, and 7 = Strongly Agree. p-values from a one-way repeated measures ANOVA with the presence of assistance as a factor influencing responses.

Table 6.4: Lunar Lander User Study

	p-value	Unassisted	ASE
I could tell when the lander was tilted	$<$ **.05**	5.67	**6.33**
I was able to straighten the lander before it tilted out of control	$>$ .05	4.42	4.58

Subjective evaluations from 12 participants. Means reported below for responses on a 7-point Likert scale, where 1 = Strongly Disagree, 4 = Neither Disagree nor Agree, and 7 = Strongly Agree. p-values from a one-way repeated measures ANOVA with the presence of assistance as a factor influencing responses.

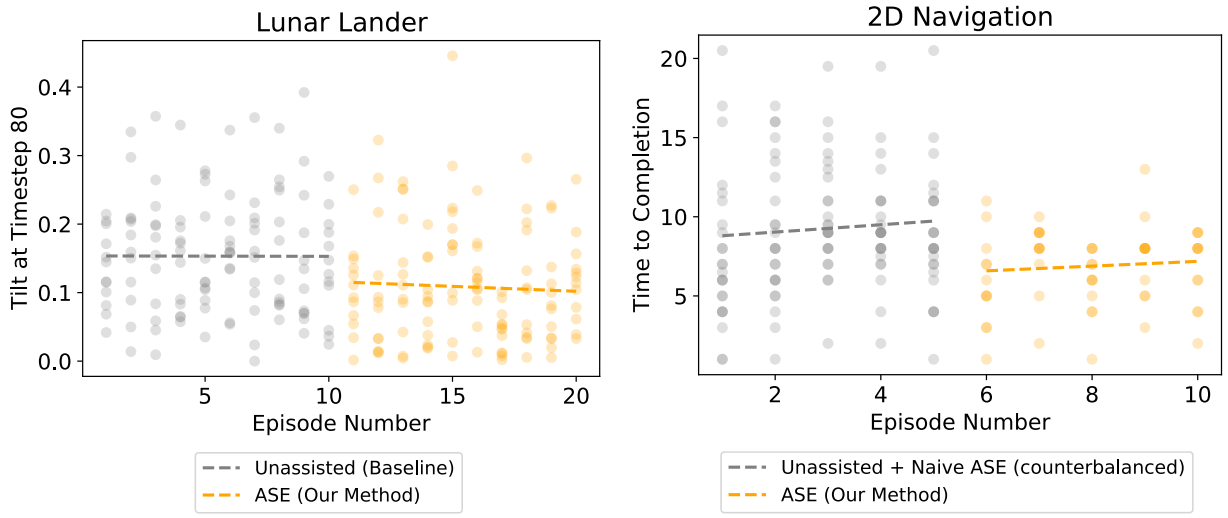

Figure 6.4: For a given episode number, each circle represents a different user. Each dashed line shows an ordinary least squares regression model trained on the data from a particular phase. Though we did not counterbalance the unassisted and ASE phases (only the unassisted and naïve ASE phases), the learning effect does not appear to be a substantial confounder. Performance is relatively constant during the unassisted phase, and sharply improves once the ASE phase begins. This suggests that the improvement in performance between the unassisted and ASE phases is primarily due to the introduction of the ASE assistant, rather than a learning effect. We plot the tilt at timestep 80 for Lunar Lander, since that is when the performance improvements from assistance tend to appear (see plot (a) in Figure 2.5).

6.2 Inferring Beliefs about Behavior from Dynamics

Experiments

Grid World Navigation

In Section 3.3 (in the main paper), we described two ways to adapt our algorithm to MDPs with a continuous state space: constraint sampling, and choosing a deterministic model of the internal dynamics. In this section, we evaluate our method on an MDP with a discrete state space in order to avoid the need for these two tricks. Our goal is to learn the internal dynamics and use it to assist the user through internal-to-real dynamics transfer. To sanity-check our algorithm and analyze its behavior under various hyperparameter settings and regularization choices, we implement a simple, deterministic grid world environment in which the simulated user attempts to navigate to a target position.

Hypothesis. Our algorithm is capable of learning accurate tabular representations of the internal dynamics for MDPs with a discrete state space. The two regularization schemes proposed in Section 3.3 (in the main paper) improve the quality of the learned internal dynamics model.

Task description. The state space consists of 49 states arranged in a 7x7 grid. The action space consists of four discrete actions that deterministically move the agent one step in each of the cardinal directions. The reward function emits a large bonus when the agent hits the target, a large penalty when the agent goes out of bounds, and includes a shaping term that rewards the agent for moving closer to the target. An episode lasts at most 100 timesteps. Each of the 49 states is a potential target, so the environment naturally yields 49 distinct tasks.

Corrupting the internal dynamics. To simulate suboptimal behavior, we create two users: one user whose action labels have been randomly scrambled in the same way at all states (e.g., the user's 'left' button actually moves them down instead, and this confusion is the same throughout the state space), and a different user whose action labels have been randomly scrambled in potentially different ways depending on which state they're in (e.g., 'left' takes them down in the top half of the grid, but takes them right in the bottom half). We refer to these two corruption models as 'globally scrambled actions' and 'locally scrambled actions' respectively. The users behave near-optimally with respect to their internal beliefs of the action labels, i.e., their internal dynamics, but because their beliefs about the action labels are incorrect, they act suboptimally in the actual environment.

Evaluation. We evaluate our method on its ability to learn the internal dynamics models of the simulated suboptimal users, i.e., on its ability to unscramble their actions, given demonstrations of their failed attempts to solve the task. The dependent measures are the next-state prediction accuracy of the learned internal dynamics compared to the ground truth internal dynamics, as well as the user's success rate when they are assisted with internal-to-real dynamics transfer (see Section 3.4 in the main paper) using the learned internal dynamics.

Implementation details. We use tabular representations of the Q_{θ_i} functions and the

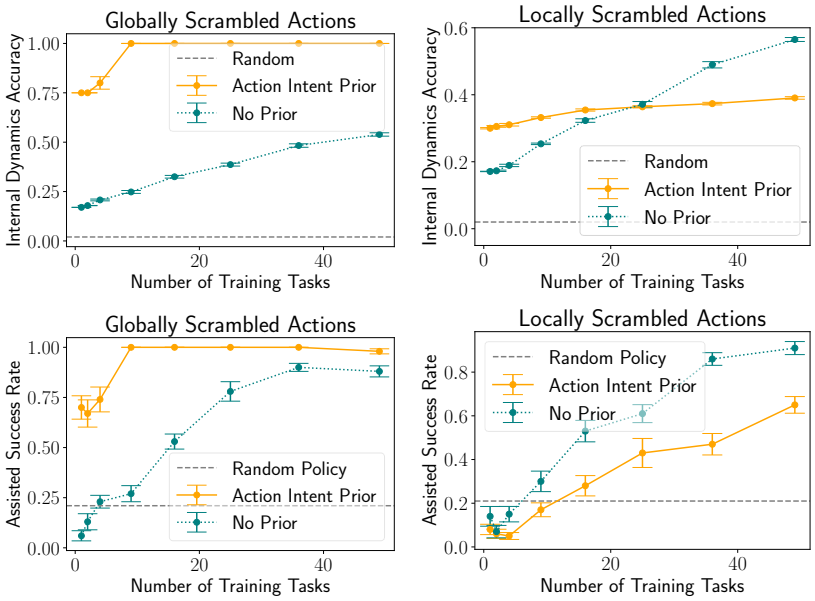

Figure 6.5: Error bars show standard error on ten random seeds. Corrupting the internal dynamics of the simulated user by scrambling actions the same way at all states (top and bottom left plots) induces a much easier internal dynamics learning problem than scrambling actions differently at each state (top and bottom right plots).

internal dynamics T_ϕ. We collect 1000 demonstrations per training task, set $\rho = 2 \cdot 10^{-3}$ in Equation 3.6 (in the main paper), and enumerate the constraints in Equation 3.6 (in the main paper) instead of sampling.

Manipulated factors. We manipulate (1) whether or not the user receives assistance in the form of internal-to-real dynamics transfer using the learned internal dynamics model – a binary variable; (2) the number of training tasks on which we collect demonstrations – an integer-valued variable between 1 and 49; (3) the structure of the internal dynamics model – a categorical variable that can take on two values: state intent, which structures the internal dynamics in the usual way, or action intent, which uses Equation 3.7 (in the main paper) instead; and (4) the user's internal dynamics corruption scheme – a categorical variable that can take on two values: globally scrambled actions, or locally scrambled actions.

Analysis. Figure 6.5 provides overall support for the hypothesis that our method can effectively learn a tabular representation of the internal dynamics for an MDP with a discrete state space. The learned internal dynamics models are accurate with respect to the ground truth internal dynamics, especially when the user's internal dynamics corruption is systematic throughout the state space (top and bottom left plots).

We also compare the two regularization schemes discussed in Section 3.3 (in the main paper): training on multiple tasks, and imposing an action intent prior. Internal models are easier to learn when the user demonstrates their behavior on multiple training tasks, as

shown by the increase in accuracy as the number of tasks (on the horizontal axis) increases. Regularizing the internal dynamics using action intent can be useful in some cases when the internal dynamics systematically deviate from the real dynamics, like when the user's actions are scrambled in the same way throughout the state space (top and bottom left plots, compare orange vs. teal curve), but can have a varying effect in other cases where the internal dynamics are severely biased away from reality, like when the action scrambling varies between states (top and bottom right plots, compare orange vs. teal curve).

2D Continuous Navigation

In the previous section, we adopted a tabular grid world environment in order to avoid constraint sampling in Equation 3.6 (in the main paper). Now, we would like to show that our method still works even when we sample constraints to be able to handle a continuous state space.

Hypothesis. Our algorithm can learn accurate continuous representations of the internal dynamics for MDPs with a continuous state space.

Task description. As mentioned in Section 3.1 (in the main paper), a classic study in cognitive science shows that people's intuitive judgments about the physics of projectile motion are closer to Aristotelian impetus theory than to true Newtonian dynamics [18]. In other words, people tend to ignore or underestimate the effects of inertia. Inspired by this study, we create a simple 2D environment in a which a simulated user must move a point mass from its initial position to a target position as quickly as possible using a discrete action space of four arrow keys and continuous, low-dimensional observations of position and velocity. The system follows deterministic, linear dynamics. Formally,

$$\boldsymbol{x_{t+1}} = A\boldsymbol{x_t} + B\boldsymbol{u_t} \tag{6.1}$$

where $\boldsymbol{x} = (x, y, v_x, v_y)^\intercal$ denotes the state, $\boldsymbol{u} \in \{(\pm 0.01, 0)^\intercal, (0, \pm 0.01)^\intercal\}$ denotes the control,

$$A = \begin{pmatrix} 1 & 0 & a_{13} & 0 \\ 0 & 1 & 0 & a_{24} \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{pmatrix}, B = \begin{pmatrix} b_{11} & 0 \\ 0 & b_{22} \\ b_{31} & 0 \\ 0 & b_{42} \end{pmatrix}.$$

At the beginning of each episode, the state is reset to $\boldsymbol{x_0} = (x_0 \sim \text{Unif}(0, 1), y_0 \sim \text{Unif}(0, 1), 0, 0)$. The episode ends if the agent reaches the target (gets within a 0.02 radius around the target), goes out of bounds (outside the unit square), or runs out of time (takes longer than 200 timesteps).

Corrupting the internal dynamics. In the simulation, actions control acceleration and inertia exists; in other words, $b_{11} = b_{22} = 0$ and the rest of the parameters are set to 1. We create a simulated suboptimal user that behaves as if their actions control velocity and inertia does not exist, which causes them to follow trajectories that oscillate around the target or go out of bounds. The user behaves near-optimally with respect to their internal beliefs

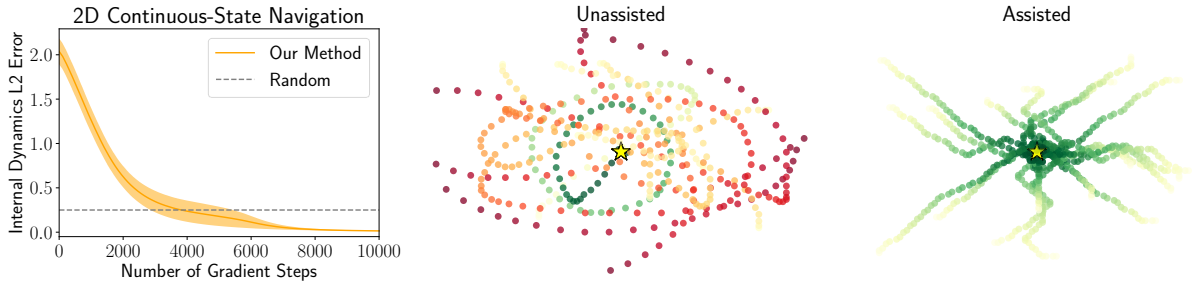

Figure 6.6: Our method is able to assist the simulated suboptimal user through internal-to-real dynamics transfer. Sample paths followed by the unassisted and assisted user on a single task are shown above. Red paths end out of bounds; green, at the target marked by a yellow star.

about the dynamics, but because their beliefs are incorrect, they act suboptimally in the real environment.

Evaluation. As before, we evaluate our method on its ability to learn the internal dynamics models of the simulated suboptimal user given demonstrations of their failed attempts to solve the task. We manipulate whether or not the user receives assistance in the form of internal-to-real dynamics transfer using the learned internal dynamics model. The dependent measures are the L2 error of the learned internal dynamics model parameters with respect to the ground truth internal dynamics parameters, and the success and crash rates of the user in each condition.

Implementation details. We fix the number of training tasks at $n = 49$, and use a multi-layer perceptron with one hidden layer of 32 units to represent the $Q_{\boldsymbol{\theta_i}}$ functions. We use a linear model based on Equation 6.1 to represent the internal dynamics $T_{\boldsymbol{\phi}}$, in which $\hat{a}_{13}, \hat{a}_{24}, \hat{a}_{33}, \hat{a}_{44}, \hat{b}_{11}, \hat{b}_{22}, \hat{b}_{31}, \hat{b}_{42} \in [0, 1]$. We collect 1000 demonstrations per training task, set $\rho = 2$ in Equation 3.6 (in the main paper), and sample constraints in Equation 3.6 (in the main paper) by collecting 500 rollouts of a random policy in the real world (see Section 3.3 in the main paper for details).

Analysis. Our algorithm correctly learns the following internal dynamics parameters: (1) $\hat{a}_{33} = \hat{a}_{44} = 0$ in the learned internal dynamics, which corresponds to the user's belief that inertia does not exist; (2) $\hat{b}_{11} = \hat{b}_{22} = 1$ and $\hat{b}_{31} = \hat{b}_{42} = 0$ in the learned internal dynamics, which matches the user's belief that they have velocity control instead of acceleration control. The learned internal dynamics maintains $\hat{a}_{13} = \hat{a}_{24} = 1$, as in the real dynamics, which makes sense since the user's behavior is consistent with these parameters. Figure 6.6 (left plot) demonstrates the stability of our algorithm in converging to the correct internal dynamics.

Figure 6.6 (center and right plots) shows examples of trajectories followed by the simulated suboptimal user on their own and when they are assisted by internal-to-real dynamics transfer. The assisted user tends to move directly to the target instead of oscillating around it or missing it altogether.

Learning Rewards from Misguided User Demonstrations

The previous simulation experiments show that our algorithm can learn internal dynamics models that are useful for shared autonomy. Now, we explore a different application of our algorithm: learning rewards from demonstrations generated by a user with a misspecified internal dynamics model. In order to compare to prior methods that operate on tabular MDPs, we adopt the grid world setup from Section 6.2, with globally scrambled actions as the internal dynamics corruption scheme.

Hypothesis. Standard IRL algorithms can fail to learn rewards from user demonstrations that are 'misguided', i.e., suboptimal in the real world but near-optimal with respect to the user's internal dynamics. Our algorithm can learn the internal dynamics model, then we can explicitly incorporate the learned internal dynamics into standard IRL to learn accurate rewards from misguided demonstrations.

Evaluation. We evaluate our method on its ability to learn an internal dynamics model that is useful for 'debiasing' misguided user demonstrations, which serve as input to the MaxCausalEnt IRL algorithm described in Section 3.4 (in the main paper). We manipulate whether we use the learned internal dynamics, or assume the internal dynamics to be the same as the real dynamics. The dependent measure is the true reward collected by a policy that is optimized for the rewards learned by MaxCausalEnt IRL.

Implementation details. We implement the MaxCausalEnt IRL algorithm [119, 50]. The reward function is represented as a table $R(s)$.

Analysis. Figure 3.2 (in the main paper, right plot) supports our claim that standard IRL is not capable of learning rewards from misguided user demonstrations, and that after using our algorithm to learn the internal dynamics and explicitly incorporating the learned internal dynamics into an IRL algorithm's behavioral model of the user, we learn accurate rewards.

User Study on the Lunar Lander Game

Task description. The reward function emits a large bonus at the end of the episode for landing between the flags, a large penalty for crashing or going out of bounds, and is shaped to penalize speed, tilt, and moving away from the landing site. The physics of the game are deterministic.

Evaluation protocol. We evaluate our method on its ability to learn the internal dynamics models of human users given demonstrations of their failed attempts to solve the task in the default environment. We manipulate whether or not the user receives assistance in the form of internal-to-real dynamics transfer using the learned internal dynamics. The dependent measures are the success and crash rates in each condition.

Implementation details. We fix the number of training tasks at $n = 9$ and use a multi-layer perceptron with one hidden layer of 32 units to represent the $Q_{\boldsymbol{\theta}_i}$ functions. We collect 5 demonstrations per training task per user, set $\rho = 2 \cdot 10^{-3}$ in Equation 3.6 (in the main paper), and sample constraints in Equation 3.6 (in the main paper) by collecting 100 rollouts of a random policy in the real world (see Section 3.3 in the main paper for details).

The physics of the game are governed in part by a configurable vector $\boldsymbol{\psi} \in \mathbb{R}^7$ that encodes engine power, game speed, and other relevant parameters. Since we cannot readily access an analytical expression of the dynamics, only a black-box function that forward-simulates the dynamics, we cannot simply parameterize our internal dynamics model using $\boldsymbol{\psi}$ (see Section 3.3 in the main paper for details). Instead, we draw 100 random samples of $\boldsymbol{\psi}$ and represent our internal dynamics model as a categorical probability distribution over the samples. In other words, we approximate the continuous space of possible internal dynamics models using a discrete set of samples. To accommodate this representation, we modify Equation 3.4 (in the main paper):

$$\begin{aligned}\delta_{\boldsymbol{\theta_i},\boldsymbol{\phi}}(s,a) &\triangleq Q_{\boldsymbol{\theta_i}}(s,a) - \mathbb{E}_{j\sim\text{Cat}(100,\boldsymbol{\phi})}\left[\int_{s'\in\mathcal{S}} T_{\boldsymbol{\psi_j}}(s'|s,a)\left(R_i(s,a,s') + \gamma V_{\boldsymbol{\theta_i}}(s')\right)\mathrm{d}s'\right] \\ &= Q_{\boldsymbol{\theta_i}}(s,a) - \sum_{j=1}^{100} \boldsymbol{\phi}_j \cdot \int_{s'\in\mathcal{S}} T_{\boldsymbol{\psi_j}}(s'|s,a)\left(R_i(s,a,s') + \gamma V_{\boldsymbol{\theta_i}}(s')\right)\mathrm{d}s'.\end{aligned}$$

Subject allocation. We recruited 9 male and 3 female participants, with an average age of 24. Each participant was provided with the rules of the game and a short practice period of 9-18 episodes to familiarize themselves with the controls and dynamics. Each user played in both conditions: unassisted, and assisted. To avoid the confounding effect of humans learning to play the game better over time, we counterbalanced the order of the two conditions. Each condition lasted 45 episodes.

Counterbalancing the order of the two conditions sometimes requires testing the user in the assisted condition *before* the unassisted condition, which begs the question: where do the demonstrations used to train the internal dynamics model used in internal-to-real dynamics transfer assistance come from, if not the data from the unassisted condition? We train the internal dynamics model used to assist the k-th participant on the pooled, unassisted demonstrations of all previous participants $\{1, 2, ..., k-1\}$. After the k-th participant completes both conditions, we train an internal dynamics model solely on unassisted demonstrations from the k-th participant and verify that the resulting internal dynamics model is the same as the one used to assist the k-th participant.

Analysis. After inspecting the results of our random search over the internal dynamics space, we found that the game speed parameter in $\boldsymbol{\psi}$ had a much larger influence on the quality of the learned internal dynamics and the resulting internal-to-real dynamics transfer than the other six parameters. Hence, in Figure 3.3 (in the main paper, bottom left plot), we show the results of a grid search on the game speed parameter, holding the other six parameters constant at their default values. The game speed parameter governs the size of the time delta with which the game engine advances the physics simulation at each discrete step. This parameter indirectly controls the strength of the forces in the game physics: smaller time deltas lead to smaller forces and generally slower motion, and larger deltas to larger forces and consequently faster motion.

We ran a one-way repeated measures ANOVA with the presence of assistance as a factor influencing success and crash rates, and found that $f(1, 11) = 109.58, p < 0.0001$ for the

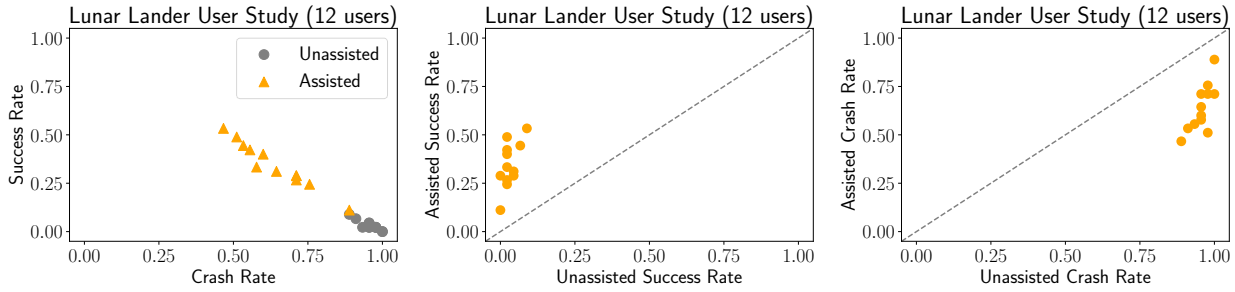

Figure 6.7: Assistance in the form of internal-to-real dynamics transfer increases success rates and decreases crash rates.

success rate and $f(1, 11) = 126.33, p < 0.0001$ for the crash rate. The assisted user succeeds significantly more often and crashes significantly less often than the unassisted user. Figure 6.7 shows the raw data.

Table 6.5: Subjective evaluations on Lunar Lander. Survey of $n = 12$ participants. Responses on a 7-point Likert scale, where 1 = Strongly Disagree, 4 = Neither Disagree nor Agree, and 7 = Strongly Agree. p-values from a one-sample t-test comparing the responses to mean 4.

		p-value	Mean	1	2	3	4	5	6	7
Pilot	I improved over time	> 0.05	4.33	1	2	1	1	4	1	2
	The game was too difficult	> 0.05	5.42	0	2	0	0	3	3	4
Copilot	**The copilot was generally helpful in completing the task**	$\mathbf{< 0.01}$	**5.92**	0	0	0	1	3	4	4
	I understood what the copilot was trying to do	> 0.05	4.75	0	1	1	3	4	1	2
	I could anticipate the copilot's behavior	$\mathbf{< 0.05}$	**4.17**	0	0	5	2	3	2	0
	The copilot improved over time	> 0.05	5.08	0	0	2	3	1	4	2
	The copilot was helpful in avoiding crashing	$\mathbf{< 0.001}$	**6.17**	0	0	0	0	3	4	5
	The copilot was helpful in avoiding flying out of bounds	> 0.05	4.50	1	0	2	4	1	2	2
	The copilot was helpful in landing quickly before time ran out	> 0.05	3.92	1	3	1	1	3	3	0
	The copilot was helpful in landing between the flags	> 0.05	4.83	1	1	1	2	1	3	3
	The copilot was too aggressive and didn't give me enough control	> 0.05	4.67	0	0	4	1	3	3	1
	I play differently with the copilot than without the copilot	$\mathbf{< 0.001}$	**6.75**	0	0	0	0	0	3	9
	I prefer playing with the copilot	$\mathbf{< 0.01}$	**6.17**	0	0	1	0	2	2	7

Table 6.6: Subjective evaluations for the quadrotor perching task. Survey of $n = 4$ participants. Responses on a 7-point Likert scale, where 1 = Strongly Disagree, 4 = Neither Disagree nor Agree, and 7 = Strongly Agree. p-values from a one-sample t-test comparing the responses to mean 4.

		p-value	Mean	1	2	3	4	5	6	7
Pilot	I have flown a quadrotor before	> 0.05	3.50	2	0	0	0	1	0	1
	I improved over time	> 0.05	5.25	0	0	0	1	2	0	1
	The game was too difficult	> 0.05	5.25	0	0	1	0	0	3	0
Copilot	**The copilot was generally helpful in completing the task**	$\mathbf{< 0.01}$	**6.75**	0	0	0	0	0	1	3
	The copilot made the task less stressful	$\mathbf{< 0.05}$	**6.50**	0	0	0	0	0	2	2
	I understood what the copilot was trying to do	$\mathbf{< 0.001}$	**7.00**	0	0	0	0	0	0	4
	I could anticipate the copilot's behavior	> 0.05	5.50	0	0	1	0	0	2	1
	The copilot improved over time	> 0.05	4.00	1	0	0	2	0	0	1
	The copilot was helpful in landing on the pad	$\mathbf{< 0.001}$	**7.00**	0	0	0	0	0	0	4
	The copilot was helpful in avoiding flying out of bounds	> 0.05	5.00	0	0	1	1	0	1	1
	The copilot was helpful in landing quickly before time ran out	> 0.05	5.00	0	0	1	0	2	0	1
	The copilot was helpful in pointing the camera	< 0.05	2.25	2	0	1	1	0	0	0
	The copilot was too aggressive and didn't give me enough control	> 0.05	3.50	0	1	1	1	1	0	0
	I play differently with the copilot than without the copilot	> 0.05	6.25	0	0	0	0	1	1	2
	I prefer playing with the copilot	> 0.05	5.25	1	0	0	0	0	1	2

6.3 Shared Autonomy via Deep Reinforcement Learning

Tables 6.5 and 6.6 summarize the subjective evaluations of the participants in our experiments.