Error Discovery through Human-AI Collaboration

by

Ramya Ramakrishnan

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

Signature redacted

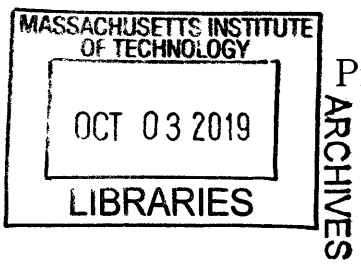
Author .....
Department of Electrical Engineering and Computer Science
August 30, 2019

Signature redacted

Certified by .....
Julie A. Shah
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Signature redacted

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students



Error Discovery through Human-AI Collaboration

by

Ramya Ramakrishnan

Submitted to the Department of Electrical Engineering and Computer Science
on August 30, 2019, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

While there has been a recent rise in increasingly effective human-AI teams in areas such as autonomous driving, manufacturing, and robotics, many catastrophic failures still occur. Understanding the cause(s) of these errors is crucial for reducing and fixing them. One source of error is due to an agent's or human's limited view of the world, which means their representations are insufficient for acting safely. For example, self-driving cars may have limited sensing that causes them to not recognize rare vehicle types, like emergency vehicles.

This thesis focuses on identifying errors that occur due to deficiencies in agent and human representations. In the first part, we develop an approach that uses human feedback to identify agent errors that occur due to an agent's limited state representation, meaning that the agent cannot observe all features of the world. Experiments show that using our model, an agent discovers error regions and is able to query for human help intelligently to safely act in the real world. In the second part, we focus on determining the cause of human errors as either occurring due to the human's flawed observation of the world or due to other factors, such as noise or insufficient training. We present a generative model that approximates the human's decision-making process and show that we can infer the latent error sources with a limited amount of human demonstration data. In the final thesis component, we tackle the setting where both an agent and a human have rich perception, but due to selective attention, they each only focus on a subset of features. When deploying these learned policies, important features in the real world may be ignored because the simulator did not accurately model all regions of the real world. Our approach is able to identify scenarios in which an agent should transfer control to a human who may be better suited to act, leading to safe joint execution in the world.

Thesis Supervisor: Julie A. Shah
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

My PhD experience has been transformative because of the support and guidance of many people. First, I would like to thank my advisor, Julie Shah. From my very first meeting with her in Flour bakery, I was inspired by her excitement and passion, and that spirit has lasted all the years of my PhD. Choosing to join her lab was the best decision I could have made. Julie taught me to be a stronger researcher. Her unique ability to tie together different research works and see the overlap among them quickly and easily, as I've seen in many lab tours, have inspired me to think big and have a vision for how all these projects fit together to some longer-term goal. I'm also grateful for all of the opportunities she has provided for professional and personal development. It is clear that her sincere desire to help students resonates in everything that she does. Her encouragement and positivity lifted me up any time things weren't going well and helped me move forward. I could not have gotten through the PhD without her.

My thesis committee has been critical to making the work in this thesis stronger and more clear. First, my internship host, Ece Kamar, has been a wonderful mentor and collaborator over the years. My first internship at Microsoft Research with her led me to choose the topic I wanted to pursue for my PhD. Because of the extremely positive experience I had there, I went back for a second internship. Her guidance during these short but productive months helped me learn how to quickly iterate on ideas and make progress quickly. Her friendly nature is comforting and speaks to her ability to connect so well with her mentees. She continued to be a collaborator as I returned to MIT, and I'm so thankful to have gotten the opportunity to work together.

Leslie Kaelbling and Russ Tedrake were instrumental committee members whose perspective helped me to increase clarity in notation and results throughout the works. Leslie provided advice on notation to use when referring to observations, states, POMDPs, etc, which led to more precise explanations of the ideas. Russ was helpful as well in making sure all the notation was correct, such as assumptions about

the dynamics of simulation and real world, and gave great feedback on thinking more critically on results.

My mentors and collaborators along the way have helped me to grow stronger as a researcher. Eric Horvitz, Debadeepta Dey, and Besmira Nushi were mentors during my internships at Microsoft Research, who each provided a very unique perspective on the problems we worked on. Several collaborators in the lab helped me to learn quickly during grad school. Stefanos Nikolaidis was my first senior graduate student mentor and helped me navigate through the early part of grad school. I also worked with Chongjie Zhang, who provided valuable feedback on my Master's project on perturbation training. Keren Gu and Pem Lasota were also wonderful labmates to collaborate with. On my last thesis work, I also had the pleasure of working with Vaibhav Unhelkar, who was incredibly patient and helped me learn so much about graphical models, inference, and math while working on the project. I'm thankful for his guidance, which made all the difference.

I'm also thankful to faculty mentors. First, Regina Barzilay has been an incredible mentor and role model. She helped me through tough times in the journey and provided invaluable advice on professional and personal goals. James Mickens at Harvard was kind enough to provide advice on career and job search as well as useful feedback on talks, as he's a wonderful communicator. Finally, I'd like to thank the undergraduate researchers I've worked with – Matthew Beveridge, Dolly Yuan, Claire McGinnity, and Eric Chen – because they were dedicated collaborators and produced interesting results through their projects. My collaboration with Claire on human subject experiments for identifying agent errors and with Eric on the mini autonomous car demonstration are included in Chapter 3.

The Interactive Robotics Group is the best lab anyone could ever ask for. Over the years, I'm very lucky to have become close friends with my labmates. I still have the set of "open when..." letters that they gave me with various prompts and fun notes inside. Celebrating birthdays, having board game nights, and going out for lab outings are just a few of the fun memories we have had together. I'm grateful for the all of the people I've gotten to know well in the lab: Vaibhav, Abhi, Rebecca,

Lindsay, Claudia, Been, Joe, Pem, Ankit, Mycal, Chris, Shen, Kailah, Serena, Sarah, Thavishi, Yilun, Tariq, Stefanos, Heitor, Keren, Jessie, Reymundo, Matthew, Brad, Chongjie, Kyle, Jorge, and Brittney. You all make me so happy. I also want to thank our friendly admin assistant, Liz, who is one of the most organized people I've met and is always on top of everything.

MIT has the most sincere, friendly, and quirky people I have ever met. All of the people I know at MIT have contributed to my overall happiness during graduate school. My girls group (MDMRAJ or Mandy-Danielle-Mengfei-Ramya-Amy-Jennifer) have been a source of comfort and joy. Girls nights and girls lunches would always be a cozy environment for sharing happy memories, thoughts, and struggles, and getting loving support. My roommates over the years – Jess, Yu-Chien, Himani, and Emmy – have supported me at home after stressful days at work. It has been fun to dance together with Malvika and attend various Ashdown events. I have loved going on fun-filled hikes and having interesting conversations with Karthick, Lakshmi, Varuni, Mel, Julian, Himani, Ankur, Nisha, Tejas, Ardavan, Mikhil, Megha, and many others.

Most importantly, the love from my family has shaped me to be the person I am today. My mother and father sacrificed everything to take care of me and my brother. They came to visit me in Boston more times than I can count to help me with moves, preparation before important milestones, etc. They have taught me to always be calm no matter what chaos is going on around me, which has helped me to slow down and better handle stressful occasions. My brother was in my shoes many years ago, but I had no idea when he went through it what a roller coaster the journey would be. His brotherly support and guidance was extremely comforting while I navigated the PhD process. My sister-in-law has been so much fun to hang out with. Every time I go visit them in California, we go on interesting hikes, play competitive board games, and cook delicious food together. I also want to thank my grandparents and relatives for their support and care over the years.

My husband's family has been equally supportive and encouraging. My mother-in-law, father-in-law, and brother-in-law welcomed me with open arms into the family, which has been a wonderful feeling. I'm thankful I got to spend time with them for

a few months in the summer. It has been so much fun cooking together, watching interesting shows, organizing the house, and so much more, all of which kept me sane during the final stretch. His extended family and relatives have been amazing cheerleaders as well. It has been especially nice to be just 20 minutes away from his aunt and uncle in New Jersey. Finally and most importantly, I'm grateful to MIT for bringing my husband, Karthik, and I together. Having research discussions with Karthik is always extremely fun and has motivated me to think about new, interesting ideas. He has gone through every roller coaster in graduate school with me, and I truly think he has learned the optimal policy in our relationship, always making me happy regardless of which state I'm in. He has no blind spots, and is the best possible human collaborator.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There has been a steady increase in AI systems and people working together to accomplish complex tasks across a variety of applications. In autonomous driving, for example, self-driving cars use human feedback and intervention to improve driving performance and reduce errors [245, 255, 75]. AI systems are also integrated into healthcare applications to aid doctors with treatment decisions based on prior patient data [177, 123, 168]. People are working alongside robots as well in manufacturing and disaster response settings where humans help robots generalize and adapt to variations in tasks [159, 3, 186]. While there is a rise in increasingly effective human-AI teams, unfortunately, many catastrophic failures of both AI systems and people still occur [78, 49]. For example, an Uber self-driving car recently killed a pedestrian due to the inability of the car to execute the correct action and the inability of the human overseer to take control early enough. Similarly, accidents by human drivers occur every day across the world, leading to many injuries and deaths.

Understanding the cause of these errors can be extremely beneficial for reducing and fixing costly mistakes. This thesis focuses on errors that occur due to representation deficiencies. In the first part of the thesis, an AI system is unequipped to observe important aspects of the real world, which results in costly errors. For example, a simple robot with a proximity sensor will not be able to observe rich visual properties of the world to perform tasks such as fetching specific objects. In the second part, the thesis considers a setting in which people may not be able to observe parts

of a task due to limited perception. For example, a person who is color-blind may not be able to prepare food properly if they cannot observe the colors of different ingredients. The final part of the thesis considers a scenario in which agents and people have rich perceptual capabilities but only use a subset of features for acting in the world due to selective attention. This can cause in-attentional blindness [137, 207, 136], which refers to the inability to see visible aspects of the world due to focused attention elsewhere. In the context of this thesis, agents or humans may not pay attention to important features that matter for a complex task due to limited training. Incompleteness of training environments can lead to flawed representations as well as selective attention to limited aspects of the world. For instance, a human driver trained on small roads may be able to perceive a truck on a highway but not focus their attention on this truck, resulting in a catastrophic accident. Representation errors can be fixed through the addition of sensors, technologies, and improved training environments that help AI systems and people be more aware of the world when making decisions.

All of the components in this thesis consider sequential decision-making tasks where an agent or a human is tasked with taking actions given an observation of the world. The true state of the real world encodes all of the information about the world needed to act. As a motivating example, let's imagine the task of driving. A state may include information about all nearby cars, traffic lights, pedestrians, etc. The driver gets an observation of this state based on her limited view. For example, the driver may only observe a flawed view of the region in front of the car with glare and with limited peripheral view. If color-blind, the driver additionally may not know the true colors of objects in the world. With this information, she is tasked with acting based on this knowledge. The driver decides what action to take – drive forward, take a right, stop, change the gear, look into the rear mirror, etc. Afterwards, the driver gets another observation of the world and again makes an action choice, and this process continues.

While a majority of the time, everything goes as planned, occasionally there may be accidents or collisions due to a variety of factors. If the error occurred because

the observation the driver was using to act had a critical aspect missing, the cause of the error is a representation limitation. For example, a self-driving car may not have complex sensors to distinguish large white vans from ambulances. When the car is near a van, it should keep driving, but near an ambulance, the car should slow down and yield. The driver's inability to observe the true type of the vehicle can lead to catastrophic accidents. Errors due to representation deficiencies are difficult to discover without external feedback.

In this thesis, agents and humans are assumed to be trained in limited settings, which often leads to flawed representations that do not generalize well beyond their respective training environments. There has been a large body of prior literature on building generalizable machine learning models that do not overfit to the training data by using, for example, methods for ignoring units in neural networks while training (Dropout) [209, 251, 165, 55, 72] and identifying adversarial examples [215, 121, 201, 13, 230]. More closely related to work in this thesis, Lakkaraju et al [122] propose a method to identify where trained models fail in the real world by discovering regions in which the model has high confidence but actually is incorrect. The proposed method aims to identify unknown unknowns of a machine learning model due to limited training data. While these works are useful for determining how AI systems can generalize better when trained in limited environments, they are primarily focused on supervised learning settings, rather than sequential decision-making scenarios, which introduce additional complexities.

In sequential decision-making tasks, such as in reinforcement learning, prior work in AI safety [73, 9, 166, 225, 110, 133] proposes many approaches for reducing the reality gap and making simulators more similar to the real world. For example, many realistic simulators, such as AirSim, CARLA, and Matterport3D, have been developed to more closely emulate the real world [200, 63, 45]. Some works develop approaches for exposing agents to larger varieties of scenarios in training to increase generalizability [225, 250, 166, 237]. In addition, several works have proposed methods for guaranteeing safety in the real world through added constraints and specifications [107, 71, 25, 232, 91, 4]. A large body of work in transfer learning [220, 163, 221, 218,

23, 120, 239, 167, 222] also relates to the problem of safely transferring from a limited training environment to the real world. These works, however, assume that an agent's representation is sufficient for doing these types of learning. If the representation itself is flawed, it becomes difficult for these methods to guarantee agent safety or accurately predict uncertainty in the real world. A notable gap in prior literature on AI safety is identifying safety errors due to representation incompleteness.

Representation learning [24, 48, 60] is a relevant area in that the aim is to learn generalizable representations that are capable of handling new data and tasks. Many recent works have focused on improving the generalization capability and interpretability of learned representations [135, 150, 52, 160, 151]. However, these representations may be insufficient for more complex real-world problems because they were learned using limited training data. This thesis is focused on identifying errors that occur due to flawed representations. Discovering these unexpected failures, or unknown unknowns, of AI systems and people can be very difficult to do autonomously.

Thus, this thesis takes a new angle of leveraging human-AI collaboration. Introducing a human helper provides unique opportunities for learning about representation deficiencies of AI systems that may not be possible to learn independently. Similarly, taking advantage of the strengths of agents can enable better understanding of human failures. On the human-machine interaction side, there has been extensive prior work on humans and agents working together on cooperative tasks [229, 173, 158, 155, 172, 197, 223, 35]. However, there is less work on using feedback for identifying representation deficiencies. Imitation learning [94, 189, 252, 11, 154, 1, 253, 227, 249], which refers to learning from expert data (humans or other agents), is applicable to the ideas in this thesis as well because expert data is used in various ways to identify limitations in representation. Prior work on multi-agent teaming [211, 20, 22, 21] is also related since it focuses on agents coordinating towards a shared goal or assisting teammates on a joint task. These works, however, often assume a shared representation, but when this is not the case, explicitly handling the mismatch in representation can lead to more accurate discovery of errors.

Overall, this thesis aims to address a gap in prior literature: leveraging human-

AI collaboration to identify failures due to representation limitations. Specifically, the goal is to discover representation errors of both agents and people, with the use of the other's help, and overcome these errors through intelligent transfer of control. Discovery of these errors can be extremely valuable for iteratively fixing errors through for example, augmented perception technologies, and improving safety.

1.1 Overview

This section provides an overview of the thesis, depicted in Figure 1-1, and outlines each of the three thesis components. The first part of the thesis introduces an approach for identifying agent errors due to an agent's limited representation of the world. The second part presents an approach for identifying human errors due to the human's flawed view of the world. Finally, the aim in the third part is to discover agent and human errors that occur because important aspects of the world are ignored during real-world execution. Identifying these errors can enable safe transfer of control. In this overview, each thesis part and how each relates with respect to assumptions and context is described. Then, the contributions of the thesis are detailed in the following section. In the final three sections, the inputs and outputs of each work are included with details of the approach and results.

The true real-world representation of the world is defined as $\mathcal{S}_{real}$, which is consistent across all three parts. Figure 1-1 presents an illustrative example that outlines the three thesis works with what assumptions are made in each problem. For simplicity, let's imagine that the true state consists of three features $\mathcal{S}_{real} = [x_1, x_2, x_3]$. (However, all problems are formulated such that the state can be represented with any number of N features.) Each thesis part introduces a problem in which an agent and a human have different views of the world. All of the presented works can be applied to a multi-agent setting rather than a human-agent scenario, as the problems are formulated in a general way. However, these works are motivated by the consideration of humans so the problems are described with a human as the other agent. As a result, the assumptions and modeling decisions are also influenced by human-AI

collaboration settings. The left column in Figure 1-1 is shaded because it specifies which agent is discovering the errors.

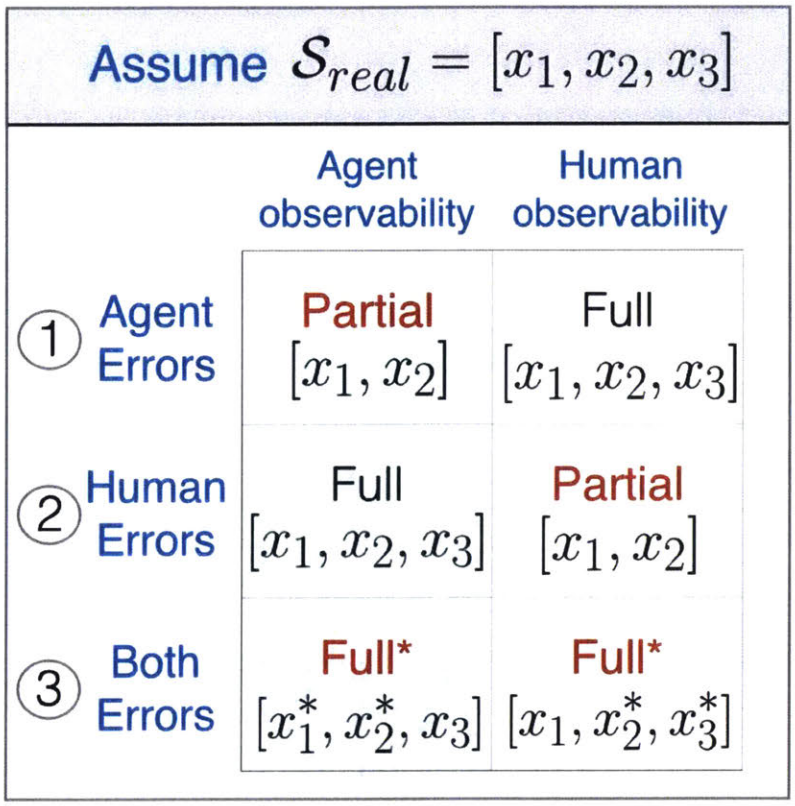


Figure 1-1: This overview figure is a simplified illustrative example of the assumptions made in each of the three thesis components. The real-world representation is defined using $\mathcal{S}_{real}$. For a clear comparison among the thesis parts, $\mathcal{S}_{real}$ is represented here using only three features: x_1, x_2, and x_3, and in each problem, the features the agent and the human have access to are described.

1. **Discovering agent errors:** The first row represents part 1 of this thesis, which introduces the problem of identifying errors due to an agent's partially observable view of the world. In this problem, an agent observes only a subset of the true features (e.g., x_1 and x_2), while the human observes all features. The agent's goal is to determine where it's likely to make errors due to this partial observability. Autonomously and safely learning about these errors is a challenging task since the agent cannot distinguish between many states that look identical (cases where x_1 and x_2 stay constant but x_3 changes). The agent

can learn about its errors by acting in the real world, making mistakes, and predicting where these mistakes are likely to occur with respect to x_1 and x_2. However, this is dangerous because it requires many errors to be made, such as injuring a pedestrian, before learning about them. In this work, feedback from a human who can observe the world fully is used to identify regions where the agent is highly likely to make errors due to limited representation.

2. **Discovering human errors:** In the second row, representing part 2, the human instead has partial observability of the world and thus, observes a subset of the true set of features (e.g., x_1 and x_2). Even though part 1 and part 2 look symmetrical, they approach the problems from different perspectives. In part 1, an agent has a limited representation and identifies its own error regions, while in part 2, the agent doing the reasoning has full observability and is identifying errors of another actor with flawed representation. While the other actor can be a human or another agent, this actor is referred to as a human throughout this thesis as the problems are motivated by ideas in human-AI collaboration. The agent is tasked with identifying when human errors occur due to flawed representation vs. other factors, such as noise. To identify the cause of a human error, we develop a generative model that describes the human's process of decision-making. The model includes latent variables, such as human "blind spots", a vector denoting which features are unobservable to a human, and random noise. Inference techniques are used to distinguish between these two sources of error. Separating these two types of errors can be useful for fixing the error in the future. Representation errors can be reduced through the addition of sensors to help the human "see" what they cannot, while other errors can be minimized through additional training.

3. **Discovering agent and human errors:** In the final row, an agent and a human have full observability of the world and thus can observe all features (e.g., x_1, x_2, and x_3). However, due to limited training environments, each focus on a subset of features through selective attention that are useful for the

trained task. So, for example, due to a limited simulator, the agent might only use features x_1 and x_2 in its decision-making. Similarly, the human may use features x_2 and x_3 based on her minimal training in a different world. In the real-world, the agent, who is ignoring feature x_3, might make a mistake due to in-attentional blindness. It's important to recognize in this scenario, when a feature might be important to pay attention to in the real world. While the agent does not know how to act in this part of the world as it was not trained, the agent can learn to transfer control to the human in these regions.

Learning about errors due to representation limitations can be extremely useful for reducing and fixing these errors in the future. Representation errors are particularly difficult to discover independently without external help, but with some feedback, these errors can be identified in the three contexts described above. Knowledge of these errors can guide the process of fixing the representation of agents and people by augmenting with the strengths of the other and additional technologies. For example, if agents are not well-suited to drive at nighttime, a human can take over, or if possible, more complex sensors for observing the environment in these challenging situations can be incorporated. Similarly, if people are color-blind, AI systems can be trained to take over control when there is a high chance of error and in addition, new technologies can be included to help people observe which color is being displayed.

1.2 Thesis Contributions

The contributions of this thesis are summarized as follows:

- **Chapter 2** presents the problem of identifying agent errors due to an agent's limited representation of the world. The proposed approach uses human feedback to predict where an agent is likely to make errors in the real world, leading to more self-aware AI systems. **Chapter 3** applies the method presented in the previous chapter to real users and a real-world application. Specifically, experiments are conducted to demonstrate that agent errors can be predicted

reasonably well with limited user data, and that this approach can be applied to a simple autonomous driving scenario.

- **Chapter 4** proposes a generative modelling approach to identify the cause of human errors: either occurring due to human representation limitations or due to factors such as noise. Results show that human blind spots and noise can be inferred given demonstration data.

- **Chapter 5** develops a method for safe joint execution in a setting where both an agent and a human can make errors. By identifying features each is using to act along with mismatches in behavior, the learned model is able to transfer control effectively between an agent and a human with complementary capabilities.

Next, each thesis component is described in more detail in the following three sections.

1.3 Discovering agent errors

The first step is to identify errors due to an agent's insufficient state representation. Agents are often trained in simulation environments that do not perfectly match the real world, which can cause costly errors when these agents are deployed. Prior works have developed approaches for improving the safety of AI systems through more realistic simulators [200, 63, 45], increasingly robust training procedures [225, 250, 166, 237], more accurate uncertainty estimates for more cautious exploration in the real world [110, 73], and human feedback [171, 70]. However, these works primarily focus on scenarios where an agent's representation is assumed to be sufficient for learning and acting. When the agent's representation is flawed, these methods may not generalize. The goal of this work is to identify where errors are likely to occur due to an agent's limited representation. Because the agent has a flawed view of the world and its policy can result in catastrophic failures in the real world, it is challenging for the agent to *autonomously* and *safely* discover its errors. In this work, human feedback is used to learn a predictive model of these representation errors, or *blind spots*, in order to reduce costly errors of the agent in real-world applications.

Humans provide data in the form of demonstrations in the real world or corrections, in which the human monitors the agent acting in the world and corrects dangerous actions. Because the person knows the true representation and the agent is operating in a limited representation, multiple signals from the human may look identical to the agent. The key idea is that our approach explicitly aggregates labels provided by the person to handle this representation issue. Finally, supervised learning is used to generalize blind spot predictions across the full agent representation space, since feedback is only given in a limited region of the world.

The approach is evaluated across two domains, and results show that our method achieves higher predictive performance than baseline methods, and also that the learned model can be used to selectively query an oracle at execution time to prevent errors. Safely collecting data with a human can help agents better predict where they might make errors in the world, leading to more self-aware AI systems. Further, an understanding of the agent's own errors with respect to its limited representation can be incredibly useful for refining its representation and its simulation environment for improved future training.

Problem

An agent is trained in a simulation environment with a specified representation that is sufficient to learn how to act in the simulator. The space of possible states under this representation is denoted as $\mathcal{S}_{agent}$. Using the simulator, the agent learns a policy $\Pi_{sim} : \mathcal{S}_{agent} \rightarrow \mathcal{A}$, which maps states in $\mathcal{S}_{agent}$ to actions $\mathcal{A}$. In the real world, however, the agent's representation lacks necessary features to represent the true state of the world, and thus the agent cannot distinguish between numerous states. This representation deficiency can lead the agent to make costly errors in the real-world environment.

To identify agent errors in the real world, our approach takes as input the agent's learned simulator policy π_{sim} and an oracle that can provide feedback. This oracle or human is assumed to have access to the true representation $\mathcal{S}_{real}$ and can provide either demonstrations in the real world or monitor the agent and correct the agent

while the agent acts according to the learned policy. An agent blind spot is defined as an observation in $\mathcal{S}_{agent}$ where an agent takes an unacceptable action in at least one real-world state that corresponds to it. The output is a trained blind spot classifier $M : \mathcal{S}_{agent} \rightarrow \{0, 1\}$, which specifies whether each observation in the agent's representation is a blind spot or is safe.

Approach

The pipeline for discovering agent blind spots begins with a data collection phase, in which the agent gets data from an oracle through various forms of feedback, such as demonstrations and corrections. In all feedback types, because multiple real-world states look identical to the agent, the collected data has **state representation (SR) noise**, which refers to one agent observation receiving many, potentially contradictory, labels from the oracle. Another form of noise, **action mismatch (AM) noise**, is introduced in feedback types in which the agent only observes oracle actions. When agent and oracle actions differ, it can either indicate two equally acceptable, different actions or a potential blind spot.

To handle both forms of noise, a label aggregation step is introduced, which estimates the noise in the labels with an approach from crowd-sourcing called Dawid-Skene. This method uses Expectation Maximization to predict the true label of each agent observation through aggregation of labels. Since the agent only receives feedback in a limited part of the observation space, a classifier is trained with these aggregated labels to generalize to unseen regions. The learning step makes the assumption that blind spots do not occur at random and instead are correlated with existing features that the agent has access to. Since blind spots are often rare in data, learning about them is an imbalanced learning problem. To address this, blind spot observations are oversampled and then calibration is used to correct estimates of the likelihood of blind spots. The final blind spot model $M : \mathcal{S}_{agent} \rightarrow \{0, 1\}$ outputs for each observation in $\mathcal{S}_{agent}$, a label of either safe (0) or blind spot (1).

Results

The approach is first evaluated on two domains with a simulated oracle. In both domains, a simulated environment is constructed, and the agent learns a policy in this world based on its representation. The proposed approach is then used to learn agent blind spots on a more complex "real-world" environment that is more rich than the simulation world. Our model is able to predict blind spot regions much more accurately than baseline methods. Additionally, we evaluate the quality of the learned model by deploying an agent into the real-world environment and having the agent query for oracle help any time the model predicts a blind spot with high probability. Compared to an agent that always queries for help and one that never queries, the agent using our model to query is able to get 80% of the total possible reward while only querying 30% of the time, which highlights the benefit of discovering error regions for increased awareness of high danger regions. Results also show that the bias in different forms of data collection affect the blind spots the agent learns, which can guide future exploration on the most suitable feedback types based on domain characteristics.

The approach is next evaluated with human users. First, we conduct experiments on the same two domains from simulated experiments. Results show that the model predicts blind spots reasonably well with limited user data. Further, participants prefer different types of feedback based on the domain, which may be an important consideration when determining how to best collaborate and use human help. Finally, our method is applied to a mini autonomous car application. In this domain, a car is trained in a simple simulated environment and evaluated on a more complex environment, with many new objects the agent has not seen before. Due to limited object sensing capabilities, the agent cannot differentiate between different objects. After learning a blind spot model, the robot car is able to ask for help at appropriate times to avoid obstacles and reach the goal in the real-world environment.

1.4 Discovering human errors

The next step is to discover errors of people that occur due to human representation limitations. In this part, an agent has full observability of the world and can observe a human perform a task and the errors she makes. However, the agent does not know why the human made a mistake. The agent's goal is to identify the cause of a human error based on observed demonstration data. While motivated by trying to understand the errors of humans, our general model can be used to analyze errors of any AI system. The model allows us to separate errors that occur due to representation limitations (blind spots), in which the human is unable to observe critical parts of the task, from non-representation errors, which can be noise in execution, systematic errors in a learned policy, etc. Separating these two types of errors can allow for future refinement accordingly (i.e., representation errors require perceptual augmentation, while other errors can be reduced through improved training).

Prior work in human-machine interaction [173, 158, 172, 223] learns models of human teammates, while work in reinforcement learning [74, 108] infers latent states. However, these works do not explicitly distinguish between representation errors vs. other sources of error. Work in learning with flawed representations [24, 235, 181, 182] consider the problem of limited representation, but mainly consider limitations in the agent's own representation rather than the opposite scenario here, in which an agent identifies the cause of an error of another agent or human.

This work presents a generative model to infer the cause of human errors based on observed demonstration data. The key idea is that the model explicitly includes two latent factors that can affect the human's actions and thus, human errors: the human's blind spots, which refers to features the human may not be able to observe when seeing the world, and human noise, which accounts for any other randomness in her decision-making. An inference approach is used to recover these two error sources given demonstrations of a human performing a task. Evaluations on two domains, one with simulated human demonstrations and one with real user data, indicate that human blind spots and noise can be inferred reasonably well using our approach.

Further, other aspects of the human's decision-making process can be inferred by querying for other latent variables in the model. Understanding the reason for why mistakes occur is a step towards identifying and repairing human and AI system errors.

Problem

An agent has access to the true state space of the world, denoted as $\mathcal{S}_{real}$, and is observing a human, with potential representation limitations, perform actions in this world. The person may not be able to observe all features of the true state $\mathcal{O}_h \subset \mathcal{S}_{real}$, which can lead to costly errors. The true human's observation space $\mathcal{O}_h$ is unknown to the agent. Instead, the agent only has access to human demonstration data, which includes a set of states in $\mathcal{S}_{real}$, which the agent can observe, actions of the human at each of those states, and whether or not the human's action caused an error in the world.

Given this demonstration data, the agent's goal is to identify 1) if the human's unknown representation lacks important features for performing the task, and 2) the amount of noise in human action decisions, both of which will help to explain the human actions and errors in the data. The human's representation limitations are explained by a blind spot vector $\mathbf{b} = [0, 1, 0, ..., 0]$, which describes whether each feature in the real world is a blind spot (unobservable) for the human. Human noise $\eta = [0, 1]$ is denoted by a value that represents what percentage of time the human is taking random actions. The goal of the agent is to determine given demonstration data D, the probability of human blind spots and noise $P(\mathbf{b}, \eta | D)$.

Approach

To do this, we develop a generative model that represents the human's decision-making process. The model includes two latent factors that affect the human's actions and errors: representation limitations and the amount of noise in execution. Given demonstration data, the agent's goal is to infer these two quantities to better understand the cause of human errors. The model assumes that the agent observes the true

state of the world $\mathbf{s}_{real} \in \mathcal{S}_{real}$. The human's inability to observe parts of the world is included as a latent variable denoting human blind spots $\mathbf{b} = [0, 1, 0, ..., 0]$, where each value is 1 if the human cannot observe that feature of the state. Using the true state and the blind spot vector, the human's observation $\mathbf{o}_h$ of the world, unknown to the agent, is modeled as a mask of the true state and the human's blind spots.

Human actions are based on the human's observation of the world $\mathbf{o}_h \in \mathcal{O}_h$ and some noise, modelled as a value between $\eta = [0, 1]$. To explain human actions, one approach could be to estimate the human's policy $\Pi_h : \mathcal{O}_h \rightarrow \mathcal{A}$, which maps human observations $\mathcal{O}_h$ to human actions $\mathcal{A}$. However, this can be quite challenging to model given limited human demonstration data. Alternatively, the agent is assumed to have access to an optimal policy given the true state representation $\Pi^* : \mathcal{S}_{real} \rightarrow \mathcal{A}$, which can be used to explain human actions. However, because this policy is not a function of the human's flawed observation, the optimal action given the human's observation $\mathbf{o}_h$ cannot be directly computed. We bridge the gap by modelling the human's implicit view of the world. If the human does not observe a certain feature, the model fills in the missing value with the most probable value the human may be assuming about the world when making an action decision. This gives an approximation of the human's state of the world from his or her perspective. For example, if the human is color-blind, she may not be able to observe the color displayed on a traffic light, but she may make an assumption about the color and act accordingly (e.g., assume red and stop).

The action of the human can then be estimated by using the human's implicit view of the world $\mathbf{s}_h$, the optimal policy π^{*s}, and the amount of human noise η. The error is computed using a given acceptable function. To compute $P(\mathbf{b}, \eta | D)$ given this generative model, variable elimination is used as an exact technique and Gibbs sampling as an approximate technique.

Results

The approach is evaluated on two domains, one with simulated human demonstrations and another with real user data. In both, the human is unable to observe a

critical component of the task (e.g., cannot distinguish salt from sugar in the kitchen). Inference with our generative model is able to characterize human blind spots as well as other human errors, grouped together as noise. One challenge is that it is difficult to always learn the ground truth generative model as there is limited data and a large number of possible explanations for the data. Thus, the model may not always learn the true blind spot vector and amount of noise. In our experiments, the model is able to identify blind spots accurately, but not the exact amount of noise in human execution. Thus, more informative priors can be used to better guide the model towards more reasonable solutions.

Results additionally show that other latent variables in the graphical model can also be queried to better understand the human's decision-making process. This can enable more informed fixing of human representation. On a real-word kitchen task in which we collect human data, the model is able to separate errors occurring due to unlabelled ingredients (representation limitations) vs. failures due to time pressure or low memory. Knowledge of this representation deficiency can motivate the need for augmenting the world with new information so that people's observation more closely matches reality, leading to reduced errors.

1.5 Discovering agent and human errors

The final step is to enable safe execution in the world when agents and humans can both make errors. In this setting, both have full observability of the world, but due to limited training environments, they each focus on a limited set of features that are important for that task. The real world can have complex regions that were not modeled in simulation, and thus factors that were irrelevant in training might actually be important in the real world. It may be necessary in these cases for an agent to transfer control to a human when novel regions are encountered. Prior works in AI safety [213, 73] and imitation learning [252, 11] consider problems of simulation-to-real world transfer, but they do not consider the problem of identifying agent and human errors for safe transfer of control. Outlier and novelty detection [43, 169] could

be used to identify novel situations different from the agent's training environment that may require a handoff, but results show that focusing on surface-level features that differ between the two worlds can be deceiving for learning when to transfer control.

Our framework learns error models and intelligently transfers control accordingly. The models are trained using data from the agent's simulation training as well as human demonstration data. The key idea is that our approach identifies which features the agent and human are selectively paying attention to and uses these features along with differences in agent and human behavior as signals for identifying who should act in different situations in the real world. For example, imagine a driving scenario, in which an agent is trained only in highways with cars and trucks, and a human is trained only in neighborhood settings with cars and emergency vehicles, such as ambulances. Using demonstration data, our system first identifies that a person considers ambulances as important, which the agent never saw in training and thus would completely ignore. In states with ambulances, the model predicts that an agent will likely keep driving near an ambulance (it ignores this feature in its trained policy) and demonstration data shows that a person slows down and yields. Since actions of the agent and human deviate, and the system identified an important feature ignored by the agent in this state, the human is assumed to be more suited to take control.

Our method is evaluated on two domains, and results show that using our approach, an agent can transfer control intelligently to the human for safe joint execution. This provides a step towards improved detection of errors and better awareness of where training environments need to be improved, leading to iterative refinement of policies as teams of agents and humans are deployed in the world.

Problem

Let's assume that there exists a real-world environment with state space $\mathcal{S}_{real}$. The states in this world are defined by a large set of rich features (e.g., color, location, and size of thousands of objects near the agent). An agent is trained in a simulation environment that is limited in that it only observes a subset of the real-world state

space $\mathcal{S}_a \subset \mathcal{S}_{real}$. Because of this limited world, the agent only focuses on a subset of the full feature set that are necessary to perform the task in training. Similarly, a human is trained in a complementary limited environment, which includes a potentially different subset of states $\mathcal{S}_h \subset \mathcal{S}_{real}$. Based on this, the human also pays attention to a limited subset of features. Selectively paying attention to a small set of features allows both the agent and human to learn tasks efficiently by focusing on the most important factors in that world. However, this may result in the agent ignoring important features that may actually matter for safe real-world execution because the real world includes scenarios that an agent may not have been trained on. The goal of this work is to identify regions of the world in which an agent should transfer control to a human that is better trained in that area.

The approach takes as input the learned simulator policy from the agent's training, data of the agent acting in simulation in the form of state-action pairs, and human demonstration data in a similar form. The goal is to learn two error models, one for the agent $\Theta_a : \mathcal{S}_{real} \rightarrow [0, 1]$ and one for the human $\Theta_h : \mathcal{S}_{real} \rightarrow [0, 1]$, both of which map states to probabilities of error in the real world. Given these models, the agent can transfer control if the human has a lower probability of error in a novel region of the world.

Approach

To learn agent and human error models, our system has a two-step pipeline. The first step identifies features that the human is using in her decision-making that the agent learned to ignore in simulation. To do this, the system applies behavior cloning and feature elimination techniques to human demonstration data to learn an estimate of the human's policy and a set of features that are important for the human's action decisions.

Once these important features are identified, the system learns error models for the agent and the human in order to transfer control effectively. To do this, noisy labels are extracted from simulator and demonstration data and then supervised learning is used to train models of agent and human errors. The process is as follows: 1)

For each datapoint in the simulation data, the agent is assumed to act well. Human errors are computed by predicting the human's actions in each of those states using an estimate of the human's policy and computing how far they deviate from the value of the optimal action, known to the agent. 2) Using demonstration data, the most likely agent action is computed using the agent's simulator policy. To assign labels of errors here, both action mismatches and feature activations are considered. For example, if the human's and agent's actions differ, and there is a feature activated in the real world that the agent ignored but the human did not, this can indicate that the agent made an error. The system uses these noisy signals as labels to train two classifiers that estimate the probability of error in each state for the agent and human respectively.

Results

The method is evaluated on two domains, in which an agent is trained in one limited region of the true environment and the human is trained in an overlapping but different limited region. In the real world, the team has to act safely in all situations. Results show that across both domains, an agent using our model to transfer control is successfully able to obtain consistently high reward in regions where the agent or the human makes mistakes. Simply deploying the agent's policy or the human's policy results in costly errors, while our handoff policy is able to obtain similar reward to an optimal meta-policy trained to either take the agent's action or the human's action at each state. The intelligent handoff occurs due to our pipeline selecting important features for determining who should act.

Results also show that our model prunes out surface-level features that differ between the two training environments that don't affect the policy. These surface-level features can mislead the agent into incorrectly handing over control to the human in some regions of the real world. Our model instead focuses on important factors that matter for acting in the world and thus, learns an effective hand-off policy. Finally, results indicate that our model is robust to human noise, as the model learns to prefer the agent in situations similar to the agent's training environment.

Chapter 2

Discovering Agent Errors

2.1 Introduction

Agents designed to act in the real world are often trained in a simulated environment to learn a policy that can be transferred to a real-world setting. Training in simulation can provide experiences at a low cost, but mismatches between the simulator and the real world can degrade the learned policy's performance in the open world and lead to costly errors. For example, consider an autonomous driving simulator that includes components for learning how to drive, take turns, stop appropriately, etc., but does not incorporate any emergency vehicles, such as ambulances or fire trucks. When an agent trained in such a simulator is in the presence of an emergency vehicle in the open world, it will continue to drive rather than pull over because it lacks knowledge of the true representation of the world, potentially leading to costly delays or even accidents.

We formally define the problem of discovering agent blind spots in reinforcement learning (RL). *Blind spots* are regions of the world where agents make unexpected errors due to mismatches between the training environment and the real world. Different kinds of limitations lead to different types of blind spots; in this work, we focus on blind spots stemming from limitations in state representation.

Limitations in the fidelity of the state space result in an agent being unable to distinguish between different real-world states, as highlighted in Figure 2-1. In the

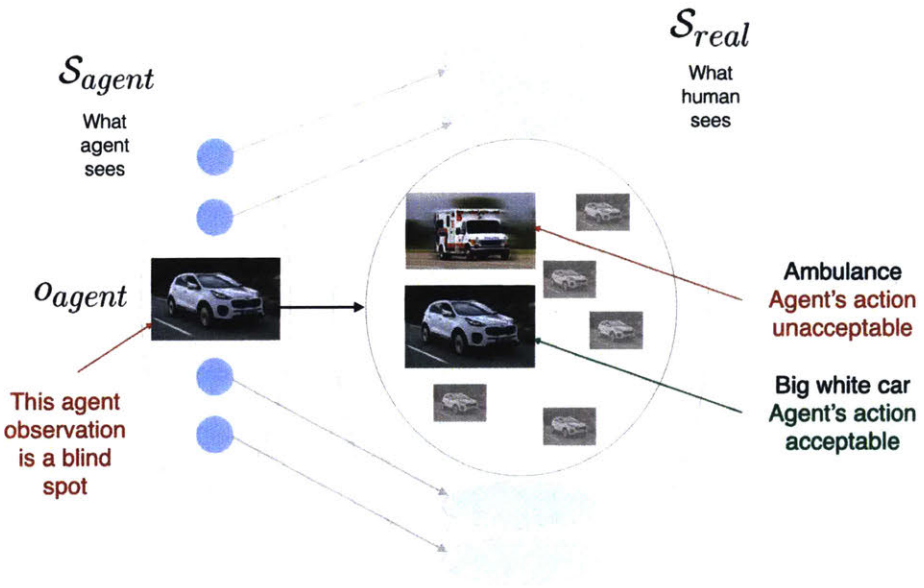


Figure 2-1: An example of a mismatch between agent representation ($\mathcal{S}_{agent}$) and real-world representation ($\mathcal{S}_{real}$), which can cause agent blind spots. In this case, the agent interprets an ambulance and a big white car as identical and thus continues driving. Because the agent takes an unacceptable action in the case of an ambulance, the corresponding agent observation represents a blind spot.

driving example mentioned above, for an agent trained in simulation, states with and without an ambulance would appear the same according to the learned representation. However, the optimal action to take in the open world in these two situations differs significantly, and it would be impossible for an agent that could not distinguish these states from one another to learn to act optimally, regardless of the quantity of simulation-based training.

Such representational incompleteness is ubiquitous in any safety-critical RL application, especially in robotics and autonomous driving, since real-world data can be dangerous to collect. An expert could make an agent's state representation more complete if the true representation were known a priori, but even with extensive engineering, a gap often exists between simulation and reality. Further, the agent's representation cannot be pre-specified when it is learned automatically through deep reinforcement learning. In cases where complete real-world representation is impossible, it is critical to invest in meta-level reasoning techniques for *identifying* blind

spots, which in turn enable representation and policy refinement.

Recent work has increasingly addressed safety in various ways with the goal of enabling safer deployment of AI systems. Several works have focused on building realistic simulators that provide rich training environments that are more similar to the real world, reducing the gap between simulation and reality [200, 63, 45]. Others [225, 250, 166, 237] have focused on developing more robust training, which involves training on perturbed environments to increase variety in what the agent sees before real-world execution. In addition to improved training, cautious exploration in the real world can enable safe deployment with fewer catastrophic failures during execution [110, 73]. Other recent works have also introduced approaches for using human feedback to avoid costly errors [171, 70].

While many of these works improve transfer from simulation to the real world, they assume that the agent's representation is sufficient for learning and acting. Our work considers the problem of identifying errors when the agent's representation is incomplete. We propose using human feedback to learn a blind spot model that can improve self-awareness and lead to safer real-world execution.

To discover errors caused by representation limitations, we propose a transfer learning approach that incorporates a learned simulator policy and *limited* oracle feedback to learn where blind spots are likely to occur. The oracle provides information either by performing the task itself (demonstrations) or monitoring and correcting the actions of the agent (corrections), which provides signals to the agent indicating whether its actions were acceptable in each state. We assume that blind spots do not occur at random, and that they correlate with features known to the agent. For example, an agent may lack the features to recognize emergency vehicles, but the existence of emergency vehicles may correlate with observable features, such as vehicle size or color. Under this assumption, we formalize the problem of discovering blind spots as one of supervised learning, where the objective is to learn a blind spot map that provides the likelihood of each agent observation being a blind spot by generalizing predictions to unseen regions of the world. That is, the agent learns the probability that each agent observation corresponds to at least one real-world state

in which the agent's chosen action would result in a high-cost error.

Note that learning a predictive model for blind spots is preferred over updating a learned policy when the agent's state representation is insufficient. In the driving example, two states that are indistinguishable to the agent require different actions: a state that includes an ambulance requires the agent to pull over to the side of the road and stop, while a state without an ambulance requires the agent to continue to drive at the speed limit. If the agent updates its policy for both of these similar-appearing states, the consequence could be costly and dangerous. Instead, a blind spot model can be used in any safety-critical real-world setting where the agent can query an oracle for help in potentially dangerous areas instead of committing to an incorrect and potentially catastrophic action.

Formalizing blind spot discovery as a supervised learning problem introduces several challenges. First, each label received from the oracle provides a noisy signal indicating whether the corresponding agent observation is a blind spot. Since an agent's observation of the world may correspond to multiple real-world states, identifying whether an agent observation is a blind spot requires aggregating multiple labels. In addition, the noise in labels varies across different types of oracle feedback: for example, corrections clearly indicate whether an agent's action is acceptable in a state, whereas demonstrations only depict when the agent's and oracle's behaviors differ. Second, blind spots can be rare, and thus learning about them represents an imbalanced learning problem. Finally, oracle feedback collected through executions (including both corrections and demonstrations) violates the i.i.d. assumption and introduces biases into the learning process.

Our approach leverages multiple techniques to address the problems of noise and imbalance. Prior to learning, we apply expectation maximization (EM) to the dataset of oracle feedback in order to estimate noise in the labels and reduce noise through label aggregation. We also apply oversampling and calibration techniques to address class imbalance. Finally, we experiment with different forms of oracle feedback, including corrections, demonstrations, and randomly-selected states, to quantify the biases in different conditions.

We evaluated our approach using two game domains, and our findings indicate that blind spots can be learned more effectively with our method than baseline approaches, highlighting the benefit of reasoning about different forms of feedback noise. Further, the learned blind spot models are useful for selectively querying an oracle for help during real-world execution. Our evaluations also show that each feedback type introduces different biases that influence the blind spots the agent learns.

Overall, corrections are informative, as they provide direct feedback about the agent's actions and include the set of states induced by the agent's policy in the real world. Demonstrations, however, only provide information about whether the agent's actions deviate from or match the oracle's behavior – a noisier and less informative signal. Further, demonstration data includes states based on the oracle's policy, which can be quite different from the states the agent would likely visit during actual execution. Our findings indicate that the effectiveness of demonstrations varies: in some cases, demonstrations do not account for important errors that the agent may make, resulting in inadequate coverage of all blind spots; while in other cases, demonstration data is sufficient to enable the agent to avoid dangerous regions altogether.

Our initial formulation assumes optimal oracle demonstrations and an optimal simulator policy; we also include an analysis of how our method performs when these assumptions are relaxed. We found that with suboptimal demonstrations, the model can better separate safe and blind spot states because the distribution of labels from the oracle is more varied. With a suboptimal simulator policy, an agent identifies a larger number of states as blind spots, resulting in a more conservative agent. When both of these assumptions are relaxed, we can obtain equivalent – and occasionally better – performance than when using the original, optimal policies.

We also propose data augmentation techniques that improve performance for each feedback type. The disadvantage of corrections is that the agent never visits many important blind spot states, as the oracle turns the agent away beforehand. In order to enable the agent to safely learn about these blind spots, we include an augmentation that propagates unacceptable labels to expected future states without the agent

actually visiting them. Secondly, because the oracle only visits a limited part of the real world in demonstrations, we augment this data with corrections. By combining these data sources, we can improve performance compared with a model that only incorporates demonstrations.

The contributions in this chapter are six-fold: (1) formalizing the problem of discovering agent blind spots caused by representation incompleteness in reinforcement learning, (2) introducing a transfer learning framework that leverages human feedback to learn a blind spot map of the real world, (3) evaluating our approach across two domains, (4) assessing the biases of different types of human feedback, (5) analyzing the effect of suboptimal oracle and simulator policies on our approach, and (6) proposing modifications to our method to increase performance with demonstrations and corrections data.

2.2 Related Work

Here, we discuss prior work related to the problem of agent blind spot discovery in reinforcement learning. We first outline work in simulation to real-world transfer that is closely related to our problem; we then discuss relevant work in transfer and lifelong learning, which can involve high-level transfer between different tasks and domains. We also highlight work in supervised learning, in which more robust models are developed to better generalize when training and test distributions differ. Finally, we discuss techniques for outlier and novelty detection.

2.2.1 Safe simulation to real-world transfer

Reinforcement learning under safety constraints is an active research topic, particularly with regard to transfer from simulation environments to the real world [73, 149, 9]. Many prior works have focused on either developing realistic simulation environments with more robust training or on more cautious exploration within the real world. While these methods improve robustness and safety, they do not address scenarios in which the agent has a flawed representation that prevents it from

learning calibrated uncertainty estimates.

One approach to safe real-world execution is creating more realistic simulators that match the real-world setting as closely as possible. Shah et al introduced a photorealistic environment called AirSim for autonomous cars and drones [200]. CARLA [63] and TORCS [243] are car simulators that model different aspects of the real world. Chang et al introduced a dataset with realistic images for scene understanding, which could potentially be used for then learning how to act in the real world [45].

While these simulators provide realistic environments, agents still need to train and learn robust models that perform well in the real world. Domain randomization represents one method of developing such robust policies [225, 250, 166, 237]. It involves training the agent on many variants of a task or a wide set of parameters in order to better generalize to a real-world environment. Bousmalis et al used randomized synthetic data to improve a robot's performance of a grasping task in the real world [29]. Several other works have also optimized performance for specific areas in which an agent does not typically perform well, such as adversarial settings and worst-case scenarios [233, 191].

Another approach to improved real-world generalization uses progressive networks, which allows for sequential task learning [192]. In this method, columns are added to the network as new tasks are introduced, preventing the network from performing worse through task-specific fine-tuning. Another method of bridging the reality gap is to learn reusable skills that allow for quick adaptation [93, 106]. He et al considered a multi-task setting in which a robot used simulation data to learn latent skills and a policy conditioned on this space; the robot was then able to choose a sequence of these learned skills to perform unforeseen tasks in the real world [93]. Realistic simulators and robust training enable agents to be more capable of acting in the world, but there is still often a reality gap between simulation and the real world because not every nuance can be modeled or predicted [181].

Another method for safe transfer is cautious exploration during execution. Berkenkamp et al introduced an algorithm to safely obtain data to learn about the dynamics of a system with theoretical safety guarantees [25]. In work by Osband et al, an agent ex-

plores based on uncertainty estimates obtained through bootstrapping with random initialization [162]. Gal and Ghahramani interpreted dropout in neural networks as equivalent to a Bayesian model, which helps to obtain uncertainty estimates [72].

In a robotics scenario, these notions of uncertainty can be useful for safer real-world execution. In work by Kahn et al, a robot predicts its uncertainty as it acts within the world, and moves slowly to avoid high-speed collisions, while increasing its velocity within parts of the space where it has greater confidence in its actions [110]. Many other works have developed approaches for estimating model uncertainty using various techniques [133, 143, 139]. While these approaches are helpful for avoiding errors in real-world environments, they do not account for circumstances in which the agent has a limited representation of the world and its uncertainty estimates are subsequently misleading. The mistakes that can result from such a scenario will occur in areas where an agent has high confidence but is actually incorrect (blind spots).

2.2.2 Transfer and lifelong learning

Many approaches have improved the transfer of information across tasks [220, 163, 23], as tasks cannot be learned from scratch each time. Transfer learning aims to improve performance of a target task by leveraging knowledge from a source task. While this includes simulation to real-world transfer, it can also refer to transfer across different tasks and domains. Multi-task learning involves optimizing performance over a set of tasks by developing shared knowledge across those tasks. Lifelong learning is a combination of both transfer and multi-task learning that refers to a continual process of learning new tasks while retaining the quality of performance on prior tasks.

Many works in the field of transfer learning have focused on learning mappings between state and action spaces to enable Q-value function or policy transfer [221, 218]. Dai et al developed a model to learn a mapping by linking the feature space of a source task to the target feature space [56]. In work by Taylor and Stone, rules are summarized in the source task and transferred to the target [219]. Several researchers have also considered hierarchical approaches to RL that involve transferring subtasks across domains [120, 239, 167, 222]. Our setup is different from many prior transfer

learning scenarios, as they do not reason explicitly about the mismatch in the agent's representation and the true representation of the world [220, 19, 225, 23, 51].

Within lifelong learning literature, many works have introduced methods for quick transfer to new, sequentially introduced tasks [8, 7, 100]. Ruvolo and Eaton developed an approach that learns a shared basis, uses this library to transfer to new tasks, and revises the basis as additional tasks are introduced [193]; this approach performs well on classification and regression problems. Abel et al used state abstractions to capture important structure within the task to help during transfer [2]. Brunskill and Li performed a theoretical analysis of when learning options, or temporally extended actions, could be helpful in lifelong learning settings [33]. In work by Isele et al, transfer was improved considerably by learning relationships between task descriptors and task policies [99].

2.2.3 Generalization in supervised learning

Our work relates to supervised learning literature that trains models to be robust to examples outside of the training distribution. The presence of different training and test distributions during learning can also be referred to as *covariate shift*. One method of addressing this situation is a technique called dropout [209], which reduces overfitting by dropping units and connections while training a neural network. Many thinned networks are sampled and trained, and are then combined into a single network at test time. This process results in better generalization than use of neural nets without dropout across several classification datasets.

Another way to train more robust models that do not overfit is to use adversarial examples [215, 121]. Shaham et al developed a robust optimization approach in which a network is optimized over several perturbed examples in an uncertainty set, $\mathcal{U}$ [201]. Athalye et al generated a more informative set of adversarial examples based on transformations of physical objects, which identifies 3D adversarial objects [13]. While these approaches to generating examples increase the robustness of the learned model, they do not help when the set of possible examples is incomplete. In work by Tramèr et al, models are trained against *black-box* adversaries that have no

information about the model's inner workings, rather than *white-box* adversaries that have full knowledge of the algorithm and model parameters [230].

While adversarial examples represent one way of identifying model errors, it is difficult to construct these examples before deploying a model in the open world. Even with extensive adversarial training, a gap can still exist between the data used for training the model and the true distribution of data. Lakkaraju et al introduced a method for identifying regions of unknown unknowns in discriminative classifiers when deployed into the open world [122]. In this approach, data points are clustered in an unsupervised manner, followed by a multi-arm bandit algorithm (with each cluster representing an arm) to efficiently identify regions of the feature space in which the classifier is most likely to make mistakes. It is not straightforward to apply this approach to RL, however, because examples (states) are no longer i.i.d., as in supervised learning. In RL, states are visited according to a distribution induced by executing either the learned policy or the oracle's policy. There can also be multiple acceptable labels (actions) for each state, rather than a single "correct" label. Finally, certain mistakes in the real world can be catastrophic, necessitating risk-sensitive classification to prioritize identifying rare blind spot states [248].

2.2.4 Novelty and outlier detection

As indicated in previous work, outlier and novelty detection are related but not directly applicable to blind spot discovery [44, 88, 38]. Outlier detection aims to identify instances within the training data that are far from the rest of the examples. Novelty detection assumes that the training data does not contain outliers, and instead involves identifying whether a new instance not seen in training deviates significantly from the training set. One reason blind spots differ from outliers is that they are not rare instances, but systematic regions within the agent's representation where the training environment does not match the testing environment, leading to costly errors. In our work, we present a method for efficiently identifying these regions through oracle feedback. Another difference between outliers and blind spots is that detecting outliers requires access to the correct set of features in order to distinguish novel ex-

amples [140, 141]; in our work, the available features are insufficient for differentiating blind spots from safe regions.

Several prior methods have used kernel techniques to identify outliers [198, 96, 37, 28]. Hoffman introduced a novelty measure by using kernel PCA to map points into a higher-dimensional space. The system performed PCA to obtain independent components and computed the reconstruction error based on this space [96]. Bodeshim et al mapped training examples with a specific class into one point [28]; this projected subspace has zero variance within each class and can be used to predict the novelty of new instances through a distance measure.

In work by Blouvshtein and Cohen-Or, outliers were identified by reasoning about distances between data points [27]. A given distance is predicted to be an outlier if it breaks the triangle inequality for many triangles. While many of these methods rely upon distances to detect outliers, one challenge is that many examples look like outliers in high-dimensional spaces. Radovanović et al demonstrated that when attributes are not noisy, outliers can be identified as more pronounced within the high-dimensional space [176]. However, these works are, again, conditioned upon the availability of features to identify novel or irregular instances, which we do not assume in our own work.

2.3 Problem Formulation

We formulate the discovery of agent blind spots in reinforcement learning as a transfer learning problem, in which an agent is trained in a simulation environment M_{agent_sim} and is deployed and evaluated in a real-world environment M_{real}. The simulator is modeled as a Markov Decision Process (MDP): $M_{agent\_sim} = \{\mathcal{S}_{agent}, \mathcal{A}, T_{sim}, R_{sim}\}$. The state space $\mathcal{S}_{agent}$ defines all possible states in the simulation world. The representation of this state space can be manually designed or learned. The action space of the agent is defined as $\mathcal{A}$. The transition function $T_{sim} : \mathcal{S}_{agent} \times \mathcal{A} \times \mathcal{S}_{agent} \rightarrow \mathbb{R}$ models the dynamics in the simulation environment, which is an approximation of the real world dynamics based on the agent's representation $\mathcal{S}_{agent}$. The reward function

$R_{sim} : \mathcal{S}_{agent} \times \mathcal{A} \rightarrow \mathbb{R}$ specifies the reward the agent receives for taking an action from a specific state. The agent trains in this world and learns a policy $\Pi_{sim} : \mathcal{S}_{agent} \rightarrow \mathcal{A}$, which maps states in $\mathcal{S}_{agent}$ to actions $\mathcal{A}$.

The real-world environment is defined as a Partially-Observable Markov Decision Process (POMDP) because the agent does not observe the true state of the world and instead receives an observation of this true state: $M_{real} = \{\mathcal{S}_{real}, \mathcal{A}, T_{real}, R_{real}, \Omega, O\}$. In this environment, $\mathcal{S}_{real}$ defines all possible real-world states, and $\Omega = \mathcal{S}_{agent}$ defines the space of all possible agent observations. The agent observes states of the world through its representation defined in simulation. While the state space $\mathcal{S}_{agent}$ was sufficient for learning in the simulator, it lacks important features for acting safely in the real world so the agent's observation space is a subset of the real-world state space $\mathcal{S}_{agent} \subset \mathcal{S}_{real}$. The observation function $O : \mathcal{S}_{real} \times \mathcal{A} \rightarrow \mathcal{S}_{agent}$ is a deterministic mapping between the true state and the agent's observation of that world based on $\mathcal{S}_{agent}$. The action space $\mathcal{A}$ is identical to that in training. The true transition and reward functions T_{real} and R_{real} in the real world are based on $\mathcal{S}_{real}$ rather than $\mathcal{S}_{agent}$, so these functions do not match the approximations in simulation.

For each real-world state $s_{real} \in \mathcal{S}_{real}$, the agent receives a flawed observation of the world in terms of its representation $o_{agent} \in \Omega = \mathcal{S}_{agent}$. Because $\mathcal{S}_{agent} \subset \mathcal{S}_{real}$, multiple states look identical to the agent, also known as perceptual aliasing [109]. The agent has access to an oracle or human that observes the true state of the world $s_{real} \in \mathcal{S}_{real}$. The agent can only reason within the agent's observation space $\mathcal{S}_{agent}$, while the oracle has access to the true real-world state space and provides feedback based on $\mathcal{S}_{real}$.

Our goal is to use the agent's learned policy from simulation and a limited budget, B, of feedback from an oracle, O, in order to learn a blind spot model of the real world that indicates the probability that each agent observation is a blind spot $\mathcal{S}_{agent} \rightarrow [0, 1]$. This learned model can then be used during real-world task execution to query a human for help at states with a high probability of being blind spots. Prior works have investigated the use of human feedback for guiding agents in RL tasks [10, 119, 195, 50, 84], but we use oracle feedback to learn a blind spot model of the

real world rather than to learn a policy.

An oracle, $O = \{f_{acc}(s, a), \pi_{real}\}$, is defined by two components: an acceptable function $f_{acc}(s, a)$ and an optimal real-world policy π_{real}. The acceptable function $f_{acc}(s, a)$ provides direct feedback about the agent's actions by returning 0 if action a is acceptable in state s, and 1 otherwise. In our experiments, we simulated $f_{acc}(s, a)$ by defining acceptable actions as those with values within some δ of the optimal action value for that state; however, $f_{acc}(s, a)$ can be defined in many ways. The optimal policy π_{real} is used when the oracle is providing demonstrations within the real world. In practice, oracles can be humans or other agents with more expensive and/or complementary mode sensors (e.g., lidar cannot see color but can sense 3D shapes, while cameras can detect color but not 3D shapes).

An observation in the agent's representation $o_{agent} \in \mathcal{S}_{agent}$ is defined as a blind spot ($B(o_{agent}) = 1$) if:

$$\exists s_{real} \in \mathcal{S}_{real} \text{ s.t. } f_{acc}(s_{real}, \pi_{sim}(o_{agent})) = 1. \tag{2.1}$$

In other words, blind spots are observations in the agent's representation where the agent's learned action is unacceptable in at least one real-world state that maps to it. Intuitively, if two states appear identical to the agent, and the agent takes an unacceptable action in either of them, it should mark its flawed observation as a blind spot.

The agent's objective is to use the learned policy π_{sim} and a limited budget of B labels from the oracle $O = \{f_{acc}(s, a), \pi_{real}\}$ in order to learn a blind spot model, $M = \{C, t\}$. The classifier $C : \mathcal{S}_{agent} \rightarrow \Pr(B(\mathcal{S}_{agent}) = 1)$ predicts, for each agent observation $o_{agent} \in \mathcal{S}_{agent}$, the probability that o_{agent} is a blind spot in the real-world environment, while the probability threshold t specifies the cutoff for classifying an agent observation as a blind spot.

2.3.1 Agent Observations

A perfect label for learning the blind spot map would be the tuple $< o^i_{agent}, l^i_p >$, such that l^i_p is 1 when o^i_{agent} is a blind spot – a real-world state corresponding to o^i_{agent} exists where $\pi_{sim}(o^i_{agent})$ is not acceptable – and l^i_p is 0 otherwise. Since the oracle and agent have different representations of the world, the oracle's actions are based on $\mathcal{S}_{real}$, not $\mathcal{S}_{agent}$. Thus, labels collected from the oracle are associated with *state representation (SR) noise* as the agent and the oracle perceive the world differently. In addition, some forms of oracle feedback may provide weaker information about the quality of an agent's actions: instead of providing information about whether an action is acceptable, the feedback may indicate when the agent's and oracle's actions differ, referred to as *action mismatch (AM) noise.* We describe both types of noise in detail below.

State representation noise

State representation (SR) noise occurs because the agent and oracle are operating within two different representations. The agent's representation is limited, and has missing features that cause the agent to observe many distinct real-world states as identical to one another. When the oracle provides a label about a real-world state, the agent cannot disambiguate this label from labels provided for other real-world states that look identical and map to the same agent observation, resulting in state representation noise.

Let s^i_{real} be a real-world state and o^i_{agent} be the agent observation s^i_{real} corresponds to. A label tuple from the perspective of the oracle is defined as a tuple, $< s^i_{real}, l^i_a >$, where $l^i_a \in \{0, 1\}$ is the resulting label such that $l^i_a = f_{acc}(s^i_{real}, a^i_{agent})$ and $a^i_{agent} = \pi_{sim}(o^i_{agent})$. However, due to a limited state representation, the label tuple from the agent's perspective is $< o^i_{agent}, l^i_a >$. When many real-world states map to the same agent observation in this manner, the agent receives several noisy labels for a single observation. For example, if the agent takes an acceptable action in one real-world state and an unacceptable action in a different state that the agent interprets as

identical to the first, it will receive two labels for this agent observation: a 0 and a 1.

To return to our definition of blind spots: if the agent receives even one unacceptable label for any real-world state, the corresponding agent observation is automatically identified as a blind spot. Thus, receiving an unacceptable label is a perfect signal that the corresponding agent observation is a blind spot. However, if the agent receives many acceptable labels, the agent cannot mark the observation as "safe" (i.e., not a blind spot) because there may be other real-world states that map to this one where the agent's action would be unacceptable. The main property of SR noise is that receiving many "safe" labels does not guarantee that the corresponding agent observation is safe.

Action mismatch noise

The formulation described thus far assumes that the oracle provides direct feedback about the agent's actions using the acceptable function $f_{acc}(s, a)$. Another form of feedback allows the oracle to provide demonstrations using π_{real} while the agent simply observes, which might be of lower cost to the oracle than directly monitoring and correcting the agent's actions. In this case, when feedback about the agent's actions is not given directly, an additional form of noise is introduced: action mismatch (AM) noise. If the oracle takes action a_i but the agent planned to take a_j, this could be indicative of a blind spot because the agent is not following the optimal action. However, two actions can both be acceptable in a state, so a mismatch does not necessarily imply that the given agent observation is a blind spot. The main property of AM noise is that noisy unacceptable labels are generated for safe regions, and the agent should reason about this noise to avoid being overly conservative and labeling safe areas as blind spots.

2.4 Approach

We now present a framework for learning blind spots in RL, as depicted in Figure 2-2. The pipeline includes a data collection phase, during which the agent receives

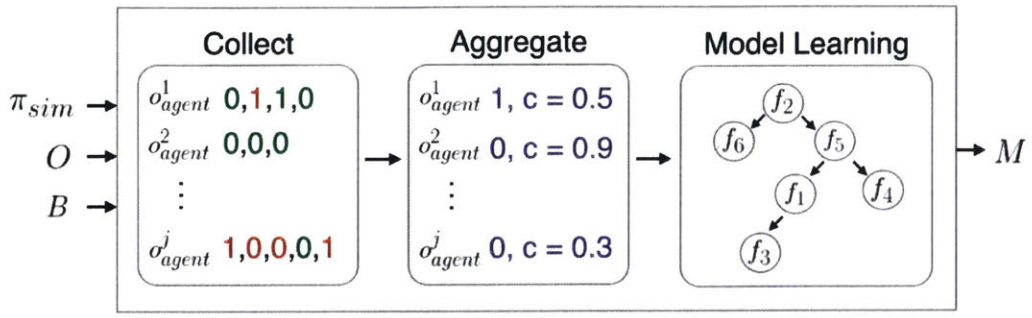


Figure 2-2: The full pipeline of our approach. To learn a blind spot model, the agent first collects data from an oracle using one of the feedback types, then aggregates noisy labels in order to predict whether each agent observation is a blind spot. Finally, a classifier is trained with this data to generalize over the full space of agent observations.

data from an oracle through various forms of feedback. Since each feedback type is associated with noise, we introduce a label aggregation step that estimates the noise in the labels using EM and predicts the true label of each visited agent observation through aggregation of labels. To generalize from visited agent observations to unseen regions, we use supervised learning. This learning step assumes that blind spots do not occur at random, but instead correlate with existing features that the agent has access to.

Since blind spots are typically rare within data, learning about them represents an imbalanced learning problem. To address this, we first oversample the blind spot examples and then perform calibration in order to correct estimates of the likelihood of blind spots. Calibrated estimates are important for our domain, as they can be used to decide whether to request oracle help. This allows the agent to accurately trade off the likelihood of error with the cost of querying the oracle in execution.

2.4.1 Data collection

In order to develop a system that can learn blind spots in a real-world environment, we first must collect data from an oracle. This oracle can either provide feedback about the agent's actions directly through the acceptable function $f_{acc}(s, a)$ or provide a demonstration of the optimal action, which the agent can observe, using π_{real}. We

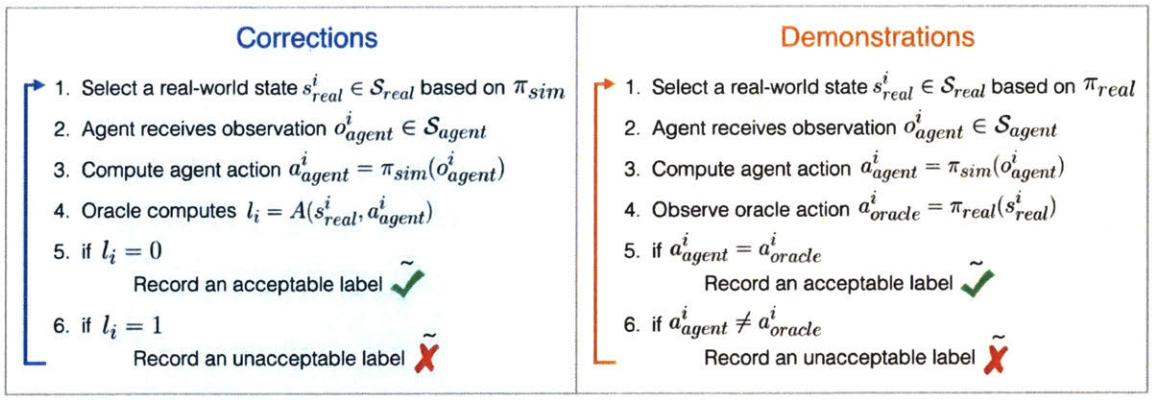


Figure 2-3: This figure outlines the process of data collection with corrections and demonstrations serving as feedback methods. With the use of corrections, the agent receives direct feedback about its own actions through the acceptable function; with demonstrations, the agent obtains noisy signals of acceptability by observing action matches and mismatches.

now discuss specific forms of oracle feedback and the type of noise (SR and/or AM) they each contain.

Corrections (C)

The first feedback type is corrections (C), in which the oracle monitors and corrects the actions of the agent as it acts within the real world. As shown in Figure 2-3, the process for data collection is as follows: The agent first visits a state $s^i_{real} \in \mathcal{S}_{real}$ in the real world based on its simulator policy, π_{sim}. The agent receives an observation of this state o^i_{agent} and computes the action it plans to take, $a^i_{agent} = \pi_{sim}(o^i_{agent})$. The agent then receives a feedback label, $l_i = f_{acc}(s^i_{real}, a^i_{agent})$, from the oracle at that state, where $l_i = \{0, 1\}$. The agent records an acceptable label if $l_i = 0$, and an unacceptable label if $l_i = 1$. Given that many real-world states appear identical to the agent and thus map to the same agent observation, the resulting dataset, $D = \{(o^i_{agent}, [l_1, ..., l_k])\}$, has many noisy labels for each agent observation due to state representation (SR) noise. (Corrections contain no AM noise because the oracle directly provides feedback about the agent's actions through the acceptable function.)

Demonstrations-action mismatch (D-AM)

Corrections can be expensive to obtain because doing so involves constant monitoring of the agent. Demonstrations represent an alternative form of feedback that involves the oracle acting in the real world using π_{real}. Figure 2-3 depicts the process of obtaining data from demonstrations. In this case, the oracle selects states based on the policy π_{real} in the real world. The agent receives an observation o^i_{agent} of the chosen real-world state s^i_{real}, and computes the action it would take given that observation $a^i_{agent} = \pi_{sim}(o^i_{agent})$. The agent then observes the oracle's action, $a^i_{oracle} = \pi_{real}(s^i_{real})$. In this feedback type, the agent does not learn about the acceptability of its own action; instead, it compares its action to that of the oracle's and uses action matches and mismatches as proxies for acceptability. If the agent's action matches the oracle's action, $a^i_{agent} = a^i_{oracle}$, the agent knows its action is acceptable to perform in the real world. However, if the actions do not match $a^i_{agent} \neq a^i_{oracle}$, this does not necessarily mean that the agent performed an unacceptable action, as multiple acceptable actions may exist for that state. The agent notes a noisy unacceptable label for all action mismatches. This condition results in a dataset, $D = \{(o^i_{agent}, [l_1, l_2, ..., l_k])\}$, with noisy unacceptable labels; thus, demonstrations-action mismatch (D-AM) includes both SR noise and AM noise.

Demonstrations-acceptable (D-A)

Corrections and demonstrations-action mismatch differ in two ways: distribution of data and AM noise. We want to decouple these effects; thus, we include a condition without AM noise. This feedback type, demonstrations-acceptable (D-A), is collected similarly to D-AM and followed by a review period in order to get direct feedback about the agent's mismatched actions. For all states with action mismatches, the agent queries the oracle function, $f_{acc}(s^i_{real}, \pi_{sim}(o^i_{agent}))$, to resolve action mismatch ambiguities. Thus, all noisy unacceptable labels become either "safe" labels (because the agent's action, while different from the oracle's, is acceptable) or true "unacceptable" labels (which confirm that the given action is unacceptable). While AM noise is

Table 2.1: This table outlines the feedback types we consider in this work. We indicate the types of noise each contains and whether each feedback can contain data bias.

Feedback Type	SR Noise	AM Noise	Bias
Corrections (C)	✓	-	✓
Demonstrations-action mismatch (D-AM)	✓	✓	✓
Demonstrations-acceptable (D-A)	✓	-	✓
Random-action mismatch (R-AM)	✓	✓	-
Random-acceptable (R-A)	✓	-	-

resolved in this condition, SR noise still exists. The only difference between D-A and C is that D-A generates states according to π_{real}, while C generates states according to π_{sim}.

Random-acceptable (R-A)

Demonstrations and corrections are forms of feedback that allow a human to naturally provide information to an agent [50]. However, the visited states are biased according to the policy that generated them; furthermore, collecting data based on policies violates the i.i.d. assumption of supervised learning. To evaluate the effect of this bias, we include two baseline approaches in which states are randomly selected. In the random-acceptable (R-A) condition, states are randomly sampled in the real world. For each state, the oracle provides a label using the acceptable function, and this label is recorded. The data collection process is identical to D-A, except that states are now chosen randomly.

Random-action mismatch (R-AM)

In the random-action mismatch (R-AM) condition, states within the real world are randomly sampled, and the oracle computes its action for a given state based on π_{real}. The agent records an action match as an acceptable label, and a mismatch as a noisy unacceptable label. (R-AM is identical to D-AM, except for the random sampling of states.)

Summary of feedback types

All feedback types result in a dataset, $D = \{(o^i_{agent}, [l_1, ..., l_k])\}$, which includes a set of noisy acceptable and unacceptable labels for each agent observation. For feedback types lacking AM noise, when an unacceptable label is given by the oracle for any real-world state, the matching agent observation is marked as a blind spot. In other words, for each $(o^i_{agent}, [l_1, ..., l_k]) \in D$, if $\exists l_i = 1$, o^i_{agent} is a blind spot. For an agent observation, if all corresponding labels are acceptable, the agent should reason about the likelihood of the observation being a blind spot or safe based upon the number of labels collected. If the feedback type contains AM noise, the agent must reason about noisy unacceptable labels when learning to identify blind spots.

D-AM is the most difficult type of feedback to reason about because it includes SR noise, AM noise, and bias, while corrections represents the most informative feedback type because it provides direct feedback about the agent's policy. However, we expect demonstrations to be easier to obtain than corrections because the oracle simply acts in the world according to its policy. In Section 2.6, we discuss the biases and tradeoffs of using demonstrations versus corrections data and how both of these compare to the baselines that involve randomly sampled data. Table 2.1 summarizes all feedback types and the noise/bias observed in each.

2.4.2 Aggregating Noisy Labels

Across all conditions, the collected data has many noisy labels that must be aggregated. Through data collection, we obtained a dataset of noisy labels, $D_n = \{(o^i_{agent}, [l_1, l_2, ..., l_k])\}$, for regions the agent has visited; these labels are noisy due to SR and AM noise. The model must reason about these noisy labels obtained from the oracle in order to determine whether the true label of the agent's observation is "blind spot" or "safe."

For label aggregation, we use the Dawid-Skene algorithm (DS), which is a popular approach for addressing label noise in data collection [57]. We prefer this approach because it has a small number of parameters to estimate, works well over sparse data

sets, and has been shown to consistently work well across problems [203]. DS takes a dataset of noisy labels as input, and its goal is to predict the true labels of instances by estimating the prior distribution of the data (the ratio of blind spots vs. safe regions) and the confusion matrix, which is the noise model between noisy labels (acceptable vs. unacceptable) and true labels (blind spot vs. safe). The algorithm is unsupervised; it uses EM in its estimation. First, it initializes true labels of instances by averaging the labels for each agent observation. Then, it estimates the confusion matrix based on the initialized true labels. It uses the confusion matrix to re-estimate the labels by weighting based on the estimated noise. The algorithm iterates until convergence.

Different forms of feedback are associated with particular noise types that can be used to inform the aggregation approach. For example, for all feedback types without AM noise (C, D-A, R-A), safe regions never receive unacceptable labels because when an agent observation is truly safe for the agent, all corresponding real-world states will also be safe. Also, since the agent receives direct signals based upon its actions, it will not receive noisy labels from action mismatches. We can modify DS to leverage this property by constraining the confusion matrix that DS attempts to estimate. Specifically, one row of the confusion matrix (corresponding to safe regions) can be constrained to have 100% acceptable labels and 0% unacceptable labels. The left-hand side of Figure 2-4 depicts the parameters that DS must learn for feedback types without AM noise: the ratio of acceptable and unacceptable labels for blind spot regions and the prior distribution of safe and blind spot regions.

In the presence of AM noise, the agent can take a different action from the oracle while remaining safe; we denote this action mismatch as a noisy unacceptable label. As both safe and blind spot regions can now receive either label ("acceptable" or "unacceptable"), there is no structure within the data to constrain the noise estimation problem. Thus, for feedback types with AM noise (D-AM, R-AM), we use the original DS algorithm to learn all parameters, as shown in Figure 2-4. The output of aggregation is a dataset, $D_a = \{(o^i_{agent}, \hat{l}, c)\}$, where $\hat{l}$ represents the estimated true label and c is the associated confidence.

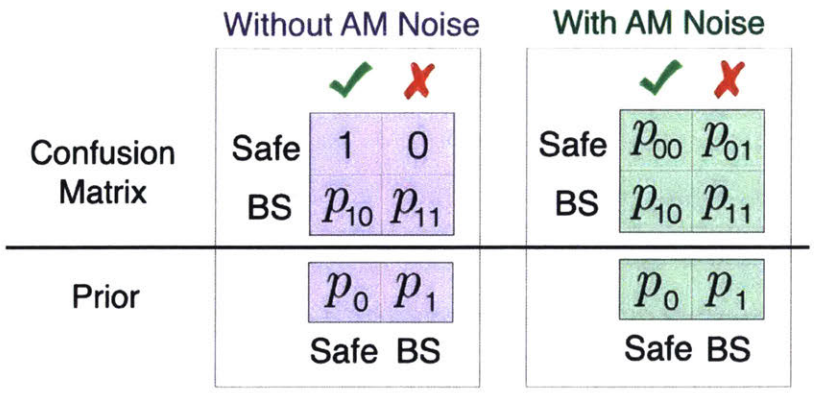


Figure 2-4: This figure depicts the way in which we constrain the Dawid-Skene algorithm for feedback types with and without AM noise. When there is no AM noise in the data collected from the oracle, there will not be any unacceptable labels for safe regions, allowing the confusion matrix to be constrained during learning. There is no such structure when the data collected includes AM noise, and thus we use the original DS algorithm.

2.4.3 Model Learning

With estimated true labels, we train a supervised learner to predict which observations in the agent's representation are likely to be blind spots in the real world. Due to the relative rarity of blind spot regions, the major challenge in model learning is class imbalance. With only a few blind spots, the model will learn to predict all regions as safe, which can be extremely dangerous. To deal with class imbalance, we oversample blind spot instances to generate balanced classes in the training data, and then calibrate the model to provide better estimates.

The full model-learning process is as follows: a random forest (RF) classifier is trained with data from aggregation $D_a = \{(o^i_{agent}, \hat{l}, c)\}$, weighted according to c. The output is a blind spot model $M = \{C, t\}$, which includes a classifier C and a threshold t. To learn M, the system performs a randomized hyperparameter search over RF parameters, runs three-fold cross-validation with oversampled data for each parameter configuration, and obtains an average F1-score. The hyperparameters with the highest average F1-score are chosen to train the final model.

For the final training round, we reserve 30% of the full training data for calibration. We oversample the rest of the data and train an RF classifier using the best parame-

ters. In order to calibrate the model after training on oversampled data, we vary the threshold that specifies a cutoff probability for classifying an agent observation as a blind spot. For each possible threshold t, the system measures the percentage of blind spots predicted by the model on the held-out calibration data, and chooses t such that the prior of blind spots on the calibration set matches the prior in the training data, as estimated by the DS approach. The final output, $M = \{C, t\}$, includes the learned RF classifier, C, and the threshold, t.

2.5 Experimental Setup

In order to evaluate our system's ability to identify blind spots, we conducted experiments within two domains. For each domain, we used one version of the domain to train the agent and a modified version to simulate a real-world setting that did not match the training scenario.

2.5.1 Domains

The first domain is a modified version of the game Catcher, in which the agent is required to catch falling fruits. In Catcher, a fruit starts from the top of the screen at a random x location, then falls straight down at a constant speed. The agent controls a paddle at the bottom of the screen that can remain stationary, move left, or move right. The state of the game in the simulation environment is represented as $[x_p, x_f, y_f]$, where x_p is the x location of the player and (x_f, y_f) represents the fruit's location. In simulation, the reward is proportional to the player's x distance away from the fruit, or $W - |x_p - x_f|$, where W represents the width of the screen.

For this domain, the real world is split into two regions: the left-hand side, which appears exactly like simulation, and the right-hand side, which has a probability p of being like simulation, and a probability $1 - p$ that a "bad" fruit will fall instead of the original fruit. The agent receives a higher reward for moving away from bad fruits, denoted by $|x_p - x_f|$. An additional high negative reward, -100, is given when $x_p = x_f$, because the space directly under a bad fruit represents a high-danger region.

The agent does not have the fruit type feature in its representation, so it lacks the "true" representation of the real world. Without the ability to distinguish between fruit types, the agent can never learn the optimal policy.

The second domain is a variation of the game Flappy Bird. The goal is to fly a bird through the space between two pipes. The state in simulation is represented by $[y_t, y_b, y_a, v_a, \Delta x]$, where y_t, y_b, and y_a represent the y locations of the top pipe, bottom pipe, and agent, respectively; v_a is the agent's velocity; and Δx is the x distance between the agent and the pipe. The agent can either move up or take no action, in which case gravity begins to pull the bird downward. The simulation environment includes high pipes and low pipes, and the agent must learn to fly high above the ground and then swoop downward in order to pass between both low and high pipes. The agent receives +10 for getting past a pipe, -10 for crashing, and +0.1 any time it flies above a certain threshold (to encourage flying at a high altitude). In the real-world environment, pipes are composed of different materials (copper and steel), which the agent is unable to observe. Copper pipes close to the ground can cause a heavy wind to pass through, requiring the agent to be cautious and fly low, while steel pipes do not cause heavy winds, and thus the agent should fly at a high altitude before passing through them. The reward function remains the same for the real world setting, except that the agent receives +0.1 for flying below a specific threshold (to encourage flying low) for copper pipes, and -100 when it flies high, because this represents a high-danger region. Without knowledge of the pipe's material, the agent is unable to learn the optimal policy for both pipes.

2.5.2 Oracle Simulation

We assume access to an oracle $O = \{f_{acc}(s, a), \pi_{real}\}$ that provides feedback to the agent. We simulate this oracle by obtaining an optimal policy π_{real} for the target task and constructing an acceptable function $f_{acc}(s, a)$ that specifies which actions are acceptable in each state. This function depends upon the domain, as well as how strict the given oracle is. A strict oracle may only consider optimal actions to be acceptable, while a lenient oracle may accept most actions except those that would

lead to significantly lower Q-values.

To simulate different acceptable functions, we first trained an agent on the true real-world environment to obtain the optimal real-world Q-value function, Q_{real}. We then computed, for each state $s_{real} \in \mathcal{S}_{real}$, the difference in Q-values between the optimal action and every other action: $\Delta Q^i_{s_{real}} = Q_{real}(s_{real}, a^*) - Q_{real}(s_{real}, a_i), \forall a_i \in \mathcal{A}$. The set of all Q-value deltas, $\{\Delta Q^i_{s_{real}}\}$, quantifies all possible mistakes the agent could make.

The deltas are sorted in ascending order from least-dangerous to costliest mistakes, and the model chooses a cutoff delta value δ based on a specified percentile p at which to separate acceptable and unacceptable actions. This cutoff value is used to define the acceptable function in an experimental setting – and, consequently, the set of blind spots (agent observations with at least one unacceptable action in a real-world state mapping to it) for the task. When $f_{acc}(s, \hat{a})$ is queried with agent action $\hat{a}$, the oracle computes the difference: $\Delta Q_{\hat{a}} = Q_{real}(s, a^*) - Q_{real}(s, \hat{a})$. If $\Delta Q_{\hat{a}} < \delta$, action $\hat{a}$ is acceptable in state s; otherwise, the action is unacceptable. An acceptable function $f_{acc}(s, a) = \{Q_{real}, p\}$ is defined by the target Q-value function Q_{real} and a percentile p for choosing the cutoff. A high p value simulates a lenient oracle, resulting in a greater number of acceptable actions and fewer blind spot states. Consequently, more AM noise exists, because if the oracle accepts the majority of actions, there is a high chance that the agent will subsequently take an acceptable action that differs from the action of the oracle. A low p value simulates a strict oracle, because even actions with Q-values only slightly lower than the optimal will still be considered blind spots. This results in less AM noise, because deviation from the oracle is a strong indicator that the agent's action is truly unacceptable.

2.5.3 Baselines

The first baseline is a majority vote (MV) aggregation method for the noisy labels. For each state, MV takes the label that appears most frequently as the true label. The second baseline is all labels (AL), which uses no aggregation and simply passes all data points to a classifier. The model learning is the same for both our method

and the baselines, which we used to assess the benefit gained from using DS for aggregation.

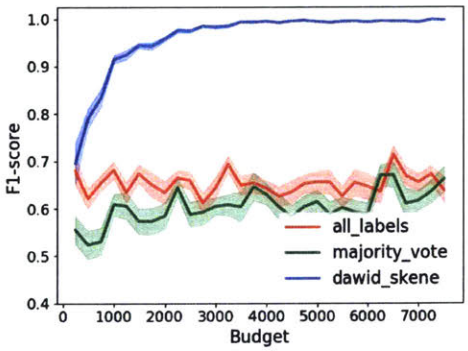
We report the baselines' performance when predicting blind spots based on the F1-score to assess both the precision and recall of the predictions, as well as the accuracy of the estimates of the likelihood of blind spots. We also compare results obtained with a strict versus a lenient oracle. The strict oracle was chosen such that only the optimal action was acceptable (no associated percentile p); for the lenient oracle, we used $p = 0.95$ for the Catcher domain and $p = 0.7$ for the Flappy Bird domain.

2.6 Results

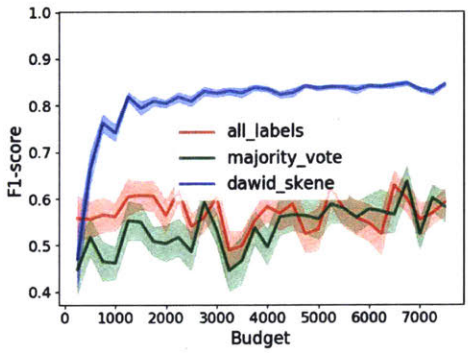
Here, we present the results of our approach in both domains. We observed better performance with our method compared with existing baselines, and also noted that different forms of feedback induced biases within the data that affected the learned blind spot model.

2.6.1 Benefits of aggregation

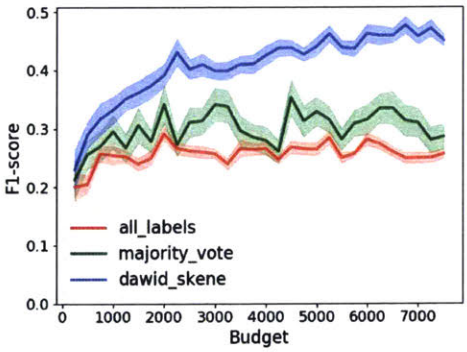
We first compared the performance of our approach, which uses DS for aggregation, to existing baselines: majority vote (MV) and all labels (AL), on the Catcher domain. In this section, we focus on feedback types with AM noise, because DS provides the greatest benefit when noise cannot be easily recovered by simple techniques. We present blind spot prediction performance for states visited during data collection, as this demonstrates the difference between our approach and the baselines with regard to the ability to estimate and reduce noise in the training data. As depicted in Figure 2-5, we varied the number of oracle labels received by the agent (the budget) and reported the resulting F1-scores, which were weighted according to the "importance" of states (represented by how often states were visited by π_{sim}). We ran the full experiment 25 times, and plotted the average performance along with standard error bars. We further compared all of the approaches against one another when applied



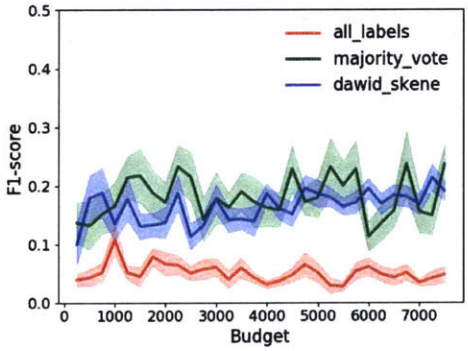
(a) R-AM with a strict oracle



(b) R-AM with a lenient oracle



(c) D-AM with a strict oracle



(d) D-AM with a lenient oracle

Figure 2-5: A comparison between our approach and baseline methods using random and demonstration data and varying oracles within the Catcher domain.

with a strict versus a lenient oracle.

For the randomly sampled data, DS performed much better than MV and AL with both strict and lenient oracles due to the uniformity of labels across all observations. DS could thus recover the prior and confusion matrix that generated this data, while MV predicted most regions to be safe because safe signals were much more common. AL exhibited lower accuracy because it did not aggregate the labels, which resulted in a poor prior estimate of blind spot regions.

Performance dropped overall with demonstration data (compared to random data), since feedback was biased based on the oracle's policy, preventing the system from learning about some blind spots that the agent would face during real-world task execution. Despite the decrease in performance, DS still performed well compared with both MV and AL in the presence of a strict oracle. With a lenient oracle, many safe regions were associated with unacceptable labels (due to action mismatch noise) that an unsupervised learning method such as DS could not completely recover. Nevertheless, DS performed equally well to MV, and much better than AL.

Overall, DS performed well compared with the baselines; however, performance dropped when states were sampled in a biased form rather than randomly, and also with a lenient oracle rather than a strict one. We observed similar trends in the Flappy Bird domain with regard to the benefit of DS versus baselines. We discuss details of the effect of feedback types and resulting biases across the two domains in Section 2.6.3.

2.6.2 Effect of feedback type on classifier performance

Next, we evaluated the best-performing approach (learning with DS) while varying the oracle feedback type. We evaluated the classifier on states observed in oracle feedback, which measures the ability of DS to recover true labels from noisy ones; and on unseen data, which highlights the ability of the classifier to generalize to unvisited regions. We report F1-scores in Table 2.2 for each condition.

The results show that learning from random data performed well across conditions, regardless of whether the condition included SR noise (R-A) alone or both SR and

Table 2.2: The effect of feedback type on classifier performance in the Catcher domain, reported as F1-scores.

	Strict		**Lenient**	
Feedback Type	Seen	Unseen	Seen	Unseen
R-A	0.996	0.994	0.825	0.700
R-AM	0.997	0.993	0.837	0.833
D-A	0.476	0.453	0.084	0.152
D-AM	0.487	0.477	0.209	0.274
C	0.478	0.461	0.636	0.520

AM noises (R-AM). R-AM performed well in these cases, despite the presence of AM noise, because DS was able to recover the labels for randomly sampled data when the labels were distributed uniformly across all agent observations. However, the performances of both R-A and R-AM dropped when the oracle was lenient, as there were fewer blind spots to learn from.

Our findings also indicate that the correlated nature of observations from demonstrations and corrections overall reduced the performance of classifiers compared with random data feedback types. In the strict oracle case, the performances of D-A, D-AM, and C were comparable because AM noise was low, resulting in similar feedback obtained from corrections and demonstrations, because the monitoring oracle that corrected the agent would redirect the agent any time it deviated from the optimal. On the other hand, a lenient oracle would only correct an agent if an action would be very dangerous, resulting in a more informative state distribution than that obtained through demonstrations. In this case, both versions of demonstrations failed to collect observations about major blind spot regions. Furthermore, D-A yielded few unacceptable labels due to the absence of AM noise; this hurt the prior estimate and made it difficult for the system to learn an accurate classifier. Thus, we report that D-A's performance was even worse than that of D-AM for this scenario.

2.6.3 Effect of feedback type on oracle-in-the-loop evaluation

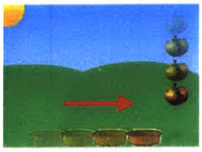
The ultimate goal of our work is to use limited feedback to learn a blind spot model of the real world that could enable an agent to act more intelligently. Ideally, the agent

could use this model to selectively query for human help, avoiding costly mistakes without overburdening the human.

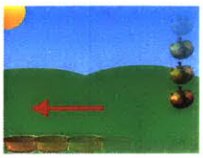
The next set of results evaluates the effectiveness of the learned model in oracle-in-the-loop (OIL) execution. In this case, the agent executes actions in a real-world environment using the source policy. When the learned model predicts a state to be a blind spot using the learned calibrated threshold, the agent queries an oracle for the optimal action, then takes this provided action and resumes acting according to its policy in the world. We compared our method of querying the oracle based on the learned model to an insufficiently cautious agent that never queried and an overly conservative agent that always queried.

Figure 2-6 shows that in both domains, demonstrations and corrections provided different feedback, and thus introduced separate biases into the data. Corrections provided direct feedback about the actions the agent would take, while demonstrations followed the policy of an optimal oracle. In the Catcher domain, an optimal oracle moved toward good fruits and away from bad fruits; an agent trained in the source task with only good fruits learned to move closer to all fruits. In Figures 2-6a and 2-6b, the fruits represent the movement of a bad fruit, and the agent moving at the bottom represents the feedback bias. Demonstrations provided observations about states that were far away from bad fruits, while corrections provided observations about states closer to bad fruits. In the Flappy Bird domain, the agent was required to be careful and fly low around copper pipes; as depicted in Figures 2-6c and 2-6d, a demonstration would show the agent flying low for a copper pipe, while a correction trajectory would allow the agent to fly slightly higher and correct only before it went too far. This provides information about the more informative states the agent would likely visit.

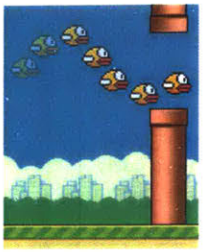
Figure 2-7 depicts the performance of our model on OIL evaluation with different feedback types. We report performance as the number of oracle labels (budget) increases. The left y-axis shows the reward obtained in the real-world environment by an agent that used the model to query compared with an agent that never queried (NQ) and one that always queried (AQ). On the same graph, the dotted line indicates



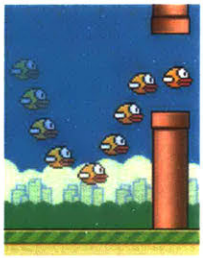
(a) Catcher Corrections



(b) Catcher Demonstrations



(c) Flappy Corrections



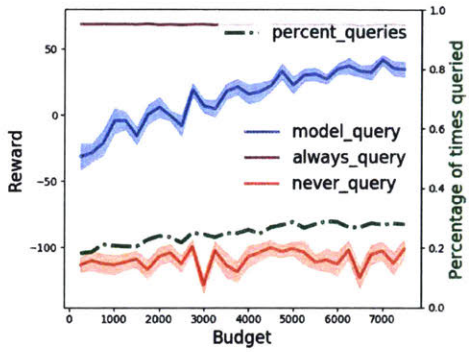
(d) Flappy Demonstrations

Figure 2-6: The data bias of demonstrations and corrections observed in both the Catcher and Flappy Bird domains.

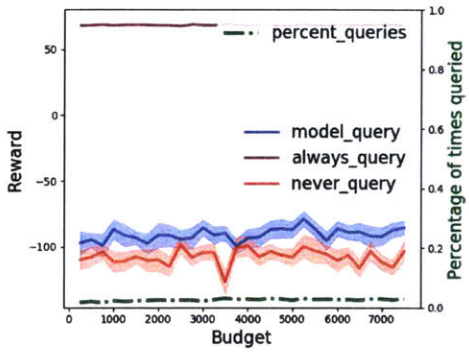
the percentage of times the agent queried the oracle for help using our model. Across all feedback types, the model achieved higher reward than the NQ agent, while still querying with relative infrequence.

Figures 2-7a and 2-7c show that, with a lenient oracle, D-AM and C both yielded greater rewards than NQ, while C had a lower percentage of queries. C, however, did obtain less reward than D-AM, because corrections did not receive labels about many blind spot regions, as the oracle steered the agent away from dangerous areas before the agent actually visited them. Thus, when the agent began the task close to a blind spot region, the model was uninformed about that part of the world, and thus the agent did not know to move away. In Section 2.8.1, we introduce a modification to our approach that allows the agent to learn about the blind spot regions it did not visit by estimating a dynamics model of the world and propagating oracle labels to potential future states.

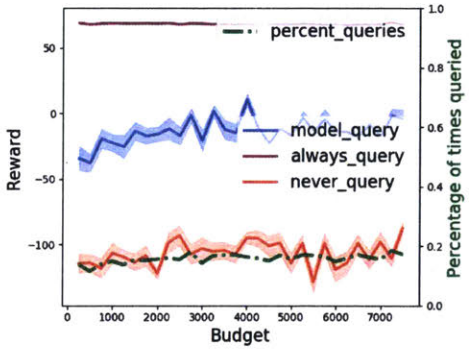
Interestingly, even though D-AM did not earn a high F1-score on classifier performance, it performed well during OIL evaluation because D-AM considered any action mismatch (where the agent deviated from the optimal action) as an unacceptable label, resulting in an overly conservative agent that queried for help at all mismatches. For example, in the Catcher domain, when the oracle was far away from a bad fruit and moving further from it, D-AM marked these regions as blind spots due to action mismatches. When the agent queried for help in these states, the oracle instructed the agent to move away. For D-A (Figure 2-7b), states far from the fruit were re-



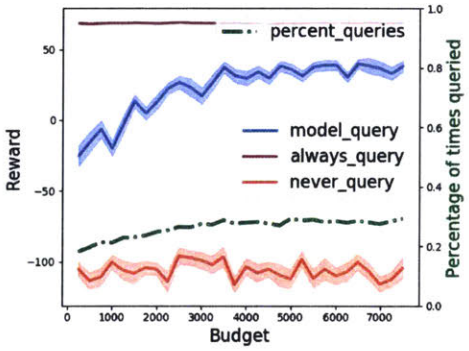
(a) D-AM with a lenient oracle



(b) D-A with a lenient oracle



(c) C with a lenient oracle



(d) D-A with a strict oracle

Figure 2-7: Oracle-in-the-loop evaluation in the Catcher domain with varying feedback types.

solved as safe; the agent thus moved towards the fruit, but because there was never any demonstration data in which the agent was close to bad fruits, D-A did not know to act. With a strict oracle (Figure 2-7d), all regions with action mismatches were considered blind spots, so D-A did not have this issue and queried the oracle in the same states as D-AM.

We observed similar trends in the Flappy Bird domain, as shown in Table 2.3. With a strict oracle, all three feedback types performed similarly and better than NQ, while also querying much less frequently than AQ. With a lenient oracle, D-A performed poorly because the initial states (where the oracle demonstrated how to fly low and between the pipes) were resolved to be safe. The agent thus did not query and ended up flying high, resulting in a substantial negative reward. Note that the

Table 2.3: Reward and percentage of times queried for oracle-in-the-loop evaluation in the Flappy Bird domain.

	Strict		**Lenient**	
Feedback	Reward	% Queries	Reward	% Queries
AQ	12.69	100%	12.66	100%
NQ	-318.77	0%	-316.17	0%
D-A	-151.15	25%	-373.46	5%
D-AM	-197.84	20%	-186.46	23%
C	-147.21	21%	-125.44	14%

classifiers used during OIL evaluation valued precision as much as recall, meaning that they put equal emphasis on making a mistake and querying the oracle unnecessarily. In cases where errors were more costly than querying, the classifier threshold could be chosen accordingly to increase the aggregate reward of oracle-in-the-loop execution in exchange for a larger percentage of queries to the oracle.

2.7 Relaxing Optimality Assumptions Analysis

Here, we present analyses of our approach when we relaxed two of our initial optimality assumptions. The first assumption we made in our formulation was that the oracle policy π_{real} was optimal in the real world, resulting in optimal demonstrations. In this section, we evaluate the performance of our approach when the oracle's policy was suboptimal in the real world. The second assumption we made was that the policy π_{sim} learned from simulation was optimal; we also present an analysis of our approach with a suboptimal π_{sim}. Finally, we evaluate our method when we relaxed both assumptions simultaneously.

2.7.1 Suboptimal oracle policy π_{real}

In previous results, we assumed that the agent received demonstrations from an oracle that always followed the optimal policy when acting in the real world. We relaxed this assumption and incorporated an oracle that selected acceptable, but not necessarily optimal, actions. To simulate a suboptimal oracle, in each state $s_i \in \mathcal{S}_{real}$, a random

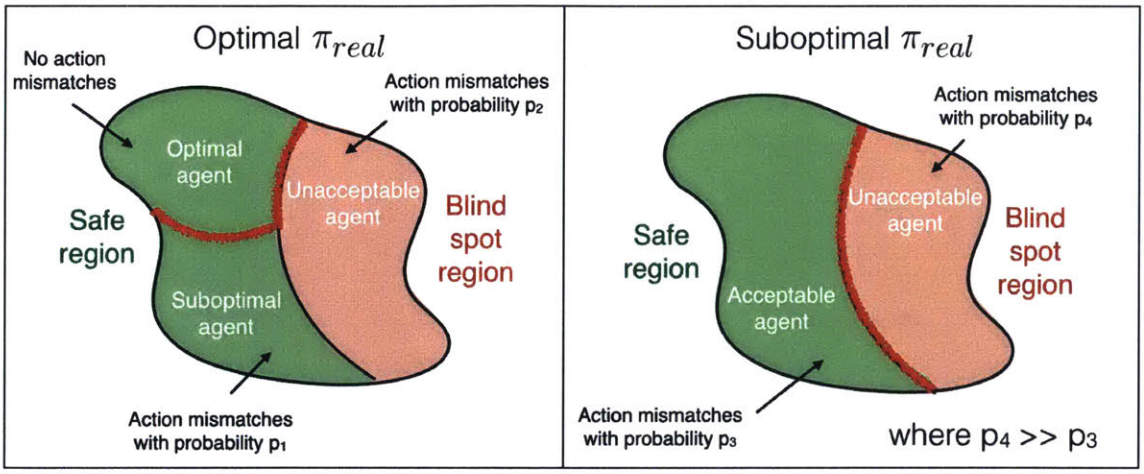


Figure 2-8: A visualization of differences in the data when an oracle followed an optimal vs. a suboptimal policy. The red lines represent the separation boundaries that the Dawid-Skene algorithm learned in order to distinguish safe and blind spot regions.

acceptable action a_i from the set of all possible acceptable actions $\{a \in \mathcal{A} | f_{acc}(s_i, a) = 0\}$ was chosen by the oracle, where $\mathcal{A}$ represents the set of all possible actions and $f_{acc}(s_i, a)$ is the acceptable function. Collecting demonstrations from a suboptimal but acceptable oracle increases the variety of acceptable and unacceptable labels in demonstration data, which can provide more information to the agent. However, this also leads to a larger number of noisy signals for blind spot discovery, because when the oracle is providing non-optimal demonstrations, mismatches become more prevalent throughout.

Difference in data

First, we discuss the differences between the labels the agent receives when an oracle follows an optimal policy vs. a suboptimal but acceptable policy. We analyze the data by observing the ratio of acceptable and unacceptable labels for safe and blind spot regions.

In an ideal scenario, for DS to properly distinguish between the two different categories, we ideally want the proportion of acceptable and unacceptable labels to appear consistently different between safe and blind spot regions. As depicted in

Figure 2-8, we observed that with an optimal oracle providing demonstrations, two clusters within the safe region received different proportions of labels. One cluster of the safe region had no action mismatches, because these were areas that appeared in the simulation task with good fruits only, in which the agent and oracle always acted optimally. Another cluster in the safe region existed, which represented areas in the real world in which the agent acted suboptimally, but still acceptably; thus, both action matches and mismatches existed. Given the presence of two different clusters in the safe region, DS, an unsupervised learning method, learned to separate the safe regions with no action mismatches from all other regions; the red line in Figure 2-8 conceptually represents what DS learned.

To better understand the two safe clusters, consider the Catcher example: only good fruits fall on the left-hand side of the game, which is identical to the simulation environment. If the oracle acts optimally here, its actions and those of the agent will always match, and the agent will never receive an unacceptable label in this safe region. On the right-hand side of the game, good fruits fall with a certain probability; otherwise, bad fruits fall. Some parts of this region are safe, specifically areas where the agent might perform a suboptimal but acceptable action that differs from the oracle's optimal action, resulting in action mismatches.

When an oracle follows a suboptimal policy, there are no longer two separate clusters in the safe region. When the real world matches the simulation, the agent's optimal action and the oracle's suboptimal action no longer always match. Figure 2-8 shows that only two regions exist with a suboptimal oracle, because all agent observations receive both acceptable and unacceptable labels from action mismatches. Interestingly, even though the number of mismatches increases overall, the labels in the safe region become more homogeneous, making it easier for DS to separate the two categories.

Aggregation and classifier performance

Figure 2-9a depicts the accuracy of the DS algorithm when obtaining data from an optimal vs. a suboptimal oracle. The graph shows that with a low budget, DS'

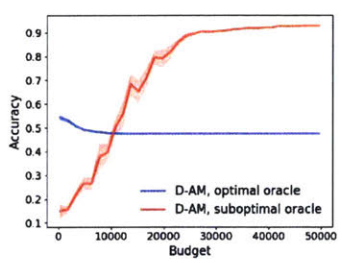
π_{real}, π_{sim}	DS Accuracy	F1 Seen	F1 Unseen	OIL Reward	OIL % Queries
Opt, Opt	0.477	0.134	0.258	44.855	0.288
Sub, Opt	0.929	0.628	0.396	-9.721	0.154
Opt, Sub	0.388	0.333	0.332	53.862	0.512
Sub, Sub	0.944	0.481	0.295	17.694	0.090

Table 2.4: A comparison of aggregation, classifier, and oracle-in-the-loop performance in the Catcher domain when π_{sim} and π_{real} were each set to both optimal and sub-optimal.

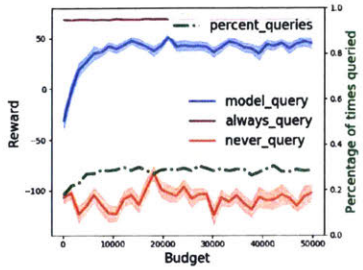
accuracy is lower with a suboptimal oracle policy π_{real}; this is because the proportions p_3 and p_4, as shown in Figure 2-8, are not easily distinguishable. As the budget increases, the data from a suboptimal policy helps DS to achieve greater accuracy because the ratio of acceptable and unacceptable labels begins to differ between the safe and blind spot regions. Specifically, the percentage of unacceptable labels from action mismatches is much higher in blind spot than in safe regions (i.e., $p_4 \gg p_3$). With more action mismatches, it would intuitively seem that differentiating between categories would become more difficult; however, with sufficient data, the action mismatches are helpful for separating the two groups more easily.

The first two rows of Table 2.4 show the performance of each part of our pipeline when the optimality of the oracle policy π_{real} was relaxed. The table depicts DS' accuracy when predicting safe and blind spot regions, the performance of the classifier on seen and unseen data, and oracle-in-the-loop performance based on the predicted blind spots. (All of these results are based on a high budget of 50,000 labels from the oracle.) DS' accuracy improved when π_{real} changed from optimal to suboptimal, which is consistent with Figure 2-9a.

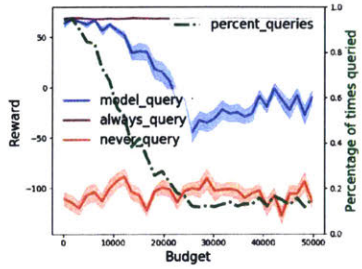
For classifier performance, having a suboptimal oracle resulted in better F1-scores than using an optimal oracle. For seen data, F1-scores were predominantly based on DS' accuracy, so it follows that the classifier predicted seen data well when DS had high accuracy. The performance on unseen data was lower overall than seen data because it is difficult to generalize well, but the suboptimal oracle condition still led to better performance than the optimal condition.



(a) The accuracy of the Dawid-Skene algorithm when estimating safe and blind spot regions given data from an optimal vs. a suboptimal oracle.



(b) Optimal π_{real}, Optimal π_{sim}. The agent could effectively avoid dangerous blind spot regions and perform well in OIL evaluation with a sufficient budget.



(c) Suboptimal π_{real}, Optimal π_{sim}. The agent performed worse with a larger budget, because the prior of blind spots was estimated to be lower.

Figure 2-9: A comparison of an optimal vs. a suboptimal oracle policy π_{real} within the Catcher domain. Figure 2-9a depicts DS' accuracy, and Figures 2-9b and 2-9c show OIL performance.

Oracle-in-the-loop evaluation

We next analyze whether high aggregation performance translates to high performance during oracle-in-the-loop (OIL) evaluation with suboptimal demonstration data. Figure 2-9b indicates that with sufficient budget, an agent trained on optimal demonstrations could get relatively close to achieving the optimal reward. Figure 2-9c shows that OIL performance actually dropped with suboptimal demonstrations as budget increased; this is because even though Dawid-Skene's performance was high, the prior of blind spots was predicted to be lower than the true ground truth prior. With a larger budget, DS was more certain about its predictions, but estimated a lower prior of blind spots. Our current approach chooses to query in the world based on a threshold selected during model learning, which aims to predict a similar percentage of blind spots in the calibration data as the training data. As the prior was estimated to be lower than the ground truth for the suboptimal oracle, the threshold was selected to be very high in order to prefer fewer predicted blind spots.

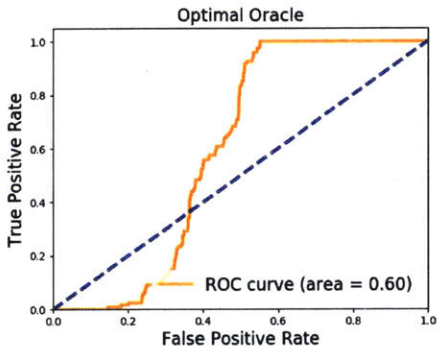
To evaluate the performance of our model invariant of the threshold, we compared the ROC curves between the optimal and suboptimal oracle. Figure 2-10a indicates that with an optimal oracle, the learned classifier did not predict much better than

a random classifier, with an AUC of 0.6. On the other hand, a classifier trained on suboptimal data yielded an AUC of 0.96 (as shown in Figure 2-10b) because it was able to separate safe and blind spot regions well across different settings of thresholds.

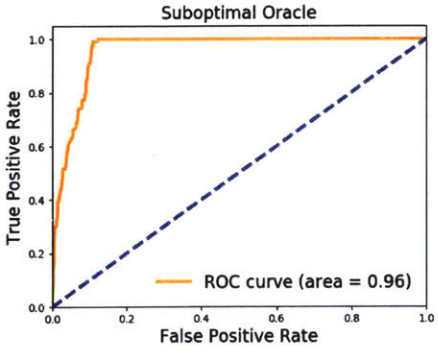
In order to understand the effect of prior estimation on OIL execution, we compared the performance of an optimal and a suboptimal π_{real} when we fixed the prior to be the true prior of blind spots in the data. This is, of course, an unrealistic condition, because the agent would never know the true prior of blind spots; however, it shows that the model can learn blind spots well if the prior can be better estimated. Figure 2-10c shows that with a fixed prior of blind spots, an agent learning from suboptimal data can achieve much greater reward than an agent learning from optimal data with the same prior.

Since we cannot set this manually to an unknown prior in the real world, we can instead combine the strengths of both optimal and suboptimal data. Optimal demonstration data helps the model to learn a more conservative prior, since many safe regions are predicted to be blind spots; suboptimal demonstrations provide more uniformly distributed labels that make it easier to disambiguate safe and blind spot groups. If there was an option to query the oracle for specific forms of data, the agent could first query for a small amount of optimal data to learn a prior, and then ask for more data from a suboptimal policy that would help to better predict blind spots in seen data. Figure 2-10d indicates that an agent could indeed perform better by combining both forms of data. In our experiment, given a particular budget on the x-axis, the agent queried for optimal data with 33% of its budget and used these labels to estimate the prior of blind spots. It then queried for suboptimal data with 67% of its budget to perform aggregation and predict blind spot probabilities for each agent observation. The learned classifier used the prior estimated from optimal data to set the threshold, resulting in much better performance during OIL evaluation than with suboptimal data alone.

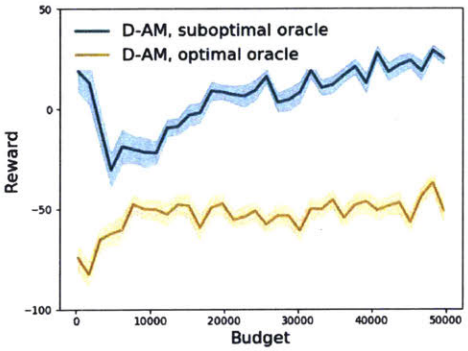
Overall, by using suboptimal instead of optimal demonstration data, the agent was able to better predict safe and blind spot regions. In terms of real-world performance, the agent that learns from optimal demonstrations earns a greater reward because



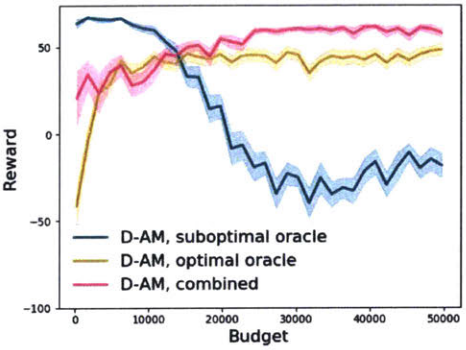
(a) ROC curve for an optimal π_{real} with an AUC of 0.6.



(b) ROC curve for a suboptimal π_{real} with an AUC of 0.96.



(c) Optimal vs. suboptimal oracle with a fixed prior.



(d) Combined optimal+suboptimal on OIL evaluation.

Figure 2-10: A comparison of optimal and suboptimal π_{real} demonstrations in the Catcher domain through an analysis of the ROC curve. A classifier trained with suboptimal data (Figure 2-10b) was better able to separate safe and blind spot regions compared with one trained with optimal demonstration data (Figure 2-10a).

it is more conservative. This is an important point, because it demonstrates that classifier performance on the test data is not the only measure to evaluate. Even with poor performance of blind spot prediction, an agent could avoid blind spot regions altogether by being slightly more conservative. In order to achieve both high aggregation and high OIL performance, one approach is to learn the prior with a small amount of optimal data, then learn the blind spot classifier with a set of suboptimal demonstrations; thus, the agent could benefit from both forms of data for better overall performance.

2.7.2 Suboptimal simulator policy π_{sim}

Another assumption we made was that the agent's policy π_{sim} learned from simulation would be optimal with respect to the simulation world. Learning an optimal simulation policy may not always be possible if the agent has limited training time; thus, we observed the effect that a suboptimal policy has on the agent's ability to identify blind spots and perform well in the real world with an oracle in-the-loop. The agent trained in the simulation environment for a limited time, and we evaluated our approach with this suboptimal π_{sim}.

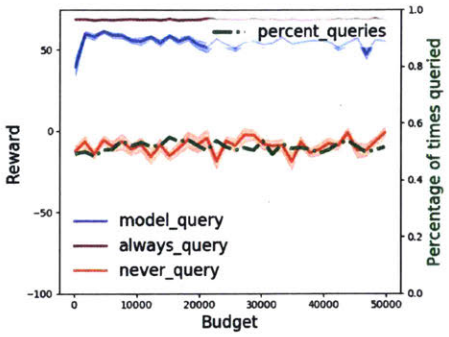
Difference in data

In terms of differences in the collected data, we observed a greater number of unacceptable labels from action mismatches for both safe and blind spot regions using a suboptimal simulator policy, as compared to an optimal one. This is to be expected, because given that the agent is taking suboptimal actions, there is a greater likelihood that the agent's action will not match that of the oracle in safe regions. Similarly, in blind spot regions, the agent's action will match the oracle's less frequently; for example, in a blind spot region where a bad fruit is close to the agent, two potential corresponding real-world states exist: one in which a bad fruit is close, and one in which a good fruit is close. If the agent's action is suboptimal in the "good fruit" region, there will be more action mismatches, even for blind spot regions.

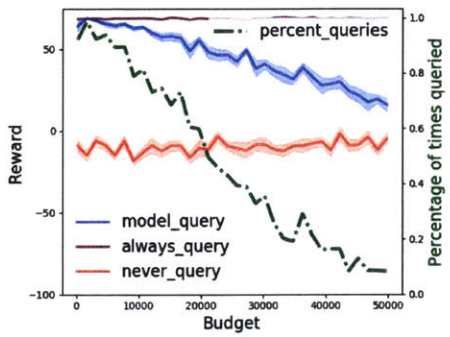
Taking a closer look at the label distribution, when π_{sim} is suboptimal, many safe regions never receive a "safe" label. The agent always takes the suboptimal action in these states, while the oracle always takes the optimal action. Thus, the agent will only receive action mismatch labels, making it impossible for DS to predict these agent observations as safe.

Aggregation and classifier performance

When the agent learns an optimal simulator policy, the Dawid-Skene algorithm learns to separate observations with only "safe" labels from everything else. With a subop-



(a) Optimal π_{real}, Suboptimal π_{sim}. The estimated prior of blind spots from DS was high, so the agent queried frequently and performed well.



(b) Suboptimal π_{real}, Suboptimal π_{sim}. The agent's OIL performance decreased as budget increased.

Figure 2-11: OIL performance with a suboptimal π_{sim} in the Catcher domain. Performance is compared between an optimal and suboptimal π_{real}.

timal simulator policy, there is an increase in the number of agent observations with only action mismatches. Thus, DS learns to separate regions with action mismatches alone from all other observations, resulting in similar accuracy for both optimal and suboptimal policies on aggregation. However, the suboptimal π_{sim} obtains a better F1-score than the optimal π_{sim}, because the prior from DS was estimated to be high for blind spots. The threshold chosen during model learning allows for prediction of a greater number of blind spots.

Oracle-in-the-loop evaluation

Because DS predicts a higher prior of blind spots, a suboptimal π_{sim} actually performs better than an optimal π_{sim} during OIL evaluation. The agent predicts many safe regions to be blind spots and queries the oracle for help; this allows the agent to avoid dangerous regions, but also results in more unnecessary (and undesirable) querying.

Also, the worst possible reward that an agent can receive by never querying is higher for a suboptimal simulator policy compared with an optimal simulator policy. This is not intuitive, but is an artifact of our domain and reward function. In the Catcher domain, if the agent consistently moves toward a bad fruit, the agent receives

a negative reward. When the agent takes a suboptimal action, it actually receives a better reward: for example, if a bad fruit is present, the agent interprets it as a good fruit and would move closer if π_{sim} were optimal, resulting in some reward, x. However, if the agent were to act suboptimally in that state, it would either remain stationary or move left, both of which would result in $> x$ reward.

2.7.3 Suboptimal oracle policy π_{real} and suboptimal simulator policy π_{sim}

When both assumptions were relaxed, performance improved over that observed when either assumption was relaxed separately. Figure 2-12 summarizes the results of each possible variation of π_{real} and π_{sim} being optimal vs. suboptimal. The horizontal axis varies π_{real}, and the vertical axis varies π_{sim}. The figure highlights changes to aggregation, classifier, and OIL performance when moving from one assumption box to another.

By receiving more varied demonstrations from a suboptimal π_{real}, the agent can better separate safe and blind spot regions because the proportions of acceptable and unacceptable labels are more separable between the two classes, resulting in improved aggregation performance. With a suboptimal π_{sim}, the prior of blind spots predicted from DS increases, resulting in a threshold that captures a greater number of blind spots – and, in turn, better classifier and oracle-in-the-loop performance.

2.8 Data augmentation improvements

We now present two improvements to our pipeline that boost performance when working with correction and demonstration data. The first improves the performance of corrections: estimating the transition model of the world and propagating unacceptable signals to later states even when the agent never visits those states (as they are dangerous and the oracle will not allow the agent to go there). The second improves demonstrations: augmenting demonstration data with corrections to have

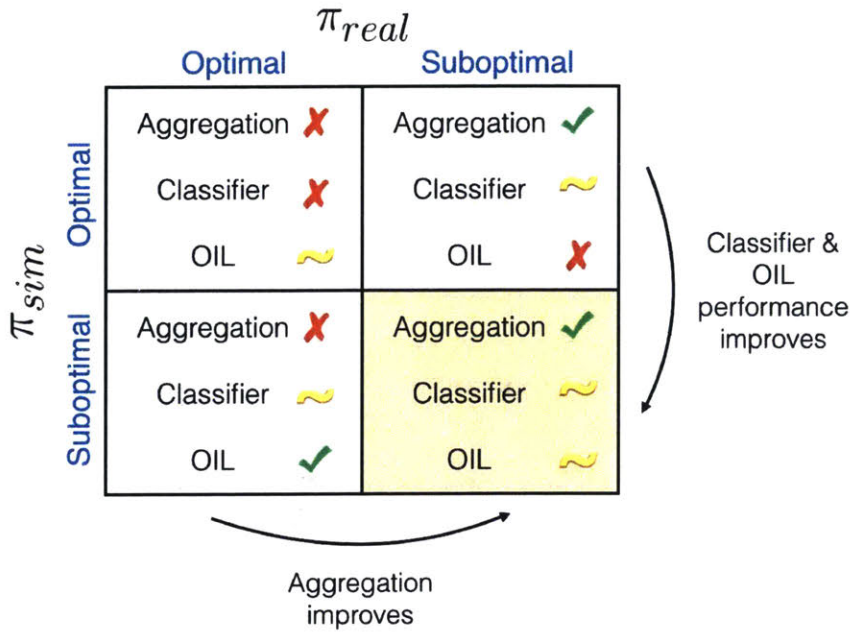


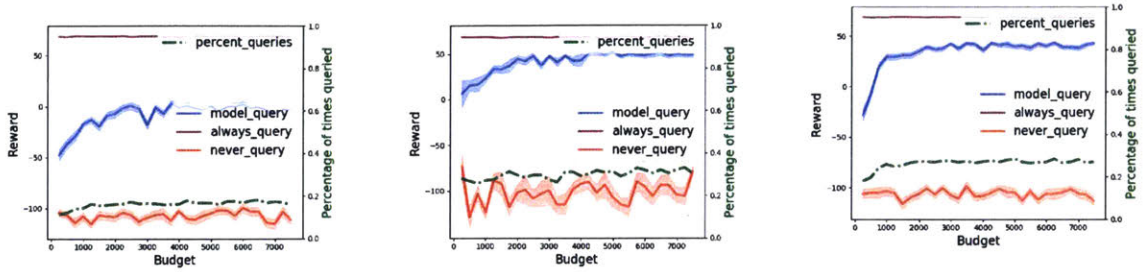
Figure 2-12: This figure summarizes the results from relaxing π_{real} and π_{sim}. When π_{real} was suboptimal, it became easier for Dawid-Skene to aggregate labels. When π_{sim} was suboptimal, the prior of blind spots increased, improving classifier and oracle-in-the-loop performance.

better exposure in the real world, and thus improved prediction of blind spots.

2.8.1 Boosting corrections performance

The first improvement is intended to increase performance when using corrections data. When the agent obtains labels from corrections, it receives direct signals about its own actions. Consequently, the unacceptable labels the agent receives are true labels, not based on action mismatches. The issue, however, is that the agent never receives labels in dangerous blind spot regions, since the oracle always steers the agent away before the agent reaches that area. Figure 2-13a depicts the performance of an agent using corrections to predict blind spots within the Catcher domain: it was unable to achieve high OIL performance because when it began the task close to blind spot regions, it did not have a good model of blind spots and therefore had not learned to avoid them.

To analyze the impact of an agent not receiving signals when close to blind spot regions, we ran an experiment with a modification in the data collection for correc-



(a) Original corrections data. The agent was unable to achieve the same performance because it lacked signals about blind spot regions that it never visited.

(b) π_{sim} in the real world, without action corrections. (Note this is an unsafe and infeasible data collection method, and is only used as a comparison to corrections.)

(c) Corrections along with additional labels propagated by estimating the transition model of the real world. The agent achieved performance similar to π_{sim} without corrections, but *safely*.

Figure 2-13: The system's performance in the Catcher domain when propagating unacceptable labels to future states based on an estimated transition model of the world. This resulted in improved performance with corrections data, while also maintaining safety.

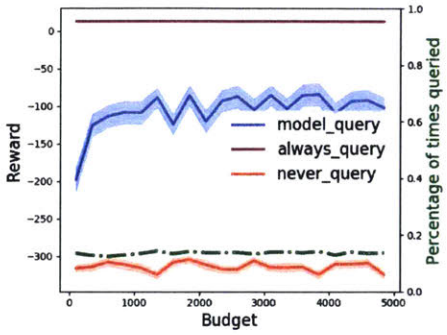
tions. With the original corrections feedback type, the agent receives labels from the oracle, and in the event of unacceptable labels, the oracle steers the agent away from blind spots by providing the optimal action to take at that state. With our modification, the agent simply uses π_{sim} directly in the real world during data collection, and obtains acceptable and unacceptable labels without being corrected. This allows the agent to receive true labels for many more blind spots, resulting in more accurate blind spot modeling. However, running π_{sim} without corrections cannot be safely transferred to the real world; we include the analysis here only in order to assess the benefit of the agent receiving signals about blind spot regions. Figure 2-13b indicates that this modified form of corrections feedback performed much better than the original during OIL evaluation in Catcher.

Since running π_{sim} without action corrections can be dangerous and may involve costly errors, the agent can instead propagate the unacceptable labels it receives to future states based on which regions the agent thinks it will go to. In other words, the agent can safely obtain data using the original corrections feedback type (in which the

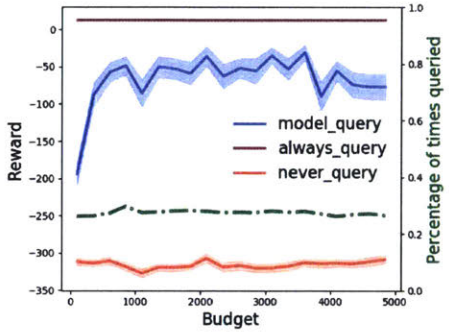
oracle steers the agent away from dangerous regions). However, to better model blind spot regions without labels from the oracle, the system can estimate the dynamics model of the world and propagate unacceptable labels to potential future states. To do this, the system first collects data from corrections in the original form (with action corrections), providing the agent with acceptable and unacceptable signals. Our system then estimates the transition model of the real world, $\hat{T}_{real}(o_{agent}, a, o'_{agent})$, using this corrections data. This transition model estimate is based on the agent observation space, $o_{agent} \in \mathcal{S}_{agent}$, because this is the representation the agent has when recording data from the oracle. We chose to estimate the transition model of the real world, rather than the one in simulation, because this is more representative of the world that the agent will operate in. If the agent has access to very little real-world data, however, another possible approach would be to estimate an approximate transition model of the simulation world and use it as a proxy for estimating real-world dynamics.

Given $\hat{T}_{real}(o_{agent}, a, o'_{agent})$ and the labels obtained through corrections, our system propagates every unacceptable label a few steps forward. This can be modified to propagate a decaying unacceptable label over a longer sequence. The aim of propagation is to provide better labels for blind spot regions that the agent never visited. However, by artificially creating these new labels, our system may unintentionally apply unacceptable labels to safe regions, which could result in overprediction of blind spots. However, as long as overprediction is minimal, this method remains preferable to the original corrections method because it results in a slightly more conservative and safer agent. Figure 2-13c indicates that when the system estimates the transition model and propagates unacceptable signals, the system obtains similar performance to π_{sim} on the Catcher domain, while also maintaining safety.

This improvement for corrections data leads to higher performance in the Flappy Bird domain as well. Figure 2-14a depicts OIL performance with the original corrections data: in this case, the agent was able to obtain much higher reward compared with an agent that never queried, and it only queried approximately 15% of the time. However, the agent was unable to obtain a maximum reward close to that obtained



(a) This approach uses original corrections data to learn a blind spot model.



(b) This approach uses corrections with propagated labels based on potential future states the agent might visit.

Figure 2-14: Performance in the Flappy Bird domain with the corrections augmentation.

by an agent that always queried. In Figure 2-14b, we include the propagation of correction labels to potential future states; this addition improved performance during OIL evaluation to be closer to the best possible reward.

2.8.2 Boosting demonstrations performance

The second improvement involves boosting performance when using demonstrations, since demonstration data suffers from limited state exposure. In the Catcher example, demonstration data only includes states located near good fruits and far away from bad fruits. The agent never sees regions close to bad fruits; however, these are still important areas that the agent would visit in the real world. Earlier results indicated that an agent using demonstration data obtained low F1-scores on classifier performance, but was able to perform relatively well on OIL performance with sufficient budget by being extra-conservative.

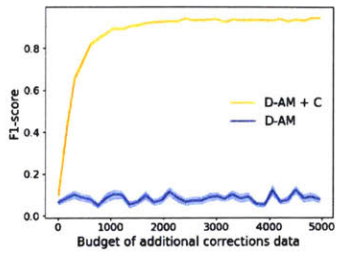
To improve F1-scores and OIL performance, we analyzed a feedback type where the agent was able to review all action mismatches in demonstration data with the oracle (D-A). Our findings showed that reviewing these action mismatches did not improve performance – and, in fact, resulted in a performance drop. Regions with

mismatches were resolved to be safe, leading the agent to unsafe regions for which it had no data and thus did not know to query the oracle. Since reviewing action mismatches in demonstration data did not help, another option would be to augment the space of visited states in demonstrations with a new set of states obtained from corrections data. Imagine that the agent has access to a set of demonstrations: to improve performance, the system could ask the oracle for a limited monitoring session in the real world in order to collect more data.

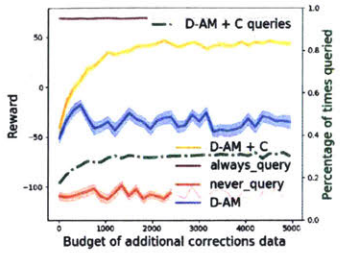
In our experiments, we set a constant budget of 250 labels obtained from demonstration data to which the agent already had access. We then observed the performance of our approach as demonstration data was increasingly combined with more data obtained through corrections. Figure 2-15a indicates that in the Catcher domain, the F1-score for seen data improved considerably with the addition of corrections data. Corrections data did not contain any action mismatches, and was therefore easier for the Dawid-Skene algorithm to separate. The performance improvement resulting from DS in turn improved classifier performance on both seen and unseen data. Figure 2-15b depicts OIL performance in Catcher of an agent that learned only from demonstrations (D-AM) vs. an agent that learned from both demonstrations and corrections data (D-AM + C). We used the modification proposed in Section 2.8.1 to propagate unacceptable labels for the corrections data, and found that D-AM + C performed considerably better than D-AM alone. In the Flappy Bird domain, we observed similar benefits from augmenting demonstrations data with corrections data. While keeping the amount of demonstration data constant at 1,000 labels, we increased the amount of combined corrections data; Figure 2-15c indicates the benefit to reward obtained in the real-world environment with this augmented data in Flappy Bird.

2.9 Discussion

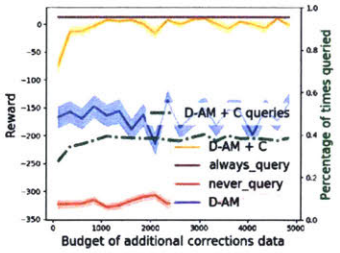
In this work, we formulate the problem of identifying blind spots of AI systems that occur due to incomplete state representation. Many current safe AI methods



(a) Classifier performance on seen data when incorporating data from demonstrations alone vs. demonstrations augmented with corrections data in the Catcher domain.



(b) OIL performance when using data from demonstrations alone vs. demonstrations with additional corrections data in Catcher.



(c) OIL performance in Flappy Bird comparing demonstrations alone with demonstrations and additional corrections data.

Figure 2-15: Assessing the benefit of combining demonstrations data with additional corrections data to increase exposure and improve performance.

assume sufficient representation for learning and acting [73, 195, 110], but when that representation is insufficient, these approaches can break down. Our system explicitly learns how to find these errors through the help of an external oracle or human; using this feedback, agents can better understand their own failures and query for help accordingly.

We envision a future with safer deployment of more self-aware AI systems, allowing for iterative refinement with safely obtained real-world data. Our approach to learning blind spots provides a first step toward using human feedback to identify agent failures. Given an agent's flawed representation, we introduce a step to intelligently aggregate signals provided by an oracle operating within a different representation, and demonstrate in our experiments that our blind spot model enables an agent to avoid costly mistakes while querying with relative infrequence.

In our current evaluation of real-world execution, the agent decides when to query in the real world according to a threshold selected during the model learning process. If the agent knew the relative cost of making a mistake in the world vs. the cost of querying an oracle, it could make a more informed decision about whether to query or proceed with an action. Decision-theoretic reasoning can thus be included in our

pipeline to enable more effective agent-oracle teaming in the real world. We can additionally include a notion of risk aversion for the agent, which can be modified based on the preference of the oracle guiding that agent, or on how high the stakes are in the real world (which would depend upon the domain). Adding decision-theoretic analyses can be a principled way of deciding how to act in the real world using the learned blind spot model, which can improve coordination between the agent and the oracle.

One limitation of our current approach is that we still require sufficient oracle data to learn blind spots. Active learning can be used to reduce the number of labeled data points by intelligently querying for labels at important states. One challenge related to active learning in this setting is that the agent must query for trajectories rather than single data points (as in supervised learning). This requires the agent to reason about possible futures and select a trajectory query that will lead to the greatest information gain across the full set of points. Further, because the representation of the agent and the human may not match, limited communication can make active learning more challenging. One way to share feedback is through actions or trajectories alone, so the state is not explicitly communicated – similar to the approach we use in this work. Another technique could be querying through an interpretable communication channel that leverages an easy-to-understand structure. One such example would be to use linear temporal logic [199] as a shared medium or to use other forms of data querying, such as preference elicitation [50, 26].

In future work, we would like to apply our method to more complex problems and richer real-world applications. Specifically, some avenues for future research include analyses involving real human users, scaling up the approach to high-dimensional state representations, and applying the method to specific applications (such as autonomous driving and robotics).

Collecting data from human users can introduce new challenges, such as access to even smaller sample sizes and the presence of additional forms of noise. As discussed earlier, active querying to users represents one way of learning good models when the cost of obtaining data is high. Human feedback can also include more noise

in both demonstrations and corrections. The analysis of suboptimal demonstrations presented in this work represents a first step toward understanding this and is encouraging, as it indicates that our system can learn equivalent or better blind spot models with simulated suboptimal demonstration data. Another form of noise in human data that would be interesting to analyze in future work is the lag in correction signals. Realistically, people cannot provide action corrections instantaneously due to slow human reflexes, which can cause an agent to misinterpret signals. Thus, it is important for the agent to learn how to perform credit assignment and penalize the most likely action the human was referring to. Human studies could help us better understand how to augment our blind spot model learning.

In addition to working with people, we would like to explore how our approach could be modified to work in high-dimensional and continuous state representations. Currently, our aggregation step can easily work in discrete state spaces, but with continuous representations, it would be difficult to reason about many real-world states mapping to a single agent observation. One possible method for addressing this would be to cluster similar states according to some metric before applying our method. (If the initial clusters are inaccurate, it may be necessary to update these clusters.) Also, in high-dimensional state spaces, features are often automatically learned through methods such as deep learning, and developing methods that are able to reason about and visualize an agent's flawed representation could be useful for refining the training environment or agent model. Finally, using our approach in real-world applications would be an interesting and impactful direction for future work. Understanding the types of blind spots that exist in today's AI systems (such as self-driving cars and commercial robots) could inform the addition of other improvements to our method.

2.10 Conclusion

In this work, we addressed the challenge of discovering agent blind spots in reinforcement learning when the state representation of an agent does not sufficiently

describe the real-world environment. We proposed a methodology to explicitly handle noise induced by this representation mismatch, as well as noise from low-precision oracle feedback. Our approach achieved better performance than baseline methods when predicting blind spots in the real-world environment. We additionally showed that this learned model helped to avoid costly mistakes during real-world execution, while also drastically reducing the number of oracle queries. We discussed the biases caused by different types of feedback (namely, demonstrations and corrections), and assessed the benefits of each based on domain characteristics. Finally, we included an analysis of how our model performed when optimality assumptions were relaxed and added data augmentation improvements to enable more accurate blind spot predictions when using corrections and demonstrations data. Further investigations are necessary for ideal integration of blind spot models into oracle-in-the-loop execution by trading off the cost of a mistake with the cost of querying an oracle. We also noted the possibility of moving beyond a heavy reliance upon high-quality training data via active learning approaches that can obtain more informative feedback from the oracle.

Chapter 3

Human-in-the-loop Experiments and Demonstrations

3.1 Introduction

In this chapter, the agent blind spot modeling approach from Chapter 2 is applied to human users and to a real-world mini autonomous car application. We first discuss human experiments and compare how well our model learned agent blind spots with human user data as compared with simulated human data. We further analyze whether different feedback types are easier for people to provide and whether certain environment characteristics affect the quality of human feedback. The experiments and analyses provide insight into how to apply our model with real user data.

Second, we present an application of the blind spot model to a simple autonomous car setup with a simulation and a real-world environment. We discuss the construction of the agent's state representation and how the simulation policy is learned. We collect human demonstrations of the real-world task, run the blind spot model, and demonstrate the full system working with the robot querying the human for help at states that are highly predicted to be blind spots. This demonstrates the applicability of the model to more realistic applications.

3.2 Human Experiments

We conduct human experiments across two domains with two forms of human feedback: demonstrations and corrections. The purpose of the study is to apply the blind spot model to real user data. We perform quantitative and qualitative analyses of how our model predicts blind spots using human feedback.

3.2.1 Experiment Setup

Data is collected in two environments, Catcher and Flappy Bird, which are the same domains used in previous simulation results (Chapter 2). People provide two different types of feedback: corrections and demonstrations, which were found in Section 2.6.3 to have different types of biases. We analyze the effect of both feedback type and environment on the accuracy of agent blind spot predictions.

Through these experiments, we aim to answer the following questions:

1. Can our model effectively learn blind spots using a small budget of human data? How does this performance compare to the performance with simulated human data? Since user data was much harder to obtain, there were much fewer samples than there were in simulation.

2. Is one feedback method easier for people to provide than the other (between demonstrations and corrections)? If so, why?

3. Is one environment easier for people to provide feedback in than the other (between Catcher and Flappy Bird)? If so, what are the characteristics of environments that make it easy or hard?

3.2.2 Hypotheses

From the previous three questions, we obtain the following hypotheses:

H1 *An agent trained using human data will perform worse than an agent trained using simulated data in these simple environments.*

In Chapter 2, we included results with a simulated human that was both optimal and suboptimal. We found that counter-intuitively, the model performed better with suboptimal demonstrations as compared with optimal demonstrations, if there is enough data. People are more likely to act suboptimally, but due to the difficulty in obtaining user data, we only have a small amount of human data, which we hypothesize will not increase performance. Further, people are also likely to make random errors that are not acceptable, which is not included in our suboptimal demonstration data analysis in Section 2.7.1. In addition, slow human reaction time and difficulties experienced by the participant when using the controls are factors that are not considered when training the model on simulated data. These factors have large effects on the quality of the data people provide. Due to limited suboptimal data and the potential presence of random errors, we predict that an agent will perform worse with human data than with simulated data.

H2 *An agent trained using corrections will perform better than an agent trained using demonstrations.*

We think that humans will provide better data through corrections as compared to demonstrations. In demonstrations, people have to react quickly and constantly be attentive to provide useful feedback. Small errors in demonstration data can mislead the agent into believing suboptimal or unacceptable actions as optimal. Corrections, however, only requires *identification* of a blind spot. This requires less frequent feedback and only minimal interactions when a potential blind spot is observed. Thus, we expect corrections to be a more convenient type of feedback and believe it will result in higher-quality human data.

H3 *An agent trained using human data for simple domains like Catcher will perform better than for more complex domains like Flappy Bird.*

We hypothesize that the blind spot model trained for Catcher will outperform the model trained for Flappy Bird for three main reasons. First, Flappy Bird is a more complicated environment because the reward function has two competing

objectives. The agent must fly through the pipes while also determining whether to fly high or fly low before reaching the pipes. Satisfying both objectives simultaneously can be tough and can result in more noisy human data. Secondly, the dynamics or transition function of Flappy Bird is more difficult to predict. It requires people to decide whether to apply a force upward or allow the bird to be pulled down by gravity at each time step, which is affected by previous actions. On the other hand, in Catcher, each action moves the agent exactly one unit in the desired direction, a easily estimated transition. Third, slow reaction times have a bigger impact on the Flappy Bird domain than in Catcher because the person must react fast in order to stay alive. More time to react will generally provide less noisy data. To reduce the impact of reaction times in Flappy Bird, we tested the game at a few different speeds. However, we were not able to completely eliminate this effect.

3.2.3 Methodology

Twelve participants were asked to provide two forms of feedback for two environments. Each participant gave feedback for each of the four conditions, thirty times each, which resulted in 120 trajectories in total. To eliminate any order effect, we changed the order in which participants provided each feedback type to reduce bias in the data due to familiarization with the task and interface as the experiment proceeded. We used a Latin square for counterbalancing and randomly assigned conditions prior to the start of the experiment. To begin, each participant completed a short demographic survey. The experimenter then described to the participant the general strategy for providing feedback in the different domains. To better understand the information provided to participants, sample instructions on general strategies and controls for Catcher demonstrations and Flappy Bird corrections are included below:

Catcher Demonstrations: *For this condition, you're going to be playing a modified version of Catcher. There are two different colored fruits, light and dark. The strategy for scoring high differs between the fruits. For light fruits, you want to try to catch them with the player. For dark fruits, you want to get as far away as possible from them. The controls are left arrow key and right arrow key to move, and nothing*

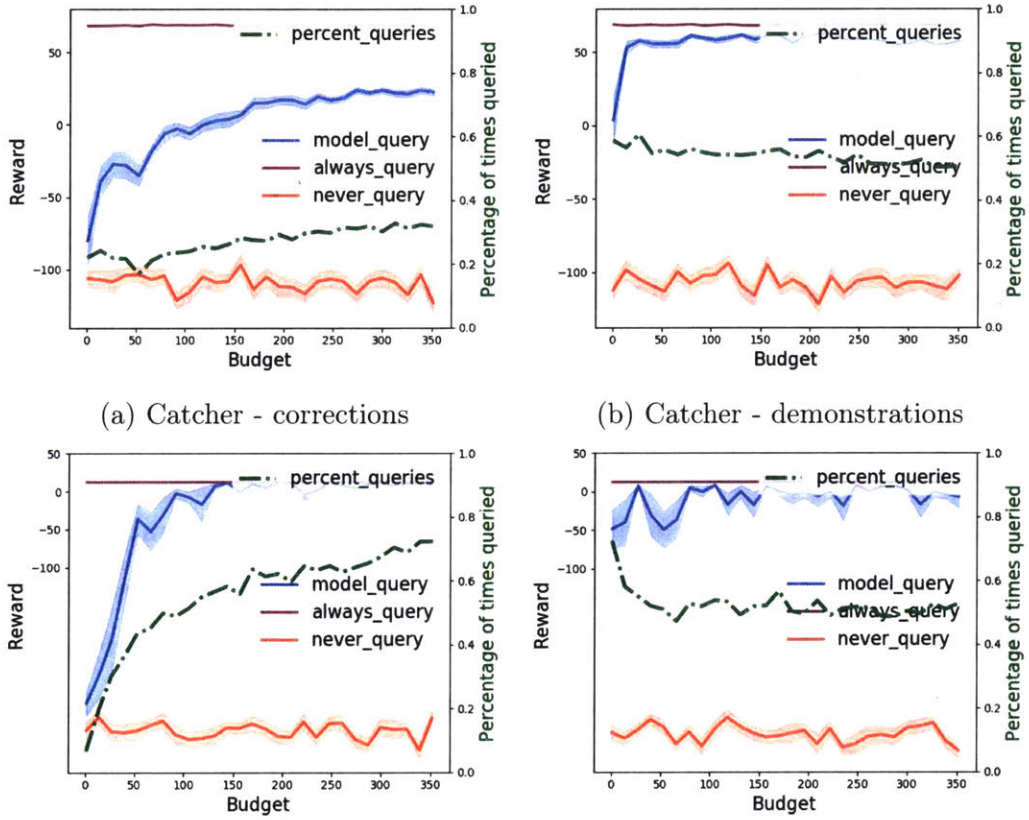
to wait in place. You can keep the keys pressed down to continue moving left or right. For light fruits, it is better to get into position under the fruit as soon as possible; for the dark fruits, it is better to get as far away as possible.

Flappy Bird Corrections: *Now you're going to watch the computer, which has already been trained to play the game. For red pipes, the lower you go before passing through the pipes, the better; for green, the higher you go, the better. If you see the computer doing something wrong, pause it with the space bar, and provide the correct action. To resume once paused, either press the up key if the bird should go up, or press the space bar again for the bird to fall. You will finish playing the round for the computer. If the computer is doing it correctly, try not to pause. If the computer is doing something wrong, the earlier you pause and provide feedback, the better.*

When acting in the real world, the agent could not see the color of the fruit/pipes and treated all fruits as light and all pipes as green. However, the participant was required to observe the agent's actions and identify unacceptable actions when possible. The person was not informed of any pattern in the blind spots in advance. Once the participant provided the first feedback type, a short survey was completed, and the process was repeated for all four conditions. The participant took a final short survey about the full experiment. All four conditions took approximately thirty minutes to complete. We recorded the state, action, next state, and reward at each time step and passed this data through the same blind spot model pipeline outlined in Section 2.4.

3.2.4 Results

We now discuss various analyses of how well the agent predicted blind spots based on the user data. Figure 3-1 shows the agent's performance on the "real-world", or more complex, environment when using the learned blind spot model to query across all 4 conditions: Catcher with demonstration data, Catcher with corrections data, Flappy Bird with demonstrations, and Flappy Bird with corrections. Catcher with corrections feedback performs quite well, as shown in Figure 3-1a. This feedback type has a comparable F-1 score when using human data vs. simulated data, which was



(a) Catcher - corrections

(b) Catcher - demonstrations

(c) Flappy Bird - corrections

(d) Flappy Bird - demonstrations

Figure 3-1: Oracle-in-the-loop evaluation on the real-world task using the blind spot model learned from user data. We include results from all four conditions of the experiment.

described in Section 2.6.3. The agent is able to receive almost 80% of the total possible reward and only query about 30% of the time, which is indicative of intelligent blind spot model learning. The agent that used simulated human data queried about 20% of the time and received about 50-60% of the reward. This means that users provided more corrective feedback than the simulated oracle, which resulted in an agent being more cautious.

For Catcher with demonstrations data, shown in Figure 3-1b, an agent using simulated oracle data received almost 90% of the total reward with just 30% queries. With human data, an agent obtained almost perfect reward but learned to query 50% of the time, which is very frequent. This indicates that the number of action mismatches with human data was so high that the agent became too conservative when acting in the real world.

For the Flappy Bird domain (Figures 3-1c and 3-1d), both corrections and demonstrations data resulted in high querying. Flappy Bird demonstrations had similar high performance compared to Catcher demonstrations, both with high rates of querying. This domain was more challenging for users, so in demonstrations, there were lots of suboptimal actions that an agent used to predict high likelihood of blind spots. In corrections, people tended to correct the agent frequently in Flappy Bird, giving strong signals of unacceptable actions. Overall, the high querying frequency makes it hard to comment on the difference in performance as the agent could be doing well due to this consistent querying for human help. A more useful model queries less.

During the feedback for Flappy Bird corrections, many participants struggled to identify the agent's unacceptable actions. In the post-experiment questionnaire, some participants stated that the agent performed very well at Flappy Bird despite wrong behavior when providing feedback. This was true more for participants that provided feedback in the form of Flappy demonstrations before Flappy corrections. The trained agent controlled the bird smoothly, whereas the human participants struggled with the controls. One explanation for the perceived capability of the agent is that the participants linked the smooth control to correct agent behavior as the agent seemed to be doing well at controlling the bird in comparison to the participants.

However, smooth control was not the main metric of good performance. In general, when providing feedback for Flappy Bird, participants were more focused on passing through the pipe than flying low or high before the pipe, which was the key factor in predicting agent blind spots.

Overall, we found that users provided suboptimal data frequently and made random errors. Even through we found that our algorithm performed better with suboptimal data in Section 2.7.1, we needed lots of data in that setting to accurately distinguish between safe and blind spot regions. With limited suboptimal user data, as is the case in these results, our model predicted too many states as blind spots. The reward obtained with user data on the real-world environment is higher than with simulated data, but the agent also queried much more. This result supports our hypothesis (H1) that an agent trained with user data will perform worse than an agent using simulated oracle data, in that the agent was overly conservative.

For hypothesis H2, we predicted that corrections will be a more effective form of feedback. In our experiments, corrections was preferable for the Catcher domain, but for Flappy Bird, demonstrations was slightly preferable because the agent queried less (50% of the time rather than 70%) with about the same amount of high reward. This partially supports our hypothesis because these results show that the effectiveness of corrections and demonstrations varies based on the domain. In some domains, corrections provides more informative feedback, while in other domains, demonstrations can be more useful.

Our final hypothesis H3 was supported because the agent's blind spot predictions and consequently real-world performance was better for Catcher than for Flappy Bird. This is because the task was clear for Catcher, and the users could easily perform the task, even with slower reaction times. For Flappy Bird, users needed to balance two simultaneous objectives: fly through the pipe and follow a path to achieve that (fly low or fly high). Users were also required to react quickly and predict the dynamics of the world accurately when selecting actions (since they needed to account for gravity when applying the upward force to the bird). This made Flappy Bird a much tougher domain to provide feedback for.

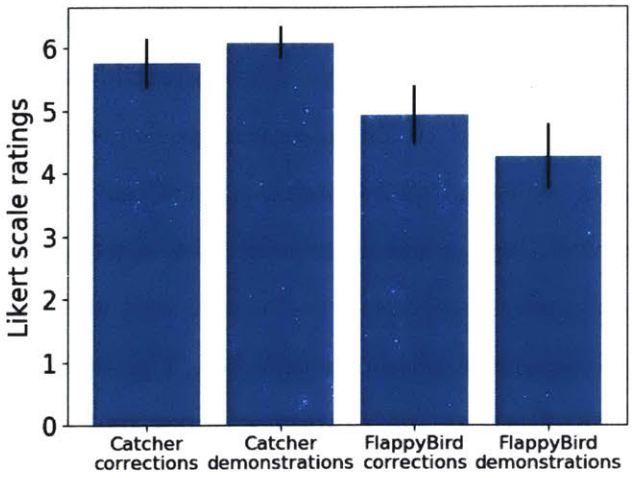


Figure 3-2: Qualitative analyses based on experiment questionnaires given to participants. Users responded to the question: How easy was it to provide feedback in this form?

We also collected subjective responses on experiment questionnaires to determine user perception of the ease of providing different types of feedback. Figure 3-2 presents Likert scale ratings for the question: "How easy was it to provide feedback in this form?". The graph indicates that people slightly preferred providing demonstrations for the Catcher domain and preferred corrections for Flappy Bird. This is because for simpler games where the task was easy to execute, people found it easier to demonstrate the task themselves. For more complex domains where proper execution was difficult as in Flappy Bird, people preferred to observe the agent act and provide corrective feedback. Qualitative results indicate that the feedback type most suitable for users depends on domain characteristics.

3.2.5 Discussion and Future Directions

We collected user data on two domains with two feedback types and applied the blind spot modelling approach from Section 2.4. We found that for both environments, the results using the simulated oracle were better than the results when users provided data. This is mainly due to lots of action mismatches in demonstration data and the high frequency of unacceptable signals in corrections. These feedback signals result in a much more conservative agent than one that was trained using a simulated oracle.

In future work, the high percentage of querying can be reduced by having a smaller threshold for determining when to query. The threshold can also be tuned as the agent acts in the world to adapt to the human helper's preferences.

In our experiments, we found that Catcher corrections was the only feedback format that performed well (high performance with low querying). Providing feedback in the form of demonstrations did not perform well, and the resulting model queried often due to the large number of action mismatches. This can be potentially improved by experimenting with alternate aggregation schemes that can better handle action mismatches in safe regions. However, if action mismatches are too frequent, any algorithm will fail to identify the true labels of different regions. Flappy Bird was a more difficult environment for the human participants due to difficult control and multiple objectives in the task. Furthermore, many participants failed to identify agent blind spots when providing feedback in the form of Flappy Bird corrections. Improving the accuracy and quality of human data is imperative for precisely predicting agent blind spots.

3.3 Autonomous Driving Application

The blind spot model approach from Section 2.4 was also applied to an autonomous driving application, in which a mini autonomous car was tasked with learning error regions that resulted from partial observability of the world.

3.3.1 Overview

First, the setup of the problem is described with a high-level description of the pipeline and final demonstration. In each of the following subsections, the various components of the pipeline are analyzed in more detail.

A robot car is trained in a simulation environment to perform a simple task: reach a goal cone at the end of a rectangular enclosure from any starting position. The leftmost image in Figure 3-3 depicts the agent's training environment. The agent's representation of the world is its distance away from the goal cone in the x dimension,

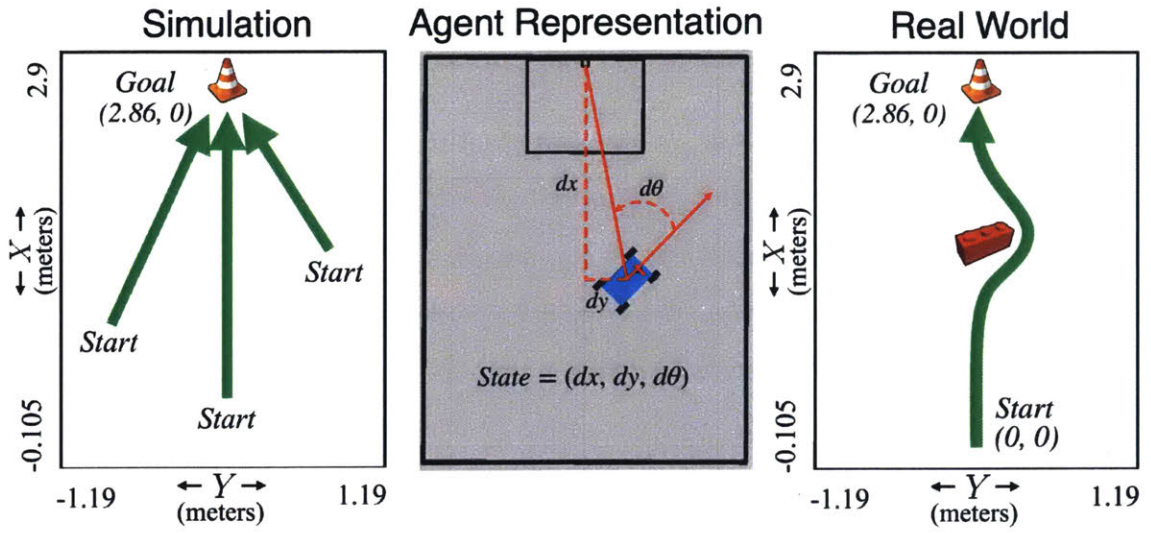


Figure 3-3: On the left, the environment being used is laid out. The green arrow represents the desired trajectory around the dangerous red block towards the target orange cone. Boundary coordinates as well as starting and ending locations are labelled. On the right, the robots state feature set is illustrated.

y dimension, and the angular direction, shown in the middle image of Figure 3-3. In this environment, the agent is able to drive forward, to the right, and to the left. After training in this simulation world, the agent learns to reach the goal from any location on the grid.

When deployed into the real world, this agent is equipped with a simple camera to sense nearby objects. Using this camera, the agent can only detect the presence of an object and cannot determine the shapes or sizes of objects. While this simple object detection capability is sufficient for acting in the simulation world, the real world is more complex with many different objects, some of which are obstacles that the agent needs to avoid. Without more complex sensors to differentiate cones from obstacle blocks, the agent cannot learn to act differently towards these different types of objects. The rightmost image in Figure 3-3 shows the setup for the real-world task, in which a new red obstacle is present, which the agent observes as identical to the orange cone. While the optimal policy in the real world is to avoid the obstacle and reach the goal cone, shown in the figure, the trained policy unfortunately directs the agent to collide with the red obstacle.

To learn where the agent is most likely to make errors, feedback from a human is

collected, and a blind spot model of the real world is learned. A human who can see the true world provides demonstrations instructing the agent to avoid the obstacle and reach the cone. This collected data, along with the agent's simulation policy, is inputted into the blind spot modelling approach from Section 2.4. The learned model is then used to query a human for help at regions predicted to be blind spots. We show a demonstration of an agent using the blind spot model to query a person intelligently and safely reach the goal in the real-world environment. A video explaining our whole pipeline for applying the model to this autonomous car application can be viewed at http://bit.ly/3100yd2.

3.3.2 Simulation training

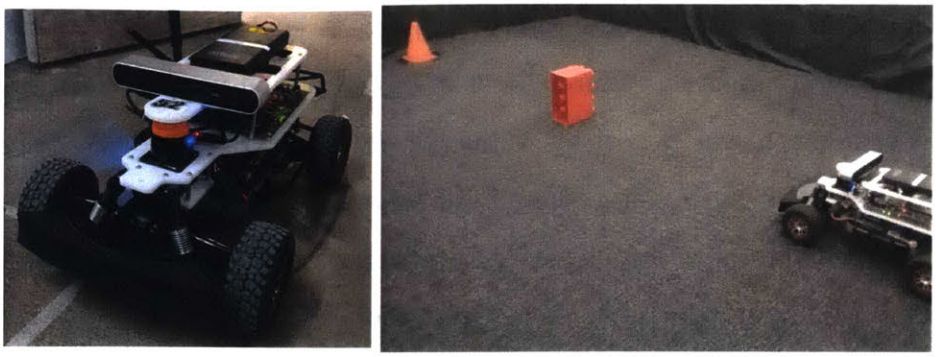
The task of the agent in simulation is to drive towards a goal cone. The simulation environment is a rectangular enclosure with a goal cone at the end, as depicted in the leftmost image in Figure 3-3. ROS [174] is used to direct the car, and the associated RViz package is used to visualize the simulation process.

The state of the robot is defined in terms of 3 feature variables – dx, dy, and $d\theta$. As shown in Figure 3-3, these variables represent the robot's spatial distance from the nearest object of interest (in this case, the orange target cone). We model each of the features as discrete variables, but they can be modelled as continuous as well. Because the real-world camera has a limited field of vision, the simulated robot reproduces this effect by only taking into account objects that lie within 70° of its heading. Given that our environment is small and that the robot most likely sees its target a majority of the time, this is not an exact measure - only an approximation given some tests with the real robot. If no object lies within the specified field of vision, then the robot is given a state of $(-1000, -1000, -1000)$. The state space is characterized by a range of $(-0.105, 2.4)$ meters and $(-1.19, 1.19)$ meters in the x and y directions respectively. The robot's heading was measured using the standard robotics convention - 0 radians is straight, with counterclockwise angles ranging from $(0, \pi]$, and clockwise angles ranging from $(0, -\pi]$. The heading difference $d\theta$ therefore ranged from $[-\pi, \pi]$.

The state representation and various important environmental features can be seen in the middle image in Figure 3-3. The discretization used to learn a successful policy in simulation was 9 possible values of dx, 14 possible values of dy, and 14 possible values of $d\theta$ (including values of -1000 for each). This results in a state space with 1352 possible discrete states, and a Q-table with 4059 entries (3 possible actions per state). Modifying the state space resolution is possible based on the sensing capabilities of the real-world robot.

At each time step, the robot takes one of three discrete actions – forward, left, or right. On the real robot car, each action is executed by publishing a specific steering angle and velocity for a set amount of time. This results in discrete action execution, with each action displacing the robot by approximately 0.3 meters in the real world. In simulation, the same displacement is emulated by driving the robot and adjusting the final position to the nearest discrete state. Although publishing smooth actions in simulation is possible with ROS, each action is mapped to the closest discrete state in order to model a discrete environment. Additionally, waiting for each action to complete can result in unrealistically lengthy training times.

Q-learning is used to learn a policy π_{sim} in the simulation environment. The reward function is defined as follows: the agent receives $+100$ for reaching the goal zone and is penalized -1 for each time step that the agent has not yet reached the goal zone. This reward function encourages the agent to reach the goal as quickly as possible by driving directly towards the object. Each episode is terminated if the agent's discrete state falls outside the rectangular boundary, if the agent reaches the goal zone, or if neither scenario has occurred after 100 time steps. At the start of each episode, the agent is initialized at a randomly selected discrete state. The simulation is run with an initial learning rate parameter $\epsilon = 1.0$ that gradually decreases to $\epsilon = 0.1$. Within 40,000 episodes, the agent learns a policy to reach the goal cone from any starting position.



(a) The robot car used in real-world experiments.

(b) The setup of the real-world environment with an obstacle in the way.

Figure 3-4: Depiction of the real-world environment setup, including the physical robot car and the environment the car was tested in.

3.3.3 Real-world execution

The real-world environment, shown in Figures 3-3 and 3-4b, has a new "dangerous" red block that was never modelled in simulation and obstructs the agent's way towards the goal cone. The robot car's goal in the real world is to navigate towards the orange goal cone at the end of the enclosure, while avoiding the red block in its way. However, the agent is equipped with very simple sensors that hinder the agent's ability to identify detailed attributes of the objects, like shape or size. Due to this limitation, the agent treats all objects as the same, and thus, the agent observes the orange cone and the red block as identical. Deploying the trained agent policy in this real-world environment will result in the agent colliding with the obstacle because the agent has no knowledge of the type of the object it sees.

To run the trained policy in the real world, we use a mini robot car shown in Figure 3-4a from the MIT undergraduate robotics class: Robotics, Science and Systems (6.141) [112]. The car has a remote control car chassis with rear-wheel drive capable of up to 30mph speeds. The main onboard computer is an Nvidia Jetson Tegra X1, with 256-core GP-GPU. With 1 Teraflop of computing power, the car is able to run ROS (Robot Operating System) [174] on an Ubuntu Linux distribution while processing input from various sensors. These include a 2D Hokuyo Laser Rangefinder (lidar), as well as a high quality ZED Stereo Camera. The real-world environment

that the car drives in is a rectangular enclosure similar to the simulation world with a red obstacle obstructing the robot's path towards the goal, shown in Figure 3-4b.

The combination of the different hardware components on the robot allows us to run a variety of robotics algorithms in real time. Data obtained from the sensors is used to obtain the robot's state in the real world. The agent's representation of the world is again the representation used for training: $[dx, dy, d\theta]$. We run a particle filter localization algorithm [69] in order to accurately obtain the robot's spatial state when recording human demonstrations. We also run a color segmentation algorithm that is capable of detecting bounding boxes around objects of interest in an image [147]. This algorithm allows us to identify the nearest object in the real world, and derive the robot's observation of the state when running the final blind spot classifier trials. We now discuss the localization and the object detection components in greater detail.

Localization

To determine the car's position in the world, we run a particle filter localization algorithm [69]. This method uses odometry to derive "particles", which are possible robot poses in the world listed as (x, y, and heading) tuples. The robot models the noisiness of its odometry data by maintaining a set of multiple possible states at which it could be located after a discrete time step. This set is trimmed down using a probabilistic model that takes into account the robot's lidar sensor reading.

Using ray tracing originating at the candidate state and a complete knowledge of the environment, a set of lidar readings corresponding to each candidate state is obtained. These candidate probabilities are compared to the actual readings gathered by the robot and are assigned a probability according to a manually defined probability distribution. The highest probability particles are averaged to determine the robot's most-likely pose. The localization algorithm is fairly precise and is a good way of accurately determining the robot's pose while a human teleoperates the robot car during demonstrations.

Object Detection

When running the final blind spot model in the real world, a vision module is used to find the position of the nearest object using color segmentation [147]. Objects in the red to orange color range (which includes both the cone and the obstacle) are segmented, and bounding boxes are drawn around them. Then, a planar homography is used to convert pixel coordinates from the center of the bounding box's bottom edge to ground plane spatial coordinates in the real world.

More specifically, the segmentation algorithm works by taking as input a static image frame from the robot's ZED Stereo Camera sensor and finding contours around continuous bodies of pixels that fall within a defined RGB color range. A bounding box is drawn around the derived contours. OpenCV [30] is used to perform color segmentation, contour detection, and bounding box calculations. If multiple objects are detected (both the orange cone and the red block in this case), the closest one is chosen. Given a bounding box for the nearest object, a spatial state is derived using a simple planar homography. This homography maps pixel coordinates in the image to spatial coordinates on the ground plane of the environment. The agent's observation of the state can then be computed using the estimated location of the closest object along with the predicted robot location from the localization component. In our setting, we were able to run the vision algorithm in real time, but as the complexity of the domain grows, making the full system run quickly can be a challenge.

3.3.4 Learning a blind spot model

Expert demonstration data is used to learn a blind spot model for the agent in the real world. If at a certain state, an expert takes an action that differs from the policy learned in simulation, then that corresponding agent observation could potentially be a blind spot. Expert demonstrations can be provided in different ways. An expert agent can correct the autonomous agent as it executes a policy in the real world, or the human can drive the car in the real world directly. In our application, we used human demonstrations to train the blind spot model. The person has knowledge about the correct actions to be taken in the real-world environment because he or she observes the true state of the world and knows the objective of the task.

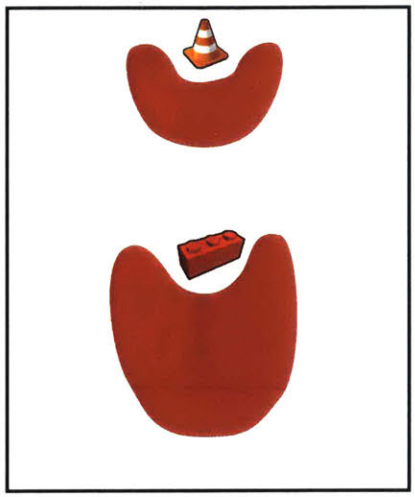


Figure 3-5: A visualization of what we predict will be blind spot regions for the robot in the rectangular enclosure environment. Red patches denote blind spot zones where the agent should query for help. Note that these red regions denote only the dx and dy components of the state space, and do not include the heading difference component $d\theta$.

To provide human demonstrations, the car was driven with a video game console controller, with the human pilot deciding what discrete actions to take and when to record them. Three separate buttons corresponded to the three different robot actions (left, right, and forward), and another button was used to record the state of the robot and the last action taken. Using this framework, the human selects an action and then records the resulting state, creating a dataset of state-action pairs that follow a near-optimal policy.

Nine different expert demonstrations were collected and used to generate a blind spot model, each with the car starting at the opposite end of the environment from the orange cone. The car was piloted towards the goal cone and state-action pairs were recorded along the way. Many trajectories around the cone were taken in order to provide expert demonstration data in various regions of the world.

We follow the approach detailed in Section 2.4 to learn a blind spot model given the learned policy from simulation and the expert demonstrations in the real world. The learned blind spot model M maps observations in the agent's representation $\mathcal{S}_{agent}$ to either a label of 1 (blind spot) or 0 (safe).

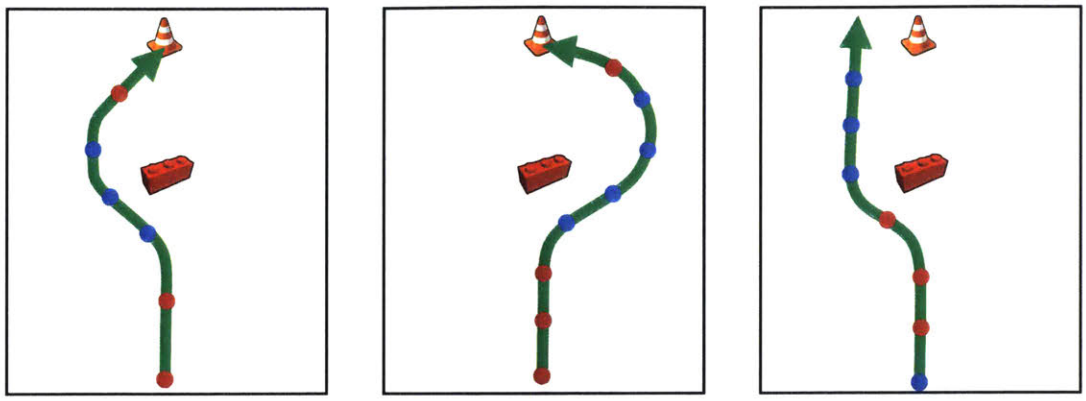


Figure 3-6: Trajectories for three trials using the blind spot model classifier to query for help in the real world. The green arrow highlights the continuous path taken by the robot. Red dots denote discrete states where the agent queried for help (blind spots), while blue dots represent discrete states where the agent used the simulation policy to take actions autonomously (safe states).

3.3.5 Safe execution using learned model to query

When deploying the robot in the real world, the learned blind spot model is used to query for help when there is a high chance of a blind spot. Specifically, at each state the robot visits $s_{real} \in \mathcal{S}_{real}$, the robot queries the blind spot model. If the model predicts safe for a specific observation ($M(o_{agent}) = 0$, where $s_{real} \mapsto o_{agent}$), the robot executes the action learned from simulation: $\pi_{sim}(o_{agent})$. If the model predicts blind spot ($M(o_{agent}) = 1$), the robot stops and queries a human for the best action to take. A person then inputs the best discrete action the robot can take in that state through a simple terminal interface. The robot executes the suggested action and continues by again querying the blind spot model and repeating.

In our experiments, the robot learned to intelligently query for help in regions close to the red obstacle as well as close to the orange cone and learned to drive autonomously in regions far away from the objects, similar to the predicted behavior in Figure 3-5. The reason the robot queries near the cone is because it cannot distinguish between states near the orange cone and states near the obstacle. Both objects look identical due to the agent's limited representation. The resulting trajectories of three

Table 3.1: Summary of the final demonstration trials. The table indicates whether each trial was a successful trajectory (no collision with the obstacle) and reports blind spot query percentages, calculated as $\frac{Blindspot\ Queries}{Total\ Actions}$.

Trial	1	2	3	4	5	6	7	8	9	10	11
Success?	1	1	1	0	1	1	0	1	1	1	1
% Queries	42.8	37.5	42.8	42.8	50.0	50.0	50.0	42.8	42.8	40.0	42.8

separate final trial runs are shown in Figure 3-6.

Eleven trials were run, and the trajectory plots for all trials are provided in Appendix A. Overall, 82% of the trials demonstrated the desired behavior by avoiding the dangerous obstacle in the center, and reaching the goal cone. We include a summary of the trials in Table 3.1, specifying whether each trial succeeded and the percentage of times the agent queried the expert for help.

Two of the trials were not successful due to a collision with the dangerous obstacle. These trials achieved a trajectory very similar to the desired route, as shown in Appendix A, but simply did not leave enough room to turn around the obstacle. This error can be potentially attributed to a slight error in the modeling of actions when learning a policy in simulation. In the real world, each action was affected by noise, with turns being especially prone to random error. This can be improved by calibrating the system frequently. The experiment highlights the need to address various forms of noise during training and execution. Additionally, even successful trajectories were not always optimal. This is partly due to flaws in the blind spot approach itself, and partly due to mechanical failures when driving the robot. When actions failed to fully complete, the drive command had to be reissued, resulting in trajectory errors. Overall, the approach helped to reduce errors in the real world (colliding with the obstacle). In the next section, we discuss a few limitations of the approach when applied to the car domain and some ideas of how to improve the performance in more complex variations of the task.

3.3.6 Discussion

We have demonstrated the application of blind spot model learning on an autonomous car domain and have shown that human demonstration data can be used to accurately predict potential errors of an autonomous car. The final result was that the robot queried for help intelligently to avoid dangerous regions and reach the goal. Through these experiments, we found the blind spot framework to be general enough to be applied to more real-world scenarios, such as this mini autonomous driving application. The method only requires access to a learned policy and expert feedback (demonstrations or corrections). While we used reinforcement learning to obtain a policy in simulation, any other technique can be substituted as well. Despite the flexibility of the method, there are several limitations of the full pipeline that lead to occasional errors.

The first challenge was a standard issue in reinforcement learning, which is what we used to train the agent's policy in simulation. Specifically, the construction of the training environment, including the specification of the states, actions, transition function, and reward function for the MDP, was challenging. The agent's representation was chosen to be discrete to make it easier to learn a simulation policy. Since the natural state and action spaces of the car are continuous, constructing a discretization that was appropriate was difficult. If the discretization was too small, Q-learning took a long time to run, while if the discretization was too large, the agent could not learn rich enough policies that allowed for varied behavior in different regions of the state space. Further, the action space granularity needed to be chosen such that it aligned with the granularity of the agent's state representation. Multiple reward functions were also explored to enable an agent to learn to reach the goal. The challenge of constructing the correct reward function for a task is a known problem and can be addressed through various techniques, such as inverse reinforcement learning [1, 154, 254, 180] or human preference elicitation [50, 61].

The second challenge was training an accurate and informative blind spot model. The main factors that affect the quality of the learned model are: 1) correlation

between features in the agent's representation and missing blind spot features, 2) the simulation policy learned from training, and 3) the quality, quantity, and variety of human data collected.

In our experiments, many of the blind spots were detected in correct regions, but the predicted blind spots were quite broad and lead to an overly cautious agent. More informative blind spot models can be learned with a richer agent representation, which can be finer discretization, access to more features that correlate with blind spots, etc. Having a lower granularity representation, for example, results in longer training times, but would also allow the system to have more accurate blind spot predictions in the real world.

Further, limited sensing and flawed observational capabilities can lead to high amounts of state representation (SR) noise. This refers to the phenomenon where an agent receives many different labels from a human because many real-world states look identical to the agent. This could lead to overly cautious behavior in complex real-world environments. In this experiment, although the agent was able to recognize regions near the red block as dangerous, the agent also predicted regions near the goal cone to be unsafe. SR noise can be improved by augmenting the agent's representation to reduce the number of real-world states mapping to the same agent observation.

Given an adequate representation, it is still important to learn a well-trained policy in simulation. There were many simplifications that made it easier to train an agent in the simulation environment. For example, using continuous actions by publishing drive commands in ROS was not feasible given the robot's limited turning radius and the small size of the environment. In this scenario, no set of continuous actions that could realistically be executed on the robot accurately mapped to a discrete grid. Choosing to move the robot by teleporting it to the next closest state allowed the agent to learn a policy, but affected the accuracy of such a policy. This had a negligible effect given the small size of the test environment. However, given a larger space, aggregating slight positional errors after each action taken could result in potentially large sources of error when learning a policy in simulation. As a result, this could prevent the agent from learning a policy that transfers well to the real

world.

Finally, the effectiveness of the learned blind spot model depended highly on the quality, quantity, and variety of human demonstrations provided. It is important for the human to provide acceptable, if not optimal, demonstrations of the task in the real world. If many errors exist in the data, it will be difficult for the robot to learn an accurate blind spot model. The quantity of data can also affect the performance of the model learning, as more data provides more evidence for classifying states as either safe or blind spot. In our experiment, we collected a decent set of samples of human demonstrations and obtained successful functionality on a limited environment. However, for more complex environments, a much larger quantity of expert feedback might be required in order to achieve proper agent behavior.

Finally, variety in the data is important for generalizing well to the agent's full feature space. If demonstrations are provided in a limited part of the world, the blind spot model may not generalize well to other regions. In our experiments, we collected data only on one variation of the task. While the learned blind spot model was informative for that particular task, human data collected across a large variety of environments will enable better generalization. Further, demonstrations can have action-mismatch (AM) noise, which is when agent and human actions differ but this does not indicate an unacceptable action. These mismatches in behavior can lead to an overly cautious agent and more importantly, limit the generalizability of the learned blind spot model.

Another method for increasing variety in the data and to remove AM noise is to collect corrections data, in which an agent acts in the world while being monitored by a human. The set of states the agent visits according to the trained simulation policy will be different than the states visited by a human in demonstrations. Collecting additional corrections data to augment the human's demonstrations can greatly improve state exposure and lead to much higher performance, as shown in Section 2.8.2. In addition to human demonstrations and corrections, it might be interesting to investigate other forms of feedback, such as preference elicitation [50, 61] and natural language [92, 152].

Another component of the pipeline that can be explored in future work is the aggregation scheme. We observed that the approach we used, Dawid-Skene, only learns to differentiate between two groups of states. The algorithm was sufficient for learning reasonable blind spots in this experiment, but in cases where there are a larger number of clusters of states with different observations, it could be interesting to investigate other methods of aggregating and processing expert data. Finally, scaling the system to even more complex real-world environments using advanced robotic platforms is an avenue for future work.

3.3.7 Conclusion

The agent blind spot modelling approach from Section 2.4 was applied on an autonomous car domain. The car was trained in a simulation environment where the goal was to reach a cone. In the real-world environment, an obstacle, which the agent had never seen before, obstructed the agent's path towards the goal cone. Due to limited sensing, the agent could only sense simple properties of objects around it and thus, could not tell the difference between these two objects. Running the agent's simulation policy in the real world would cause a collision with the obstacle. Instead, a human provided demonstrations by teleoperating the robot around the obstacle. The simulation policy along with the demonstrations were used to train a blind spot model. Finally, we show a demonstration of the full system working, in which the robot car queries for help in appropriate regions for safe task execution in the real world. While simulation environments and robotic sensing capabilities are rapidly improving, small imperfections will almost always remain. Learning to predict and mitigate the potentially dangerous effects of these imperfections is a crucial step towards deploying simulation-trained intelligent agents at scale.

Chapter 4

Discovering Human Errors

4.1 Introduction

So far, this thesis has considered the problem in which an agent has a limited representation of the world, while a human or oracle helper has access to the true state. The proposed approach uses human feedback in order to identify agent errors due to representation incompleteness, and experiments on simulated domains as well as a real-world autonomous car application show the benefit of our model in querying for human help intelligently, leading to safe execution in the world.

In this chapter, the opposite setting is considered, in which an agent is assumed to have the true representation of the world, while a human performing the task has a limited view, which can lead to errors. People are often prone to making preventable errors in many applications, such as driving, aviation, and medical diagnosis. Some of these errors are due to factors, such as fatigue, lack of training, and carelessness, which can be reduced through improved training. However, other errors are due to the inability of people to observe important factors needed for decision-making. For example, glare from the sun or a buildup of lots of ice might hinder a person's ability to see dangerous obstacles ahead. Understanding the cause of errors can be an important first step towards fixing the mistake.

To determine the cause of human errors, we present a generative model of human decision-making and explicitly separate errors that occur due to limitations in per-

ception vs. other errors from an incorrect policy. Learning this distinction can be useful because if an error is caused by limited perception, we can augment human sensing by adding technologies or modifications to the world that allow a person to "see" what they cannot (i.e., an indicator that notifies a person when there is an obstacle ahead). If the error is not due to a representational limitation, we can put resources into providing customized training to reduce these errors.

Consider a task in a restaurant kitchen, where chefs have to quickly and correctly make dishes for many customer orders. A new chef on their first day may make many mistakes due to the organization of the kitchen, their memory of the dishes, fatigue, etc. Understanding the cause of errors can enable appropriate modifications of either the kitchen itself or of training for the new chefs. For example, if certain ingredients are not appropriately labelled, chefs may make consistent mistakes, and this issue can be easily fixed by putting labels on specific ingredients that are difficult to distinguish. An error due to fatigue, on the other hand, would need to be handled in a different way, such as through more frequent breaks.

In our problem setup, the agent who is reasoning about human errors has full knowledge of the world, including the true representation as well as the optimal policy. We assume access to demonstration data of the human performing the task and labels of when errors occurred. We consider two possible sources of errors: those that occur due to blind spots, which are factors of the true representation that are unobservable to the human, or other policy errors, which we group together in this work as human noise. The goal is to recover both of these given demonstration data.

To separate the two causes of errors, our generative model explicitly encodes these two latent factors. One is modelled as the human's blind spots, which is a binary representation of which features the human can/cannot observe. The blind spots affect the human's view of the world, as the person can only observe a subset of features. To explain the human's actions, our model aims to fill in missing features with values that the human might be assuming about the world. For example, if two ingredients look identical, the person may make an assumption about which ingredients they are and move on with the task given that assumption. Our generative

model can then use the known optimal policy along with the human's view of the world to explain human actions. Since the human may not always follow the optimal policy, there is an additional latent variable, human noise, that affects the action selection as well. Using Bayesian inference techniques, we can recover the human's blind spots, as well as the amount of noise in human task execution.

Experiments on a gridworld domain show that we can successfully recover which features are not being observed by a simulated agent, our proxy for a human, and the amount of noise in the data. Further, we collect data from users on a kitchen task where some ingredients were purposefully confusing and evaluate the model's ability to separate representation errors from other sources of error. Our approach is able to learn that ingredients not sufficiently labelled in the kitchen can lead to systematic errors on the task. This knowledge can motivate the need to revise the environment so people have a more accurate view of the world. The proposed generative framework is flexible, and can be augmented based on domain-specific assumptions on observable variables or conditional independencies.

The problem formulation is described with an agent as the oracle with the true state information and a human as the flawed actor as it is motivated by understanding the errors of people. However, the approach is general and can be applied to understand the errors of any AI system. Our model provides a method for further understanding and fixing of human or AI system errors. Iteratively identifying and reducing these costly errors through such models can be extremely beneficial for helping humans and AI systems be more safe.

4.2 Related Work

In this section, prior work related to human error discovery is described. First, we discuss literature on human-machine interaction because modelling humans is an important component of understanding human errors. We next outline related work in reinforcement learning, specifically on model-based RL, multi-agent RL, and partially observable MDP settings. A discussion of human decision-making and errors from

human cognition is outlined. Finally, we include literature on learning representations and how that relates to the problem of determining human flawed representation as a cause for human errors.

4.2.1 Human-machine interaction

There has been a large body of work in human-machine interaction that learns behavior models of human teammates [229, 173, 158, 155, 125] and facilitates communication in human-agent teams for better understanding of both teammates' goals [172, 197]. One takeaway from many of these works is that modelling humans is a key component of effective human-machine collaboration [111]. A survey article [223] discusses approaches for learning about human partners, such as understanding human intent through attention parsing and through Theory of Mind, which is the ability to determine the mental state of another person.

Several works [40, 41] emphasize the benefit of estimating human mental models and incorporating them into planning problems to make learned plans more effective. One work in human-robot team training [159] proposes a framework to train robots and humans in each other's roles to facilitate convergence of mental models. Talamadupula et al [216] uses mental models in plan recognition. Agent beliefs are incorporated into the planning problem to better predict agent plans. One work [32] uses natural language to better communicate beliefs and goals. Breazeal et al [31] show that non-verbal communication enables more efficient and robust human-robot interaction. Active learning [36, 46, 35] is also used in human-robot teams to enable robots to ask good questions.

Some works focus on improving the interaction between AI systems and people, not just modelling human teammates. Amershi et al [6] broadly discuss many factors that can improve the development of interactive systems, such as providing an opportunity for users to query a model or indicate preferences. One work [18] highlights the importance of updating AI systems in a way that is compatible with a human's mental model of the system. If AI systems are modified with changes that result in new mistakes that the system was previously proficient at, people get confused

with respect to the types of mistakes the machine makes. They introduce a training objective that results in more compatible updates so that people's trust in various parts of the system can continue to be accurate predictors of model performance.

Nikolaidis et al. [157] present a game-theoretic model of human adaptation in a collaborative task, where a robot chooses between taking actions that reveal its capabilities to a human vs. taking the best action given the human's current information. While they consider a scenario in which the human has partial information, the human is not assumed to make mistakes, as in our work. Overall, many works in human-machine interaction enable more effective collaboration, but they are not focused on learning about errors of human teammates.

4.2.2 Reinforcement learning

Model-based and model-free reinforcement learning (RL) can be used to represent agent and human behaviors when performing a task [1, 254, 146, 210, 90, 64]. When trying to explain errors of humans, it is helpful to understand *why* the human took certain actions and *why* a mistake occurred, which involves understanding causal knowledge about the world and the human policy. Both model-free and model-based RL involve modeling causal knowledge [74] through hidden state inference. Work by Momennejad et al [148] leverages the benefit of both RL methods by using successor representations (SR), which keep track of likely future states. While model-based RL is an effective tool to better understand the world and decisions made in that world, it cannot be used directly to understand human or agent errors.

Multi-agent reinforcement learning [130, 217, 124, 87, 68, 202] is also relevant because many works in this area develop methods for working with several agents, whose beliefs and observations are unknown. Predicting the observations, intentions, and actions of a team of agents is important for working well in a team. In work by Iqbal et al [97], relevant information from each agent is selected to maximize joint performance. Another work [98] coordinates exploration between many agents to improve efficiency in exploring an environment. Latent goals of teammates can also be inferred [179] to improve joint performance. Similar to our work in which an

omniscient agent is observing a human performing a task, Torrey et al [228] model a teacher-student relationship where an agent advises another student agent to help the student better learn to perform the task. However, the majority of these works focus on improving joint team performance, rather than identifying errors of other agents.

Partially observable MDPs [108, 224, 208, 144, 95, 170] are also highly relevant because in this formulation, agents cannot directly observe the true state of the world and instead maintain a distribution over possible beliefs. POMDPs can be used to infer latent states, but solving them is often intractable. Many techniques are used to generate solutions for these problems, such as online heuristic search [190], Monte-Carlo tree search [206], and reduction to tractable, constrained settings [153]. POMDPs are additionally used to identify human errors in the context of assistive AI systems [104]. The system detects human errors and provides assistance to help people complete the specified task. While POMDPs are useful frameworks for modeling human representation and errors, they are not directly applicable for identifying whether errors originate from limited representation.

4.2.3 Human cognition

Prior cognitive science literature [238, 54, 16, 212, 79, 164, 234] explores models that simulate human decision-making processes. Works in this space can be useful for identifying the reasons for various human decisions and even human errors. Griffiths et al [85] argue that top-down probabilistic models can better represent human decisions and generalize compared with bottom-up connectionist approaches. In another work [86], the authors conduct a study where users are instructed to predict various common phenomena, such as lengths of movies or marriages. Results show that human predictions are close to optimal decisions, and errors in predictions are used to better understand people's assumptions about the world and their decision-making process for arriving at a decision. For example, for novel scenarios, people often draw analogies to similar settings they know well and adjust their predictions accordingly. Goodman et al [80] propose a technique for understanding human concept learning by

leveraging a concept language and performing inference based on this representation.

In addition to understanding human decision-making, several works develop models to understand agent modelling of other agents. In work by Ullman et al [234], a method is proposed to infer an agent's intention, specifically focused on whether an agent is helping or hindering another agent's goals. The approach uses inverse planning techniques that rely on the assumption that an agent is rational given its context. Baker et al [14, 15] develop a Bayesian framework to explain how people predict another agent's actions based on observations of task execution. The model enables prediction of goals as well as actions under novel scenarios. While this setup is similar to ours in that we also take as input demonstration data and aim to understand the decision-making process leading to selected actions, we are focused more on determining the source of errors, rather than determining the agent's goal.

Saxe [196] discusses different theories for how people explain and understand the actions of others. One is called Simulation Theory, which proposes the idea that people can understand others by using their own mind as a model of the other person's mind. While this theory can represent errors due to flawed inputs, it does not explain why the inputs were wrong in that particular pattern, which is important for accurately understanding reasons for errors. This work argues against Simulation Theory and gives several examples for why a theory of psychology is necessary to explain people's actions as well as people's errors. Overall, understanding human cognition is quite relevant at a high-level to our work, but our goal is different, as we would like to distinguish the cause of errors as either stemming from representation limitations or from other factors.

4.2.4 Representation learning

Learning appropriate state representations and identifying when a representation [24, 175, 59, 241] is insufficient is also relevant to this work because we consider a setting where an agent and a human operate in different representations, and the agent is tasked with understanding human errors that can occur from a human's flawed representation. In work by Diuk et al [59], object-based representations are used for

MDPs as a natural and general way of describing the world. Other works [58, 103, 114, 205, 129] also use objects as a key component of their learning algorithms for improved transfer to new tasks and increased interpretability.

Many works in reinforcement learning [101, 240] also develop approaches to learn representations automatically that generalize well across many tasks. In work by Ma et al [135], a successor representation is learned, which includes general knowledge that can be transferred between many related tasks. Another work [134] combines ideas from symbolic planning and deep reinforcement learning to handle both high-dimensional input while still learning task-level symbolic actions. Evaluations show that the method leads to increased interpretability as well as improved sample efficiency. Several works [150, 89] develop approaches for representation learning in hierarchical RL scenarios, where it is important to learn generalized representations that connect controllers from different levels. Jonschkowski and Brock [105] consider representation learning in robotics, where they leverage prior knowledge about the physics of the world to learn useful state representations. While learning these representations can be useful in understanding human decision-making, we identify what the human cannot observe and how this deficiency can lead to costly errors.

Some works specifically address flawed representations. In [235], an agent has a limited representation and uses Bayesian nonparametric techniques to learn unmodeled features. Our related work in identifying agent blind spots [181, 182] similarly assume that a flawed agent is learning from an oracle demonstrator. These settings assume the reverse scenario that an agent is learning using expert feedback. We consider the scenario of identifying human representation limitations given demonstration data.

4.3 Human Error Model

We model the generative process of human errors in order to identify the reason due to which people make different types of mistakes. We focus on two main sources for errors: representational errors and training errors. Representational errors, or

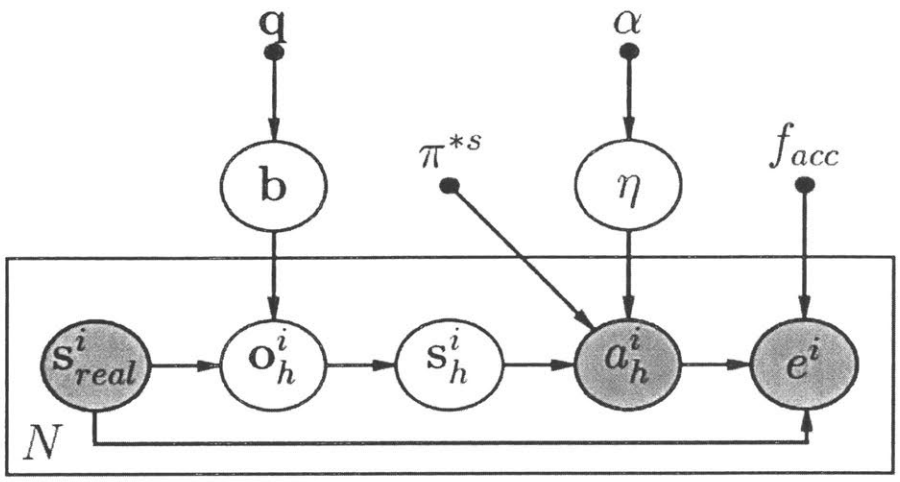


Figure 4-1: Graphical model representing the generative process for how humans make errors.

blind spots, are due to the person's inability to observe critical components of the task, which can lead to errors in execution. Training errors occur when the person views the world correctly but still makes a mistake due to factors, such as limited practice, slow reaction times, carelessness, random noise in execution, etc. We assume that the agent using this model to reason about the human's errors knows the true representation and optimal policy.

The input to our model is a set of human demonstrations: $D = \{(\mathbf{s}^i_{real}, a^i_h, e^i)\}, i \in [1, ..., N]$, a list of state-action-error tuples. The true state of the world $\mathbf{s}^i_{real} = [f_1, f_2..., f_k]$ is represented as a vector of features. The human action $a^i_h \in \mathcal{A}_h$ and an error indicator $e^i \in \{0, 1\}$ are observations that provide information about the human's decision-making and where errors occurred. We assume that the agent has access to an acceptable function $f_{acc}(s, a)$, which returns 1 if action a is acceptable in state s and 0 otherwise. Thus, given a state-action pair, the error can be computed using $f_{acc}(s, a)$, resulting in the observed demonstration data D.

Given these demonstrations, we want to identify whether errors are caused by representational limitations or due to errors in policy execution. To account for representational errors, we assume that the human might not be able to see the true state of the world $\mathbf{s}^i_{real}$. Instead, the human receives an observation $\mathbf{o}^i_h$, which is a

transformed version of $\mathbf{s}^i_{real}$. We define the transformation using a blind spot vector $\mathbf{b} = [b_1, ..., b_k], \mathbf{b}_i \in \{0, 1\}$, which encodes information about the features the person cannot observe (i.e., when $\mathbf{b}_i = 1$, the person cannot observe feature f_i of the true state). Given the true state $\mathbf{s}^i_{real}$ and the blind spot vector $\mathbf{b}$, we can obtain the human's flawed observation $\mathbf{o}^i_h$. The generative process is as follows:

$$\mathbf{b} \sim \text{Bernoulli}(\mathbf{q})$$

$$\mathbf{o}^i_h \sim Pr(\mathbf{o}^i_h | \mathbf{s}_i, \mathbf{b}) \; \forall i$$

We model the blind spot $\mathbf{b}$ as a mask over the state $\mathbf{s}^i_{real}$ to compute the observation $\mathbf{o}^i_h$; however, generally, the observation can be represented as an arbitrary 'blind spot'-dependent transformation of the state.

Now that we have the human's observation of the world, we want to explain human actions to understand when these decisions cause costly errors. We assume access to the optimal policy $\pi^{*s} : \mathcal{S}_{real} \rightarrow \mathcal{A}_h$, which is the optimal policy with respect to the true state of the world $\mathbf{s}_{real} \in \mathcal{S}_{real}$. However, since the person's observation is not identical to the true state, the representation of the true policy and the representation of the human policy are different. To bridge this gap, we propose to estimate the human's implicit view of the world in terms of the true state representation, where values for missing features denote the human's assumption of that feature without the ability to perceive it. For example, if a person is red-green color-blind, and we observe that she keeps driving at a traffic signal, we may assume that her implicit assumption of the traffic light is that it displays green, which explains her action of continuing ahead.

We map the human's observation $\mathbf{o}_i$ to the most likely state the human is predicting $\mathbf{s}^i_h$. Explicitly modelling this state from the human's point of view helps us to reason about the actions the human takes. Given $\mathbf{s}^i_h$ and our known optimal policy π^{*s}, we can compute the most likely human action.

However, people are rarely always optimal, so we include an additional variable η that denotes noise in execution. To model different levels of expertise in policy execution, we model η at three discrete levels: expert (1% noise), intermediate (10%

noise), and novice (30% noise). This variable intuitively captures how often the human deviates from the optimal policy given $\mathbf{s}_h^i$. Thus, the action is derived from the estimated human state, the optimal policy, and the noise in execution. The error can be computed using the known acceptable function. We sample actions and errors as follows. Figure 4-1 depicts the full graphical model.

$$\begin{aligned}
\mathbf{s}_h^i &\sim Pr(\mathbf{s}_h^i|\mathbf{o}_h^i)\ \forall i \\
\eta &\sim \text{Multinomial}(\alpha) \\
a_h^i &\sim \pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta)\ \forall i \\
e^i &\sim f_{acc}(e^i|\mathbf{s}_{real}^i, a_h^i)\ \forall i
\end{aligned}$$

With this generative model, our goal is to learn $P(\mathbf{b}, \eta|D)$: the probability of both the blind spot vector $\mathbf{b}$ and the noise parameter η given human demonstration data D. We use variable elimination and obtain the following result. The full derivation is included in Appendix B.

$$= \frac{P(\mathbf{b})P(\eta)\prod_i \sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) \sum_{\mathbf{s}_h^i} P(\mathbf{s}_h^i|\mathbf{o}_h^i)\pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta)}{\sum_{\mathbf{b},\eta} P(\mathbf{b})P(\eta)\prod_i \sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) \sum_{\mathbf{s}_h^i} P(\mathbf{s}_h^i|\mathbf{o}_h^i)\pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta)} \tag{4.1}$$

4.4 Experiments

The model is evaluated on two domains. The first is a gridworld environment, where we simulate a human that is color-blind and cannot recognize the difference between various colored objects. We show that our approach enables inference of both the simulated human's blind spots and noise in execution. The second domain is a kitchen task, in which human users are instructed to study a menu of several dishes and make them from memory. Some ingredients, such as salt and sugar, are indistinguishable, which results in systematic errors, switching the two. Further, due to the difficulty of the task, people made other random errors. Results show that we can similarly recover the sources of human errors, which can inform the redesign of the kitchen

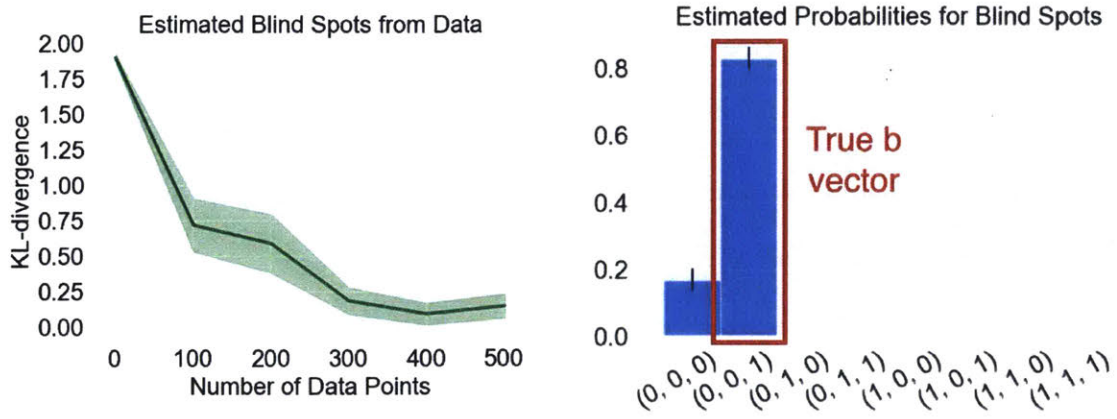


Figure 4-2: Estimated blind spots given demonstration data. Vector (0,0,1) means the object color feature is a blind spot for the human.

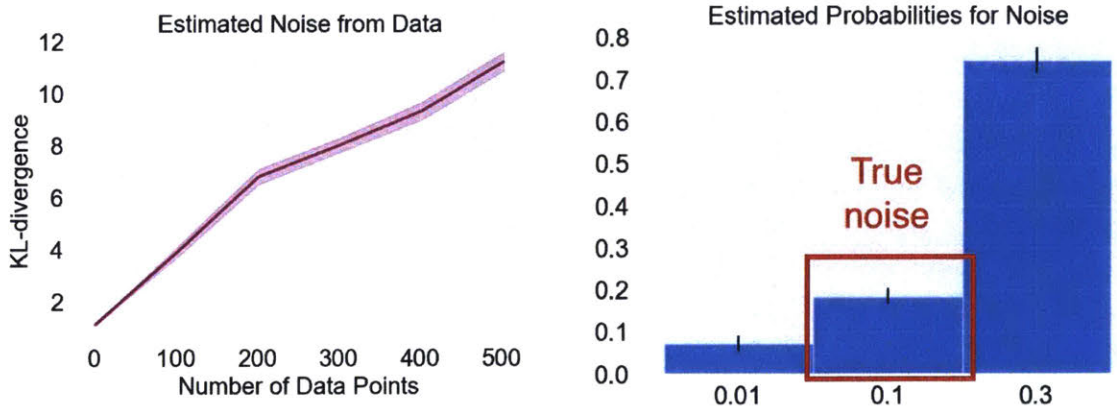


Figure 4-3: Estimated noise in execution η given demonstration data.

environment so people have a more accurate view of the world. We now discuss details of the experiments below.

4.4.1 Gridworld domain

We conduct experiments in a gridworld domain. The world is a 10x10 grid with one object, colored green or red, placed randomly in one of the 100 locations. The human is tasked with collecting green objects and staying far from red objects. The features representing the state are the x-distance from the object, the y-distance from the object, and the object color. We simulate a human that is color-blind and cannot tell the difference between green and red.

Our approach recovers human blind spots

We first show that we can recover the blind spot vector given demonstration data. We generate data from a simulated human that cannot see color and implicitly assumes the object is green. We run variable elimination to estimate $P(\mathbf{b}, \eta|D)$, averaged over 100 runs. Figure 4-2 plots the variation of the KL-divergence between the estimated $\mathbf{b}$ and the true distribution with respect to the size of the demonstration dataset. The estimate of $\mathbf{b}$ gets closer to the true vector, shown by the decreasing KL-divergence. The histogram shows the estimated probabilities for each blind spot vector when the data consists of 100 datapoints. The model correctly predicts the true vector (0,0,1) with 80% probability.

Noise is more challenging to estimate

Our model had a more difficult time estimating the noise parameter. Figure 4-3 shows the KL-divergence of the estimated η and the true distribution. Our approach infers 30% noise as more likely than 10% noise, the true value, because a higher (but incorrect) value of execution noise better explains the errors in the data. This suggests the need for regularization and more informative priors for estimating noise. This result raises the question of whether adding the noise parameter is useful if the model cannot estimate it exactly.

Modeling noise improves blind spot inference

We next compare a version of our model with no explicit noise parameter (we include a fixed ϵ=0.01 deviation to avoid probabilities of 0) to a model that estimates η, averaged over 100 runs. Figure 4-4 shows that when the human is acting with 30% noise, the model without η attributes the noisy actions to confused estimates for the blind spots. The model that estimates η is able to explain much of this noise and recover the true blind spot vector. This shows that while it is difficult to exactly estimate this parameter, estimating η provides more flexibility to the model, resulting in better estimates of blind spots. To improve the estimate of η, we can incorporate

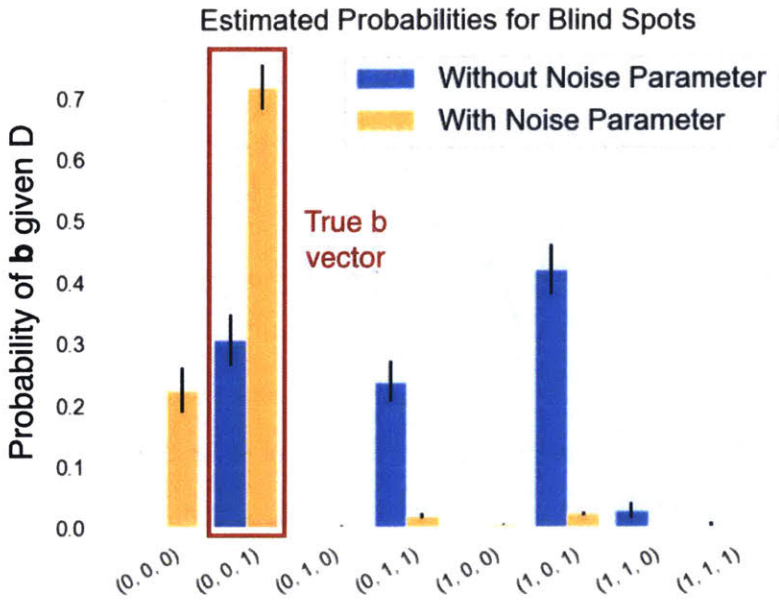


Figure 4-4: Adding the noise parameter results in more accurate estimates of blind spots with noisy demonstration data.

more informative priors for human noise.

Estimated implicit state representation captures human assumptions on missing features

Another interesting insight that can be learned from our model is the most likely estimate of $\mathbf{s}_h^i$ for a given data point. We include the derivation for $P(\mathbf{s}_h^i|D)$ in Appendix B. The highest probability $\mathbf{s}_h^i$ captures information about what assumptions the human might be making on the features she does not observe. Figure 4-5 is a visualization showing that our model estimates that the human's actions match with 70% probability to an optimal policy assuming the object is green. Note that this could either mean that the human assigns a probability every time and makes a decision under uncertainty or that the human 7 out of 10 times assumes green and 3 out of 10 times red. Our model does not distinguish between the two cases.

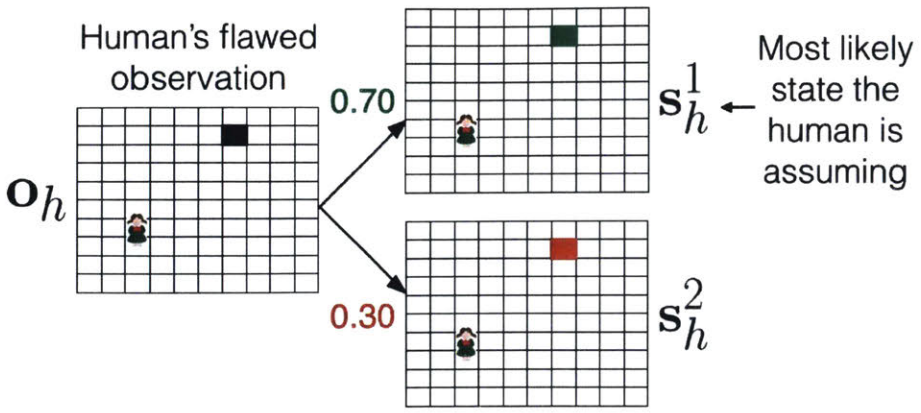


Figure 4-5: The implicit state that the human might be assuming in order to make action decisions.

4.4.2 Kitchen domain

The second evaluation is on a kitchen domain, in which people are placed in a new kitchen and are tasked with making several dishes using the available ingredients. Participants are told that they are renowned chefs and that their friend, who is sick, needs them to sub in for them at their restaurant. They are suddenly introduced into this new kitchen environment with no information about where things are or what the dishes of the restaurant are. They are given two minutes to study the provided restaurant menu, which has 3 dishes with 7 ingredients each. The order of the ingredients does not matter. Participants only need to make sure that all the correct ingredients are in each dish.

At the end of the 2-minute period, participants are tasked with preparing dishes for customers' orders based on memory. Twenty-five dishes for each participant are selected randomly ahead of time, each sampled from a uniform probability distribution over the three possible dishes. Every 5 dishes, participants receive a one-minute break, where they get a chance to restudy the menu before heading back to the busy kitchen.

Participants are not given any information about what ingredients are in the kitchen. Thus, they are forced to make assumptions about which ingredients to use based on the menu and the items in the kitchen area. While most ingredients are recognizable for the majority of participants, we intentionally put salt and sugar in

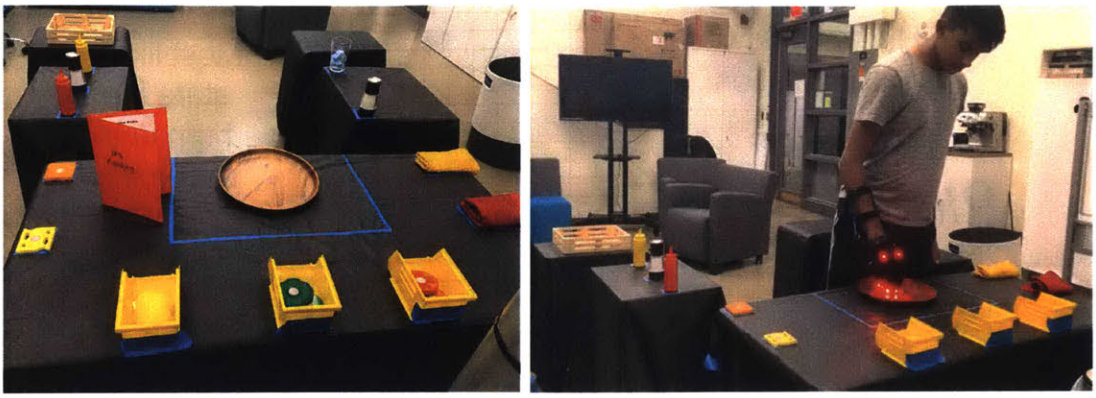


Figure 4-6: Setup for the kitchen task, in which participants learn a menu of a few dishes and make them from memory in the kitchen. Salt and sugar are intentionally unrecognizable, but their placement in the kitchen leads people to have strong priors on which ingredient they use. Using our model, we can recover that people make systematic errors, replacing salt and sugar in the dishes. This discovery enables a reorganization of the kitchen with labels for confusing ingredients to reduce errors due to partial observability of the world.

the menu without disambiguating them in the kitchen. We put sugar near ketchup and mustard and salt near tea. Since the menu always put tea and sugar together and it seems reasonable for salt to be close to the other condiments, this resulted in most participants confusing the two ingredients. Participants also made many other errors due to their inability to remember many dishes given the time constraint. The hypothesis is that our model will be able to separate consistent mistakes from partial observability of the locations of salt and sugar from other memory errors.

We collected data from 12 participants, and each made 25 randomly selected dishes. Each dish had 7 ingredients, but participants sometimes put fewer or more ingredients, so in total, this resulted in 2420 human actions. The state for this task was constructed using the following features: 1 feature for the type of dish the participant was currently making out of the possible three, 7 binary features representing whether each of the ingredients for the dish was included so far, and 14 features denoting the ingredient in each of the 14 locations in the kitchen. The action at each time step was getting one of the 14 ingredients and putting them in the main plate or a "serve" action, representing that the person had completed the dish. Errors were computed as follows: If the selected ingredient was part of the specified

dish and it had not yet been used, then the participant took an acceptable action. If an ingredient not in the dish was included or the human served a dish with missing ingredients, then the human made an error.

The blind spot vector of the human is modelled as a binary vector with the same size as the state, denoting whether the human can observe each state feature. Similar to the gridworld task, the observation is obtained using a mask of the true state and the blind spot vector. The implicit state $\mathbf{s}_h^i$ represents the human's estimate of the true state of the world and is mainly used to predict which ingredients the human assumes is in each of the 14 kitchen locations. The optimal policy and acceptable actions are computed based on the ground-truth ingredients in each dish. Because the state space is much larger for this domain, we use collapsed Gibbs sampling for inference, shown in Algorithm 1.

To evaluate our model, we compute an estimate of the ground truth based on our task setup. The ground truth blind spots are assumed to be the location of the salt and sugar, which are unobservable to the person. The human noise ground truth value is obtained by removing the salt/sugar errors and taking the percentage of remaining actions that are erroneous. While these are the aggregate ground truths we estimate, participants have different blind spots and varying levels of noise during execution, making it difficult to truly recover this blind spot vector. In the data, we observed that there was approximately 24% total error in the demonstration data, with 7.23% being attributed to blind spot (salt/sugar) mistakes and 16.78% resulting from other forms of noise. With so much noise in the data, it can be challenging to determine the true blind spot vector.

In Figure 4-7, we plot the performance of our model in predicting human blind spots as the budget of demonstration data is increased. We run the model for each participant individually, and the input is the first n datapoints in that participant's data (e.g., 60 datapoints on the x-axis indicates that the model uses the first 60 tuples from the demonstration data to infer human blind spots). The performance metric used is the accuracy of predicting the true blind spot vector.

The task of predicting the true blind spot vector is extremely challenging. Se-

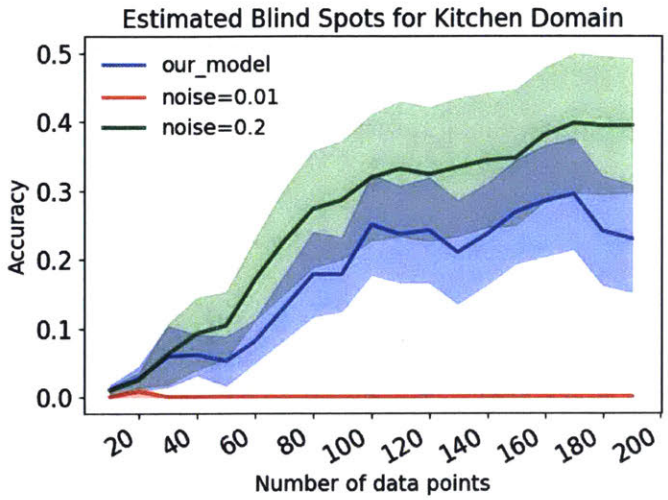


Figure 4-7: Performance on predicting human blind spots on the kitchen domain.

lecting a vector randomly at chance would result in an accuracy of 0.002 because the space of possible vectors is large, and it is challenging to predict the exact vector correctly. We first show that our model is able to get about 30% accuracy of predicting the ground truth $\mathbf{b}$, which is much better than random chance.

To further analyze the benefits of our model, we include a comparison of our model with one that has no explicit noise parameter that needs to be estimated. Instead, we set a constant value for the noise. In one condition, the noise is set to a minimal low value such as 1% noise. This results in very poor performance because the model consistently attributes noisy human actions to additional blind spots since it does not have any other way to explain the randomness in human actions. We also included a baseline where we set the model to have 20% noise, which is close to the true amount. This model is similar to an oracle that knows the true value of human noise. Our model is able to automatically infer both the blind spots and noise and achieves performance close to this oracle variant. In real-world applications, it is unlikely to know the true amount of human noise, so it is best to infer both variables automatically, as our model does. It also allows for the model to learn different noise values for different users.

Secondly, because our model includes other latent variables that are intended to

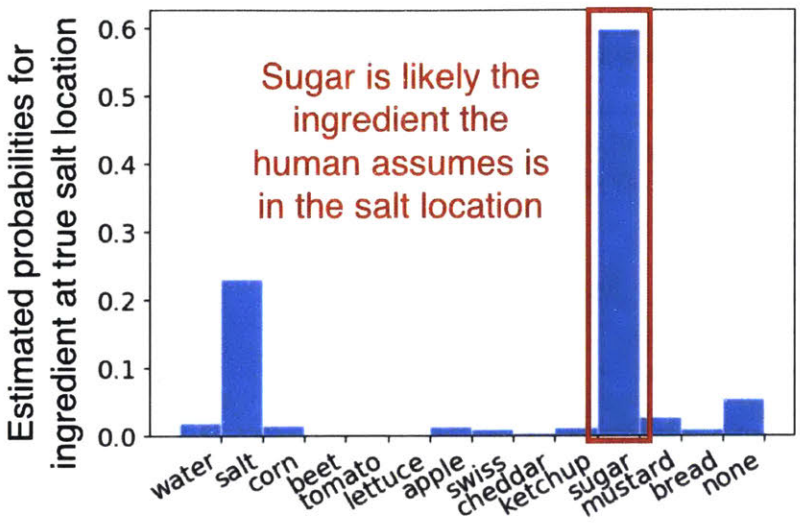


Figure 4-8: Estimate of participants' implicit state of the world on the kitchen domain.

explain human actions, we can query for other quantities, such as the most likely human's implicit state of the world $\mathbf{s}_h^i$. Similar to the gridworld task, this variable can be inferred by calculating $P(\mathbf{s}_h^i|D)$. Then, to get the probability of various values for a specific feature, marginalization can be used. Figure 4-8 plots a distribution over possible ingredients the human is assuming in the true salt location in the kitchen. The model predicts sugar as the most likely ingredient, which provides a tool for better understanding human errors and human assumptions leading to those errors.

Algorithm 1 Collapsed Gibbs Sampling

Given $D = \{\mathbf{s}^i_{real}, a^i_h, e^i\}$, $S = []$
for $(\mathbf{s}^i_{real}, a^i_h, e^i) \in D$ **do**
  $\mathbf{b} \sim P(\mathbf{b})$
  $\eta \sim P(\eta)$
  for $num \in [1, ..., N]$ **do**
    $\mathbf{b} \sim P(\mathbf{b}|D, \eta)$
    $\eta \sim P(\eta|D, \mathbf{b})$
    if $num > B \wedge e^i = 1$ **then**
      $S+ = (\mathbf{b}, \eta)$
    end if
  end for
end for
$P(\mathbf{b}, \eta) = \frac{n_{(\mathbf{b},\eta)}}{|D|}$
return $P(\mathbf{b}, \eta)$

4.5 Discussion

Results show that our approach can infer human blind spots and human noise in task execution on the two tested domains. However, we now discuss some limitations of the model when testing on other applications. First, the graphical model in Section 4.3 provides one way of inferring whether an error is caused by representation limitations or insufficient training. However, the model forces us to estimate the human's view of the world in terms of the state representation that the agent has. In some settings, it might be the case that explaining the human's behavior through this intermediate representation may not be appropriate. For example, if a person cannot see color, she might not be filling it in with some implicit color, but instead acting directly according to the flawed observation.

To model this, we present a variation of our graphical model that estimates the human policy directly with respect to the representation of the observation $\pi^o_h : \mathcal{O} \rightarrow \mathcal{A}$ without forcing an intermediate representation. The computational cost of doing inference with this model is much higher ($|\mathbf{b}| + N|\mathbf{o}_h| + N|\mathbf{s}_h|$ parameters for the original vs. $|\mathbf{b}| + N|\mathbf{o}_h| + |\pi^o_h|$ parameters for the variation where $|\pi^o_h|$ can be large). If we consider the space of all possible human policies however, it can be a rich model

to explore in future work.

Secondly, in both of these models, since there are many factors that affect action decisions, disentangling the different error sources can be challenging. For example, if there is a lot of human noise in demonstration data, it may be difficult to determine whether there is a systematic human blind spot with limited noise or no blind spot with very high noise in task execution. Because there are multiple possible explanations for the data, the problem can be ill-posed in certain applications. In those cases, it's important to include strong priors for different variables in the model to increase the chance of identifying the most likely explanation for human errors.

Next, the model depends highly on the representation of each of the variables. For example, in the kitchen domain, our model learns that people's perception of the salt and sugar in the kitchen is incorrect. Specifically, they mix up the location of the two ingredients because they look indistinguishable. To learn this result, the representation of the state, blind spot, and observation needed to include the location of the ingredients. Without this information, it would not have been possible to learn this particular blind spot. One possible extension of the model is to have the blind spot depend on the true state. For example, in tasks where the true state is an image, it may be better for the blind spot to be state-dependent, which would represent that the human cannot see a region of a particular image. Having the blind spot be independent of state would require making assumptions about the representation of the blind spot before looking at the data. Similar to this modification, other changes to the model can increase generalizability.

Depending on the domain, the model is flexible enough to make changes to conditional dependencies and observed variables. For example, in this work, the agent is able to observe whether the human's actions lead to an error in the task, but maybe the error is not directly observed and instead, the agent has access to a noisy observation of the error. Another possibility is that the error is only observed at the end of a sequence of actions. In this case, it would be important to consider a notion of time in the model. The model can accommodate these variations for improved flexibility across several application domains.

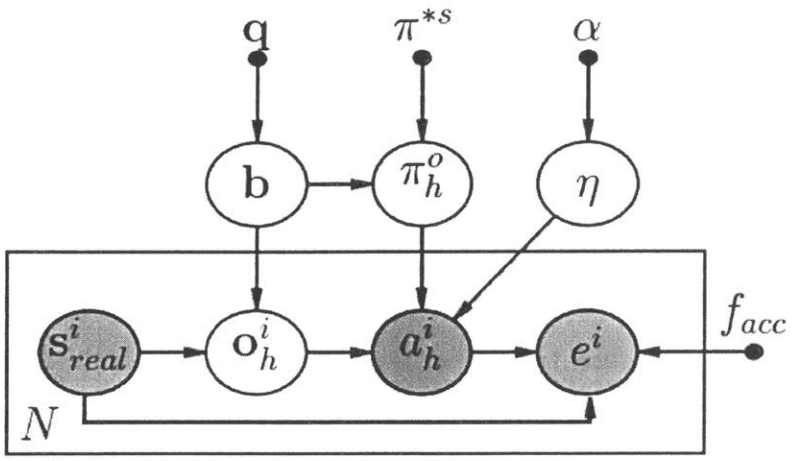


Figure 4-9: Variation of original model that directly estimates the human policy in terms of the human's observation.

4.6 Conclusion

In this work, a generative model is presented for identifying when human errors are caused by representational limitations vs. other policy inaccuracies. We assume that humans may not observe features that are important for safe decision-making. To explain human actions, our approach models and infers the human's estimate of the true state. Experiments on a gridworld domain with simulated human demonstrations and a kitchen task with real user data show that our approach is able to recover human blind spots as a cause for errors. In future work, we plan to augment the model to include additional causes for errors and apply the model to several real-world problems.

Chapter 5

Discovering Agent and Human Errors

5.1 Introduction

In the previous two problems, either the agent or the human had a limited representation of the world, and the proposed approaches identified errors with the use of the other's help. In this chapter, agents and humans both have full observability of the environment, but due to limited training, each focuses on a subset of factors to select actions. When transferring this knowledge to novel scenarios, the agent may ignore important factors that were never relevant in training. Our approach uses data from humans trained in different scenarios to identify situations in which each is suited to act.

Let's imagine a driving scenario, pictured in Figure 5-1. In this example, an agent is trained in highway settings where there are only cars and large trucks. The agent's simulation environment is quite accurate for regions that were modelled and allows an agent to learn a high-quality policy for a variety of highways. Imagine that we have access to demonstration data as well from human drivers, but these drivers mainly drove in neighborhood settings, where there are cars and special vehicles, such as ambulances and school buses. The real world consists of many complex scenarios that encompass these two subsets of environments and more. It is important for an agent to learn when to transfer control to a human who is better trained to act in that region.

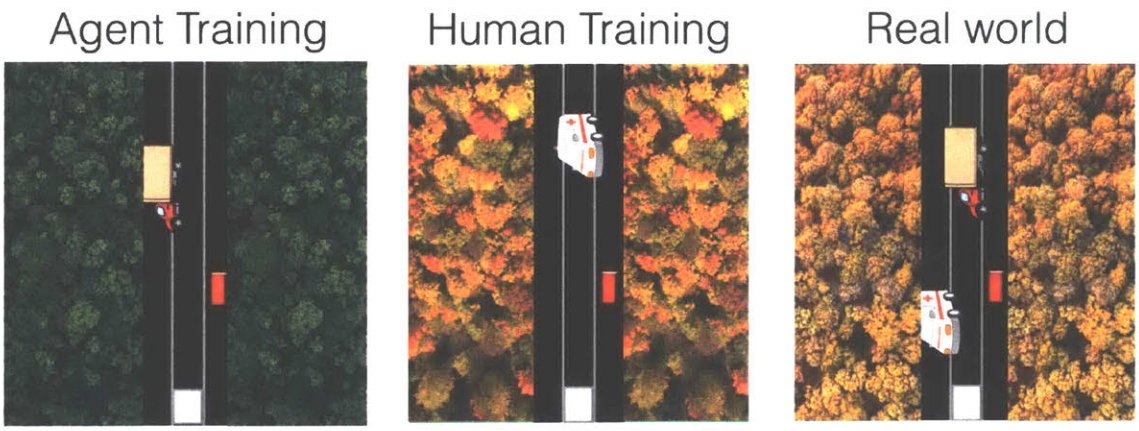


Figure 5-1: Driving scenario that motivates the problem of identifying agent and human errors. An agent is trained in highway settings with cars and trucks, while a human is trained in neighborhoods with cars and ambulances. In the real world, it may be necessary for an agent to transfer control in settings where it was not trained to act. However, comparing surface-level features, such as nature surrounding the roads, can be distracting, so it is important for the agent to identify the correct set of factors for transferring control effectively.

An important point to observe in Figure 5-1 is that based on surface-level features (e.g., the color of trees on the side of the road), the human's environment looks similar to the real world. However, if the current region of the real world is a highway, the agent is actually better suited to act. Thus, the agent needs to identify what factors are important for determining when to transfer control without relying on surface-level differences.

More concretely, we consider a setting in which agents are first trained in simulation environments [225, 138, 161] to better handle complex, real-world tasks. While simulators are providing increasingly realistic training environments [200, 63, 45] that can help accelerate initial learning, there is almost always a gap between simulation and reality [181]. Directly transferring learned policies to the open world in these cases can cause costly systematic errors that occur due to differences between the two environments. Identifying errors before deployment in the real world is crucial for safe execution, yet it is challenging since the simulator lacks signals of such failures.

We focus on errors that occur due to inattentional blindness when going from a simulator to the real world. This means that some important factors for performing

the task are ignored and lead to errors. We assume that agents can perceive numerous features but learn to ignore many of them due to the incompleteness of the simulator to reflect the complexity of the real world. For instance, in the previous driving example, the simulator might model general driving conditions, with different vehicles, such as cars and trucks, and the agent learns to drive well in this setting. To learn, the agent focuses on the most important features for decision making, such as distance to nearby vehicles and which lane it is in. Because the agent trains extensively in this simulator and has some sensor capabilities that are more accurate than human perception, it acts optimally in this world, much better than a human can act. However, the agent never encounters some aspects of the real-world (e.g., emergency vehicles) so the agent learns to ignore these aspects that did not matter in simulation. For example, the agent knows how to avoid cars and be cautious around trucks, but does not recognize ambulances as special types of vehicles that require different behavior. Because correct behavior around emergency vehicles is different than behavior around other vehicles, executing the learned policy in the real-world may cause costly mistakes.

Directly executing the agent policy in the real world poses safety concerns, so we propose complementing the agent with a *black-box* helper agent. In contrast to the problem discussed in Chapter 2, we do not assume that the helper agent knows how to act properly in all regions of the real world. In fact, we assume that the helper agent can also make errors and can be noisy during task execution. While this secondary agent could be any autonomous agent, our assumptions about the helper agent being a black-box agent with suboptimal, yet complementary knowledge, are inspired by having a human in this role. Therefore, throughout this paper, we will refer to the helper agent as the human.

The only input we assume from the human is limited demonstration data collected from "demo world", a constrained version of the real world in which the human is comfortable acting. We assume that the human behaves in settings in which she is trained well in, which is complementary to the areas where the agent acts well. In the driving example, the agent is trained in an environment with many types of

vehicles but no ambulances. A human would recognize ambulances as important for generating safe plans, but may not be comfortable driving on a highway with large trucks and vans. The demonstration data can thus be used to learn agent errors, while comparison of data from the simulator and demonstrations can provide signals of human errors. The real world can contain both of these scenarios. Figure 5-2 depicts the setup of the different worlds.

We formulate the problem of identifying agent and human errors in this setting as a two-step problem. First, the agent learns important features missed during training in simulation by using human demonstrations. Second, it learns separate predictors for agent and human errors to allow for *safe joint execution* by handing off control to the most capable agent. The first step requires learning a minimal feature representation that captures the important aspects from the simulator as well as from the real world. This representation learning is crucial because failing to learn important features in the real world can result in the agent being unable to identify important error regions, leading to costly mistakes.

Once the agent has a minimal and accurate feature representation, the agent then learns supervised error predictor models for itself and the human based on simulator and demonstration data respectively. We assess the utility and success of our learned error models by analyzing the effectiveness of a safety-oriented hand-off policy, which leverages the models to decide who should act in the real world, preventing therefore dangerous catastrophes. Our meta-learning approach for modeling errors is motivated by the fact that neither the agent nor human has an understanding of each other's flaws. Thus, the human may fail to recognize the incompleteness of the simulator. We address this gap in human and agent reasoning using a meta-model that can jointly learn about the errors of both.

In our experiments, we evaluate the end-to-end approach over two different domains with distinct agent and human error regions. We show that our framework learns important features and areas of high error that help avoid costly mistakes in the real world by appropriately transferring control to the safest agent.

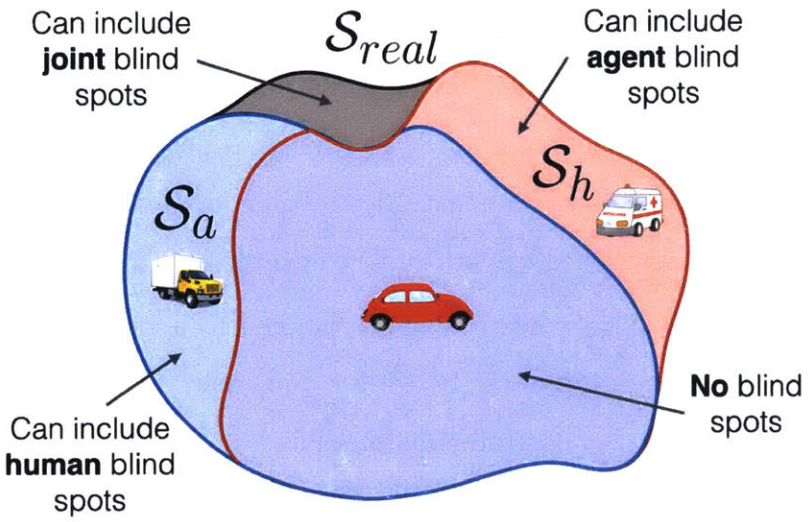


Figure 5-2: The agent is trained in a simulator world, which sees a subset $\mathcal{S}_{sim}$ of the true state space $\mathcal{S}_{real}$. Human demonstrations operate in subset $\mathcal{S}_{demo}$, and joint execution happens in the real world space $\mathcal{S}_{real}$.

5.2 Related Work

In this section, we outline prior literature related to the discovery of agent and human errors due to limited training and inattentional blindness. First, we discuss imitation learning, which refers to an agent learning from expert demonstrations. This relates to our problem as an agent has access to demonstrations from a human, but the human can also make systematic errors. Next, we outline prior work on novelty and outlier detection, which can be useful techniques for determining when regions in the real world look different from the agent's training environment. We include related literature in multi-agent teaming and ad-hoc teamwork that focuses on teams cooperating and working together towards a shared goal. Finally, we present prior work on safety in sim-to-real transfer, which relates to our goal of modelling errors for safe real-world execution.

5.2.1 Imitation Learning

Imitation learning [94, 189] is commonly used to learn a task given expert demonstrations. Inverse reinforcement learning [252, 11, 154, 1, 254, 65] is one class of these methods that can be used to estimate a policy given demonstrations by recovering

the underlying reward function that guides the expert's actions. These approaches work under the assumption that the expert is optimizing a reward function of an unknown MDP. This assumption does not apply to our helper agent as it is modeled by characteristics of humans. Thus, we instead use behavior cloning methods [227, 249] for estimating the policy of the helper agent, which directly map states to actions.

Many imitation learning works have been applied to a variety of domains, such as autonomous driving and robotics. In driving domains, to handle execution at test time, work by Codevilla et al introduce a modification that allows a trained model to respond to navigational commands [53]. This enables a more robust system that can use external commands to adapt in the real world. In another work [247], in the application of robotics, models are generalized to new tasks by first building a knowledge base from human and robot demonstration data. Using this prior knowledge and a single new demonstration, the robot can perform novel variations of the task. In related work with a similar motivation to ours [253], the authors show that combining reinforcement learning with imitation learning outperforms either one alone. Expert demonstrations are used primarily to deal with the challenge of exploration in complex domains, whereas in our work, human data is used to identify agent blind spots.

Many imitation learning works assume that the perspective of the demonstrator and the learner are identical. In work by Stadie et al [210], an agent learns by observing an expert provide third-person demonstrations, inferring the task, and learning to act from its own perspective. Similarly, in another work [132], robots are trained to learn by "imitation-from-observation", which requires imitating a behavior while taking into account changes in perspective, differences in object position, etc. These works provide methods for using demonstration data from various sources and viewpoints and can be used along with our approach to identify agent and human blind spots.

The IRL approach proposed by [126] is relevant to our work in that it jointly learns a feature representation and a reward function. The approach learns the best features to explain the data, by constructing new features that are logical conjunctions of basic

features and by pruning out irrelevant features. This work is complementary to ours, as it can be used to augment our feature set with high-level features constructed from the available set.

5.2.2 Novelty/Outlier Detection

Methods for outlier and novelty detection [43, 169, 42] are relevant to our work as they can be used to determine which instances in the real world are different from the agent's training environment and would thus require transfer of control to a human. Outlier detection identifies datapoints that differ significantly to most examples in a dataset. Discovering when instances are too different from the "normal" set of examples can be challenging. Some prior works [204, 81] convert anomaly detection into a semi-supervised learning problem. For example, in one work [204], unlabelled examples are combined with a small number of normal examples, which are labelled by a human. The provided labels can significantly improve learning as they guide the model to better learn the right set of anomalies.

In related work [244], the authors propose an approach for identifying anomalous events in video scenes. Instead of using manually designed features as many prior works do, the authors present a framework to learn discriminative features based on appearance and motion. The high-level idea is related to ours as we also want to determine instances that differ significantly from simulation such that an agent should transfer control. However, our problem setup is different as we assume access to a training environment for the agent, and our goal is to transfer control to a human, who may also make errors.

Since many of these methods focus on identifying rare instances, the discovery is often solely guided by the estimated distance of a given instance to the training distribution. Such distance metrics become uninformative when agents have access to many features that include distractor and redundant features. Instead, we focus on identifying blind spot regions that can be characterized by discoverable missing feature patterns and agent-human action mismatches.

5.2.3 Multi-Agent Teaming

Prior literature has proposed approaches for successful joint execution in teams of multiple agents or human-agent teams. Wray et al propose an approach for managing hand-off decisions between an agent and a human when models of both are perfectly known [242]. In another work [17], a demonstrator's global viewpoint of the world can differ from an agent's local viewpoint. This setup shares some similarities to our work in that we consider a setting in which agents and humans use different sets of features to act in the world based on their own training environments. The authors develop a method that identifies when this difference is large, guiding the agent's action decisions.

Human-robot team training techniques are used to train humans and robots to generalize to novel tasks [159, 186]. These methods train teams under different roles of a task and task variations to enable more robustness to unexpected scenarios. In our problem, each teammate is assumed to have been trained in a limited setting, and the model learns how to transfer control effectively when both can make errors.

Ad-hoc teamwork is a related area within multi-agent teaming as it focuses on agents coordinating with many teammates of known or unknown types on the fly to achieve a shared goal [211]. The problems in this space include learning how to model teammates, learning how to communicate with them, and achieving robustness and adaptation to diverse partners [20, 22, 21]. Our problem shares similarities with this literature in that the agent is tasked with collaborating with another agent of unknown characteristics, and they cooperate towards a common goal. Our work is complementary, as it leverages joint execution to address the blind spots of cooperating agents.

5.2.4 Safety and Transfer

Safe reinforcement learning is an active research area [213, 73, 110]. Many works [107, 71, 25, 232] propose theoretical safety methods that guarantee safe acting in reinforcement learning settings. In work by Junges et al, the authors first construct

a set of permissive strategies, which are then used to ensure safe exploration during task execution. Another work [91] also proposes an approach that explicitly encodes safety by using 1) a safety function representing the safety of taking an action in a state and 2) a backup policy that enables the agent to return to safety when in a dangerous state. An alternative method to represent safety is through a temporal logic specification [4]. A "shield" is then used to correct agent actions when a specification is violated.

Another work focuses on cautious exploration for a task of robot navigation with collision avoidance [110]. To ensure no collisions, the model estimates the agent's uncertainty about the world and the probability of a collision, which enables more safe behavior in the real world. In work by Eysenbach et al [67], a forward policy is used to predict when an agent is likely to move into a non-reversible state so that the agent can then use a reset policy to return to the initial state. While these works provide approaches for training safer and more cautious agents, they do not address the scenario where the agent selectively attends to a limited set of features that lead to costly failures in the real world.

Transfer learning [220, 23, 51] is a more general technique that applies knowledge learned in one domain to a new one, which can be simulation to real-world or across different domains. While representations may differ between domains, and agents may need to learn a mapping, it is assumed that knowledge of the agent generalizes to the new domain. We study adapting two agents with blind spots to a new domain through joint execution.

5.3 Problem Statement

We first discuss the overall formulation of our problem and then outline two subproblems that we aim to solve.

5.3.1 Overall Formulation

We define a real-world environment $M_{real} = \{\mathcal{S}_{real}, \mathcal{A}, T, R\}$ as an MDP with state space $\mathcal{S}_{real}$, action space $\mathcal{A}$, transition function T, and reward function R. Each state $s_{real} \in \mathcal{S}_{real}$ can be mapped to a set of features $\Phi : \mathcal{S}_{real} \rightarrow F$, where $F = \{f_1, f_2, ..., f_n\}$ is a set of random feature variables. The function Φ maps a state to a set of feature values and is defined according to the domain. The agent is trained in a simulation environment $M_{sim} = \{\mathcal{S}_{sim}, \mathcal{A}, T, R\}$, which is an imperfect model of the real world in that the state space is a subset of the true state space $\mathcal{S}_{sim} \subset \mathcal{S}_{real}$. Because of this limited world, the agent only focuses on a subset of the full feature set $F_a \subset F$. For example, the color of trees or buildings may not have mattered for acting in the simulation world, so the agent ignores these distracting features. Most importantly, because the simulation did not model some parts of the world, the agent ignores other important features that are needed for safe acting in the real world, such as uncommon traffic lights or ambulances.

The agent trains in the simulation environment using Q-learning and learns a policy $\Pi_{sim} : \Phi(\mathcal{S}_{real}) \rightarrow \mathcal{A}$, which maps state features to actions. In this training process, the agent collects data from all of the state-action pairs it visited in simulation $D_{sim} = \{(s_i, a_i) | i = 1, .., n\} \sim M_{sim}$. We assume that π_{sim} is optimal with respect to the states seen in simulation $s \in \mathcal{S}_{sim}$ and that agent errors in the real world are solely due to features missing from F_a.

We define blind spots as regions in $\mathcal{S}_{real}$ for which the agent or the human make systematic errors in the real world that cause high negative reward. The agent blind spot region $BS_a \subset \mathcal{S}_{real}$ is a subset of the real-world state space where there are clusters of errors caused by missing features from imperfect simulator modeling. For each state in the agent blind spot region $s \in BS_a$, the following equation holds:

$$Q^{\pi^*}(s, \pi^*(\phi(s))) - Q^{\pi^*}(s, \pi_{sim}(\phi(s))) > \epsilon \tag{5.1}$$

where Q^{π^*} is the optimal Q-value function in the real-world and π^* is the optimal policy obtained from this value function. We do not know either of these, which

makes estimating blind spots in the real world challenging.

We assume access to human demonstrations in the form of state-action pairs $D_{demo} = \{(s_i, a_i) | i = 1, .., n\}$. These demonstrations are collected from a demo environment $D_{demo} \sim M_{demo}$, where $M_{demo} = \{\mathcal{S}_{demo}, \mathcal{A}, T, R\}$ is an MDP with a state space that is a subset of the real world $\mathcal{S}_{demo} \subset \mathcal{S}_{real}$. The human may make errors and act suboptimally when acting in this world. We do not know what features the human is using to act, but the agent observes the demonstrations through its feature set F. We assume that the human actions can be approximately explained by a subset of the agent feature set $F_h \subseteq F$. The agent's estimate of the human representation is $\hat{F}_h$.

Similar to agent blind spot regions, we define a region in the real-world state space $BS_h \subset \mathcal{S}_{real}$ that denotes human blind spots. For each state in this region $s \in BS_h$, the following equation is satisfied.

$$Q^{\pi^*}(s, \pi^*(\phi(s))) - Q^{\pi^*}(s, \pi_h(\phi(s))) > \epsilon \tag{5.2}$$

Since the human is a black-box helper agent, we do not have access to π_h, so we cannot estimate the true set of human blind spots.

In terms of features, the set learned from simulation F_a is limited due to a limited $\mathcal{S}_{sim}$. The human feature set F_h is also limited because of a limited state space $\mathcal{S}_{demo}$ in the demonstration world. The true feature space that includes all important features needed to take safe actions in the real world is defined as $F^* \subset F$, where $F^* \subset (F_a \cup F_h)$. We do not have access to the true optimal feature space so we estimate it in our approach $\hat{F}^* \subset (F_a \cup \hat{F}_h)$. We assume F will include some features that correlate with blind-spot regions. The feature set does not need to be complete with respect to the "true" representation of the real world, but the stronger the correlations of features in F are to agent and human blind spots, the better the blind spot models will be.

5.3.2 Breakdown into Subproblems

Because agent errors occur due to ignored features in simulation, we first need to identify important features needed for safe execution in the world. These features can then inform our predictions of where blind spots are likely to occur. Thus, we separate the problem of learning blind spots into two subproblems:

1. Learn an estimate $\hat{F}^*$ of the true feature representation that captures important features from both F_a and $\hat{F}_h$ needed for taking safe actions in M_{real}.
2. Learn estimated blind spot models $\Theta_a : \hat{F}^* \rightarrow p_a$ and $\Theta_h : \hat{F}^* \rightarrow p_h$, where $p_a, p_h \in [0, 1]$, to predict agent and human blind spots, respectively, for safe transfer of control in the real world.

The reason we do not use the full set of features F is because there are many distracting features that differ between the simulation and real-world environments that are not informative for acting in the real-world. Including such features into blind-spot modeling would lead to incorrectly labeling regions as blind spots. In the initial example (Figure 5-1), using the color of the trees surrounding the roads can be a distracting surface-level difference that can result in the agent incorrectly transferring control at the wrong regions.

As shown in Figure 5-3, the inputs to our problem are: the simulator policy π_{sim}, the feature set F, data from simulation training D_{sim}, and human demonstration data D_{demo}. We treat the agent policy π_{sim} as a black box to minimize assumptions about how the agent is trained. The first output is an estimate of the important set of features needed to act safely in the world $\hat{F}^* = F_a \cup (\hat{F}_m \subseteq \hat{F}_h)$, which includes features from the simulator world F_a as well as a subset $\hat{F}_m$ of features the agent is missing based on the human's estimated feature representation $\hat{F}_h$. Learning these missing features is crucial as they indicate blind spots of the agent. The second output is an estimated blind spot model for the agent $\Theta_a : \hat{F}^* \rightarrow p_a$ and one for the human $\Theta_h : \hat{F}^* \rightarrow p_h$ under the compressed feature representation $\hat{F}^*$.

In summary, these are the following characteristics of our problem that inform our approach for learning blind spots:

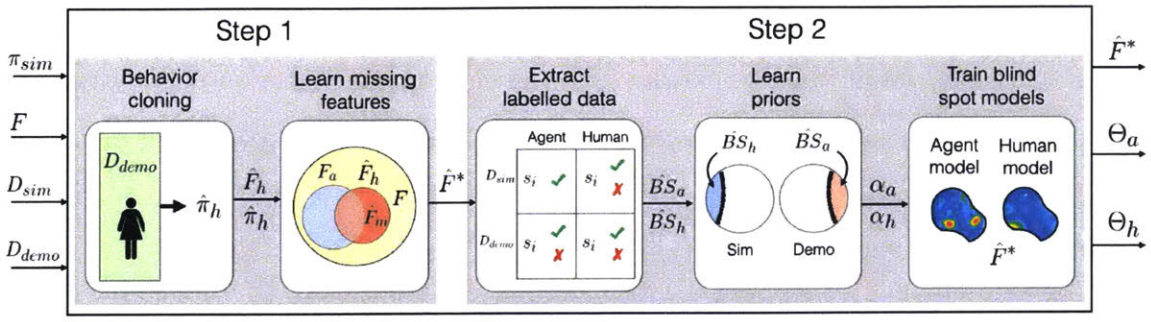


Figure 5-3: Pipeline of approach. The first step involves learning missing features that the agent learned to ignore in simulation. The second step is learning agent and human blind spot models in the real world, which involves extracting labeled data, learning priors of blind spots, and finally training the supervised learning models.

- Agent blind spots are due to an imperfectly modelled training environment that causes inattentional blindness. In other words, the agent learns to ignore some important factors due to limitations in the simulator.
- The human is a black box. Insights about the human model can be extracted from demonstration data collected from the demo world. Humans have complementary abilities to agents but can make errors and act suboptimally.
- Agent blind spots can be learned by observing data from demonstrations. Identifying features that the human is using for decision making in the demo world tells us about features the agent is missing. The missing features along with action mismatches signal agent blind spots.
- Human blind spots can be learned from simulation data. Since the agent has extensive experience in this world, it can compute differences in utility values of the agent's optimal action in simulation and the human's predicted action in that world as indicators of human errors.

5.4 Approach

We now present a two-step approach for discovering agent and human blind spots, as shown in Figure 5-3. The first step uses behavior cloning and feature selection techniques to identify the set of features that the agent learned to ignore in simulation.

The second step involves learning blind-spot models for the agent and human to safely transfer control. To do this, we extract noisy labels from simulator and demonstration data in the form of action mismatches and then train supervised learning models to predict blind spots.

5.4.1 Step 1: Identifying Missing Features

The goal of this step is to learn an augmented representation $\hat{F}^* = F_a \cup (\hat{F}_m \subseteq \hat{F}_h)$, which includes important features that the agent learned to ignore in the simulation environment but are important for safe acting in the real world.

We have multiple challenges to this end:

- *Insufficient features*: The agent's learned feature representation from the simulator F_a is not sufficient to represent the true world, so we need to identify missing features that are important to act in M_{demo} using demonstration data.
- *Distractor features*: Some features in the demo world are different from those in the simulation world, but they do not always indicate blind spots. Some of them might distract the agent because they indicate that these regions are different from the training environment while not affecting the policy. Thus, outlier detection techniques that identify which states differ from simulation would have low precision to identify blind spots.
- *Redundant features*: Features included into $\hat{F}_m$ are used as signals for identifying agent blind spots. Since F may include multiple features that are highly correlated, a feature should be included into $\hat{F}_m$ only if it is adding information for predicting human behavior in addition to the features in $\hat{F}_a$. Therefore, the construction of $\hat{F}_m$ should be conditional on $\hat{F}_a$.

To learn $\hat{F}^*$, we first use behavior cloning (BC) [194, 236] to learn an estimate of the human's representation and policy. This involves learning a state-action mapping from the demonstration data D_{demo}. We used a logistic regression model and selected features with non-zero weights to obtain an estimate of the features the human is

using $\hat{F}_h$. This step provides us two outputs: an estimate of the human policy $\hat{\pi}_h$ and an estimate of the human's representation $\hat{F}_h$. Both are noisy estimates because demonstration data is limited.

BC addresses two challenges: First, it learns features that are informative for the human policy, $\hat{F}_h$, by identifying which features correlate with actions that the human takes. Second, it eliminates distractor features that differ between simulation and real-world but are not important for acting by only selecting features that contribute to human decisions.

Since BC estimates $\hat{F}_h$ independently of F_a, $\hat{F}_h$ may include parallel redundant features. We only want to add missing features that are not already explained by features in F_a because we will then use these features to provide blind spot labels. For example, imagine that the agent already has features for detecting the distance of a car in front of it using a proximity sensor. We estimate that the human is using the feature of a car detector from camera data to obtain similar information. The agent ignored the camera data as it was less informative. These are two different features but correlated so the agent should not add this extra feature and consequently should not associate this part of the state space with agent blind spots.

We eliminate redundant features through a greedy feature elimination step. We start from the set of all possible relevant features $\{F_a \cup \hat{F}_h\}$ and iteratively eliminate features that do not decrease the accuracy of predicting the human demonstration data. The process is as follows: we first compute the initial accuracy of predicting the human actions in D_{demo} using $\{F_a \cup \hat{F}_h\}$. We then remove features one-at-a-time from $\{\hat{F}_h \setminus F_a\}$ and retrain the model to predict the demonstration data. If the best accuracy is not worse than the initial accuracy with all joint features, we remove the feature from the representation as it is not contributing to positive performance of the model. If many features, when removed, result in identical scores, we randomly choose one to remove. We continue with another iteration through all features and remove features individually, unless all features in the remaining set result in an accuracy drop. This gives us a final set of missing features $\hat{F}_m \subseteq \hat{F}_h$ that are not expressed in F_a. The final output is a succinct representation $\hat{F}^* = F_a \cup (\hat{F}_m \subseteq \hat{F}_h)$, which only

includes features that add information over F_a for predicting D_{demo}.

The greedy feature selection process is one way to remove correlated features. While there can be interactions among features that could cause greedy elimination to incorrectly remove individual features when groups of features matter, we observed through experiments that our approach was able to keep enough features to obtain high performance.

5.4.2 Step 2: Modeling Agent and Human Blind Spots

Our setup does not allow for computing ground-truth labels for blind spots since reward signals or Q-values are unaccessible for demonstrations and the real-world environment. We instead use action mismatches between predicted human behavior and the agent's policy as noisy labels of blind spots. These extracted labels are noisy because mismatches are not true indicators of where blind spots may exist. Our agent and human blind spot models learn to identify potential error regions using this information by identifying patterns (state subspaces) that correlate with action mismatches.

Extracting Human Blind Spot Labels from Simulation

The main insight for label extraction from simulation data is that the agent acts well for states modelled in simulation and any significant divergence of human behavior on these states indicates a human blind spot. For the human blind spot model, the labels are estimated as follows: For each $s \in D_{sim}$, we estimate the action the human will take $\hat{a}_h = \hat{\pi}_h(\phi(s))$ based on the estimated human policy learned from the first step.

To estimate how well $\hat{a}_h$ would perform in this state, the agent can use its learned Q-value function from the simulator. If the value of the human's action is much worse than the agent's action, as defined by some threshold, this state is assigned a blind spot label for the human (Equations 5.3 and 5.4). Otherwise, the human gets a safe label for this state. The top right quadrant in Figure 5-4 illustrates the human blind

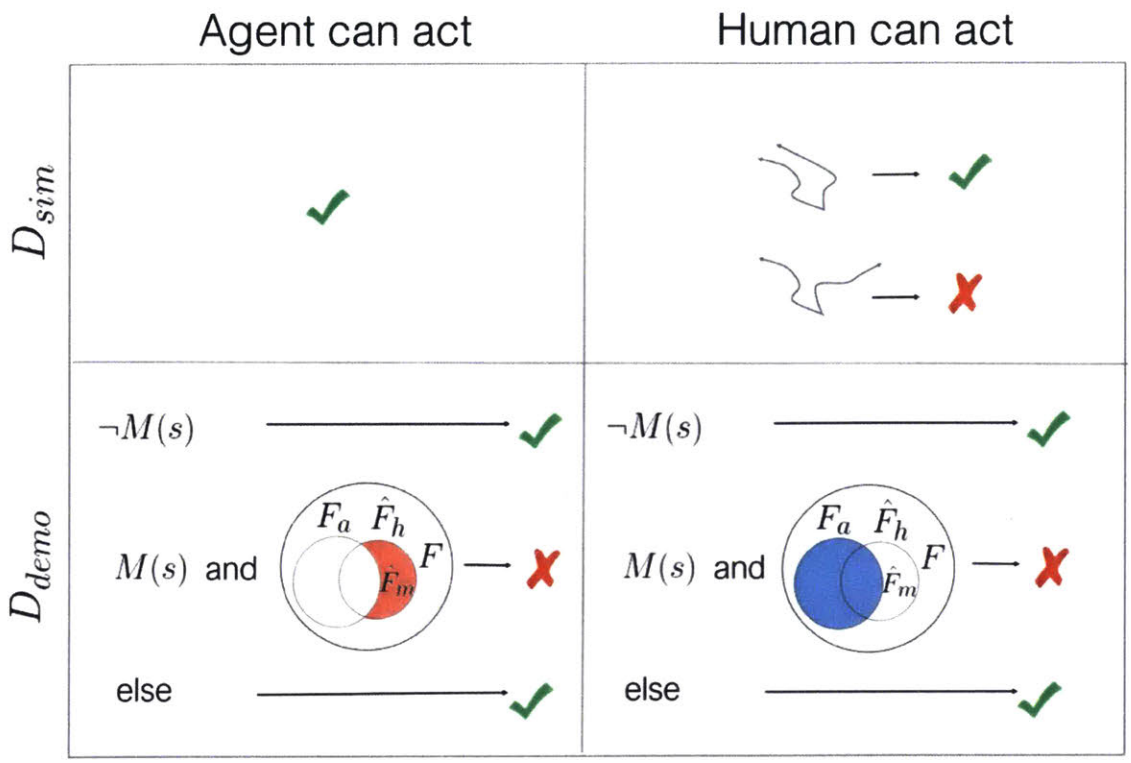


Figure 5-4: Data labeling to learn agent and human blind spots. Labels are extracted from simulation and demonstration data.

spot estimation in simulation states.

$$\Delta Q(s) = Q^{\pi_{sim}}(s, \pi_{sim}(\phi(s))) - Q^{\pi_{sim}}(s, \hat{\pi}_h(\phi(s))) \tag{5.3}$$

$$\hat{BS}_h = \{s \in D_{sim} | \Delta Q(s) > \epsilon\} \tag{5.4}$$

Each $s \in D_{sim}$ is assigned a safe label for the agent.

Extracting Agent Blind Spot Labels from Demonstrations

Extracting labels for demonstration data is more challenging. The agent is no longer assumed to act well since the agent can have blind spots outside of the simulation environment. Further, humans can be suboptimal and make errors in demonstration data as well. Thus, deviations in human and agent behavior do not necessarily indicate that the agent has a blind spot. When actions match, both agents have similar behavior and either one can act. When actions mismatch, this is a noisy indicator of

where one agent may be better suited to act than the other.

To estimate agent blind spots for states with action mismatches, we leverage the learned missing features $\hat{F}_m$ from our first step. We say that a feature $f_i \in \hat{F}_m$ is "activated" in a given state s, if $f_i = 1$ in s. Only a subset of features will be activated at any given state. If agent and human actions deviate and there is an activated feature from $\hat{F}_m$, this is an indication that the human is using a feature that the agent ignored, which is important for acting in the world. In this case, we assign a blind spot label to the agent. If actions do not match but the state does not have an activated feature from $\hat{F}_m$, there is no reason to think that the agent is acting suboptimally. Thus, this is an indication that the mismatch may be due to human random errors. In this case, we assign a safe label to the agent.

We define a function M (Equation 5.5) that indicates whether agent and human actions mismatch given a state. We then estimate agent blind spots using the mismatches and missing features $\hat{F}_m$ learned from step 1 (Equation 5.6).

$$M(s) = \begin{cases} \text{True} & \text{if } \pi_{sim}(\phi(s)) \neq \hat{\pi}_h(\phi(s)) \\ \text{False} & \text{otherwise} \end{cases} \tag{5.5}$$

$$\hat{BS}_a = \{s \in D_{demo} | M(s) \text{ and } \phi(s) \cap \hat{F}_m \neq \emptyset\} \tag{5.6}$$

Extracting Human Blind Spot Labels from Demonstrations

We assume that humans can make errors and act suboptimally in demonstration data as well. Disagreements with the optimal agent could indicate either human blind spots (systematic mistakes) or random errors. Assigning a disagreement label informs our models in two ways: If models can associate such disagreements with available features, the model identifies a blind spot. If not, disagreements inform the model about random human errors and guide blind spot models to prefer agent acting over human acting when no blind spot is detected for either. When blind spot models are oblivious to human suboptimality, hand-off decisions may be misguided. Imagine that the human is 20% suboptimal (randomly deviates from the optimal policy in

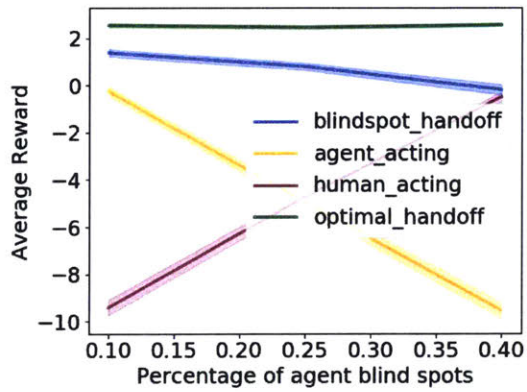
20% of demonstrations) in regions without blind spots, while the agent is optimal in this region. Blind spot models that do not know about these human errors would give equal chance of control to the human and the agent, leading to degraded performance. Thus, if agent and human actions deviate at a state, and the state does not have an activated feature from $\hat{F}_m$, we consider this mismatch as a random human error and assign a human blind spot label for this state.

Prior Estimation

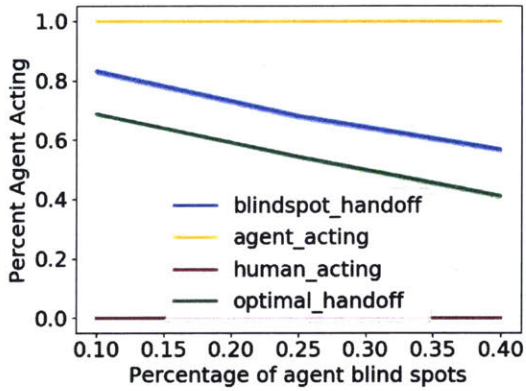
We have two data sources from which we are extracting blind spot labels. Demonstration data is limited in size, while simulation data is easily available in large amounts. These data sets differ in their representation of agent and human blind spot labels. However, these data proportions do not necessarily reflect the real-world proportions of blind spots. Simply training a supervised learner with both datasets will result in an imbalance in blind spot predictions (i.e., significantly more human blind spots as there is more simulation data available). Getting informative blind spot predictions from these models depends on estimating informative priors, guiding models about the likelihood of agent and human blind spots.

In our problem, the simulation world is more likely to contain human blind spots, and the demo world is more likely to contain agent blind spots. We assume that the relative proportions of human and agent blind spots respectively in D_{sim} and D_{demo} are representative of the proportions of these blind spots in the real world. This means that although there might be imbalance between D_{sim} and D_{demo}, the frequency of blind spot regions vs. safe regions within each environment is realistic in isolation. We use this insight to estimate the prior of agent blind spots α_a, human blind spots α_h, and no blind spots (i.e., both agent and human can act) α_n in the real world by solving the following equations, which provides a unique solution. The term α_n represents the unknown proportion of the real world in which both the agent and human can act. Estimating priors allows us to prioritize demonstration data appropriately to better learn blind spots.

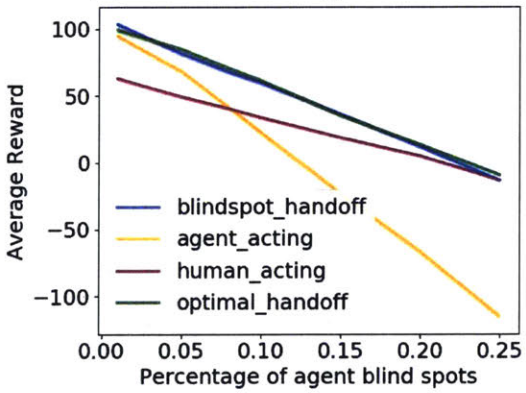
Equation 5.7 computes the proportion of blind spots in simulation and demon-



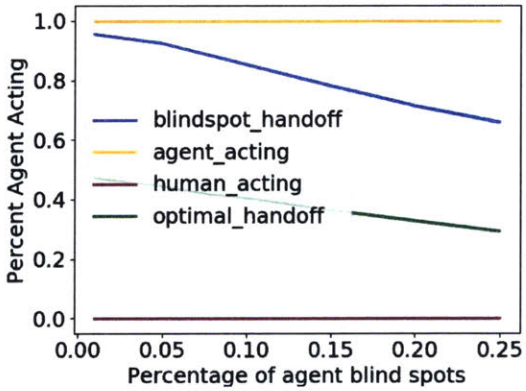
(a) Catcher: Average reward



(b) Catcher: Percent agent acting



(c) Driving: Average reward



(d) Driving: Percent agent acting

Figure 5-5: Performance of approach with varying agent blind spots in both domains.

stration data respectively based on the extracted data labels. Equation 5.8 relies on the insight that the simulation world contains mainly safe regions (both can act) and human blind spots, and similarly, the demo world contains safe regions and agent blind spots. Equation 5.9 states that the full real-world state space consists of regions with no blind spots, agent blind spots, and human blind spots. We do not handle the case where there are joint blind spots. Values α_h^{sim} and α_a^{demo} are directly computed from data as in Equation 5.7, while the actual priors α_h and α_a are recovered based on Equations 5.8 and 5.9.

$$\alpha_h^{sim} = \frac{|\hat{BS}_h|}{|s \in D_{sim}|} \ , \ \alpha_a^{demo} = \frac{|\hat{BS}_a|}{|s \in D_{demo}|} \tag{5.7}$$

$$\alpha_h^{sim} = \frac{\alpha_h}{\alpha_h + \alpha_n} \ , \ \alpha_a^{demo} = \frac{\alpha_a}{\alpha_a + \alpha_n} \tag{5.8}$$

$$\alpha_a + \alpha_h + \alpha_n = 1 \tag{5.9}$$

Training Blind Spot Models

Given the labels extracted from simulation and demonstration data as well as the learned priors, we train supervised learning models to predict agent and human blind spots in the real world. We oversample the extracted labeled data in a way that the sampled data matches the learned priors. We then train two random forest classifiers, one for predicting agent blind spots and one for predicting human blind spots. We perform cross-validation over various hyperparameters and choose the one with the best average F1-score. We train the final model on the entire training data and output a blind spot model for the agent $\Theta_a : \hat{F}^* \rightarrow p_a$ and one for the human $\Theta_h : \hat{F}^* \rightarrow p_h$, where $p_a, p_h \in [0, 1]$ are probabilities of a state being a blind spot. While we learn with batch simulator and demonstration data, similar concepts can be applied to already established systems that can improve with real-world data. This allows for iterative and incremental deployments. Here, we focus on developing a framework for safe execution in the real world *before* acting in the true environment.

Transfer of Control in the Real World through Blind Spot Models

We use the learned agent and human blind spot models to do safe transfer of control in the real-world environment M_{real}. At each state $s_{real} \in \mathcal{S}_{real}$, we query our learned agent blind spot model $p_a = \Theta_a(\phi(s_{real}))$ and human blind spot model $p_h = \Theta_h(\phi(s_{real}))$. If $p_a \leq p_h$, the agent acts. Otherwise, the human acts. Note that there are many ways to do transfer of control given these models (e.g., give preference to the agent if it can act, minimize the number of hand-offs). We use a simple hand-off scheme to evaluate how well our models capture failure regions and can avoid them through joint execution.

5.5 Experiments

5.5.1 Domains

Our approach is evaluated on two domains. The first is a modification of the game Catcher where an agent needs to catch certain types of fruits and avoid others. The agent and a simulated human are each trained in limited settings and need to work together in the real world environment. The second environment is a driving domain based on the example presented in Figure 5-1. An agent is trained in highways with cars and trucks, while a simulated human is trained in local neighborhoods with cars and ambulances. The team has to share control for successful joint execution in the real world. We now describe the two domains in more detail.

Catcher

The first domain is a variation of the game Catcher, in which an agent needs to catch falling fruits [181]. There are two features that describe the fruits: size and color. The feature space has 19 features for each value of the attributes, corresponding to the location of the agent with respect to the fruit (-9 to +9). The encoding is one-hot for each attribute-value (e.g., if there is a red fruit 4 units away, the red-color-+4 feature would be turned on while red-color-i $\forall i \in [-9, +9] \backslash (+4) = 0$). There are 20

additional distractor texture features that differ between simulation and real world but do not affect the policy. The actions the agent can take are moving left, moving right, or staying. The agent sees red and blue fruits with the same size in simulation that causes it to ignore size. The human only sees blue fruits that are either large or small. The agent needs to learn the missing and important size features while disregarding unnecessary texture features. In the true real-world representation, human blind spots are red, small fruits, and agent blind spots are blue, large fruits. The agent and human policies are obtained using linear Q-learning on the simulation and demo worlds, respectively.

Driving

The second domain is based on simulated highway driving [1, 214, 126]. The agent must navigate a three-lane highway while avoiding cars and trucks. The simulator, however, does not model ambulances, which exist in the real world. The feature space includes the following one-hot encoded features: 5 features for which lane the agent is in, 27 features for the closest car in each lane (i.e., 9 features for the closest car in the agent's lane, 9 for the closest car to the left, and 9 for the closest car to the right), a similar set of 27 features for the closest ambulance in each lane, and another 27 features for closest trucks. Additionally, there are 20 distractor tree features. The action space includes moving left, moving right, and staying in the same lane. Human blind spots are states with trucks nearby, and agent blind spots are states with ambulances nearby.

5.5.2 Performance

We first show that learning agent and human blind spot models lead to safe transfer of control in the real world. The baselines are: An agent that executes in the real world by simply using π_{sim} learned from training and a human that acts in the world using the true human policy π_h. We compare these baselines to our approach which uses the learned blind spot models to appropriately hand over control. We

also compare against an "optimal hand-off" baseline, which we obtain by training a meta Q-learner agent in the real-world M_{real} that has two action choices at each state s_{real}: $\pi_{sim}(\phi(s_{real}))$ or $\pi_h(\phi(s_{real}))$. This meta-agent learns a Q-function specifying the value of taking each action at each state. The plotted reward is the average reward received in M_{real} when each hand-off policy is run.

In Figure 5-5a, we increase the percentage of agent blind spots in the real world in the Catcher domain, while keeping the total percentage of human and agent blind spots the same. The blindspot_handoff condition, which uses our learned blind spot models to act in the world, is able to do almost as well as the optimal_handoff policy. Executing the agent policy π_{sim} or the human policy π_h in the real world receives much less reward due to their inability to operate in their respective blind-spot regions. Figure 5-5b shows the percentage of times the agent acted using each hand-off policy. Our model gives more control to the agent while still obtaining high reward. The preference to the agent acting is due to our approach reasoning about features and transferring control to the human only when the state contains important real-world features the agent ignored in simulation. Further, our prior estimation step helps to better estimate the proportion of times the agent and human should act in the world, according to how likely their blind spots will occur. Giving greater control to the agent in areas where it can act well allows us to be more robust to human noise (see Table 5.3).

Figures 5-5c and 5-5d show similar results on the Driving domain: The agent and human policies perform worse as the percentage of blind spots is varied, while our model is able to achieve similar performance to the optimal handoff. In this domain, performance is reduced as the percentage of agent blind spots increases because the scenarios become harder, and the best reward that can be achieved decreases. In other words, although the blind spot models may accurately predict a blind spot, the human or agent policies might not have a safe action available to take due to the high difficulty of the state (e.g., too many ambulances in the surrounding region).

Table 5.1: Effect of distractor features on predicting blind spots in the real world in the Catcher domain.

Condition	Budget			
	100	1500	3000	4500
blindspot_handoff	-4.20 ± 0.49	0.82 ± 0.23	0.83 ± 0.21	1.30 ± 0.17
blindspot_handoff_F	-5.50 ± 0.48	-1.56 ± 0.62	-1.12 ± 0.48	-2.22 ± 0.70
agent_acting	-4.92 ± 0.34	-4.21 ± 0.36	-4.85 ± 0.41	-5.74 ± 0.54
human_acting	-4.54 ± 0.42	-4.66 ± 0.41	-4.05 ± 0.38	-4.48 ± 0.42
optimal_handoff	2.45 ± 0.11	2.55 ± 0.12	2.61 ± 0.10	2.39 ± 0.12

Table 5.2: Effect of distractor features on predicting blind spots in the real world in the Driving domain.

Condition	Budget			
	150	750	1250	1750
blindspot_handoff	34.89 ± 2.31	46.68 ± 1.94	48.83 ± 1.80	49.79 ± 1.72
blindspot_handoff_F	33.56 ± 1.54	33.58 ± 1.29	36.34 ± 1.50	37.00 ± 1.21
agent_acting	19.72 ± 2.35	22.40 ± 2.35	22.89 ± 2.35	22.29 ± 1.97
human_acting	36.36 ± 1.45	36.20 ± 1.32	34.51 ± 1.32	33.60 ± 1.45
optimal_handoff	58.73 ± 1.56	62.24 ± 1.39	58.08 ± 1.51	61.29 ± 1.72

5.5.3 Effect of Distractor Features

We next analyze the ability of our approach to prune away distractor features, which are features that differ between simulation and the real world but do not affect policy behavior. As we increase the amount of human demonstration data, performance in the world improves. Specifically, Tables 5.1 and 5.2 show that our approach blindspot_handoff is able to reach the performance of optimal_handoff with enough budget in both domains. Again, both agent or human acting obtain much less reward.

To highlight the benefit of our feature selection algorithm (Step 1 of our pipeline), we create a baseline that skips the feature selection step and instead uses all features in F (named blindspot_handoff_F). For example, in Driving, the experimental setup is that there are green trees in simulation and red trees in the demo/real worlds. Because the features are not pruned out, this hand-off policy focuses on red trees as an important difference between sim and demo/real. Thus, the hand-off policy always gives the human full control because red trees exists in all states, which is mistakenly associated with agent blind spots. Tables 5.1 and 5.2 show that in both domains,

Table 5.3: Effect of human suboptimal behavior on our model's performance in the Driving domain.

Condition	Percentage of Human Suboptimality		
	0.0	0.2	0.4
blindspot_handoff	65.91 ± 3.53	52.90 ± 2.73	53.28 ± 5.27
agent_acting	16.10 ± 4.34	31.10 ± 5.49	27.77 ± 5.83
human_acting	34.65 ± 3.16	19.32 ± 2.65	11.16 ± 3.58
optimal_handoff	63.96 ± 2.96	44.88 ± 3.60	30.32 ± 3.98

blindspot_handoff_F only does as well as π_h because it always gives control to the human. Our model learns to ignore distracting texture features in Catcher and tree features in Driving to learn an effective hand-off policy.

5.5.4 Effect of Human Suboptimality

In the results presented so far, the human had blind spots, but there were no random errors in the human policy, or existence of suboptimal actions in non-blind-spot states. Next, we analyze how robust our hand-off approach is to human suboptimality. As shown in Table 5.3, we see a degradation in the real-world performance as we increase human suboptimal behavior. However, our model is more robust than the baselines because it learns to prioritize the agent in cases where both can act, while still transferring control to the human when there is a high chance of an agent blind spot. Table 5.3 shows that our model gives more control to the agent than the baselines. Blindspot_handoff does better than optimal_handoff because the optimal policy transfers control based on a human that does not make random errors. Even though our model has no prior knowledge on human suboptimality, we can still be robust and handoff control to avoid blind spot regions and regions with random human errors.

5.6 Discussion

Our meta-learning approach for learning agent and human blind spots is an important step towards more active consideration of safety when transferring from simulation to

the real-world. In our work, we assume access to additional features that the agent can leverage to learn blind spots. The performance of our model will vary based on how correlated the available features are to error regions. For example, if the agent has a noisy feature that can predict 30% of ambulances, our model can learn to transfer control to the human for those states. If a perfect ambulance detector is added, accuracy would improve on predicting these blind spots.

Given some correlated features, our approach aims to learn blind spots *before* acting in the real world. In our setting, both the agent and the human can make errors, and we do not have access to a reward signal. Thus, we use action mismatches as noisy indicators of where blind spots may exist. We expect disagreements to be good indicators because (1) mismatches can indicate suboptimal behavior and (2) unsafe regions are generally subsets of suboptimal regions. Therefore, if the agent can predict action mismatches well, it is likely to over-approximate blind spots, resulting in a conservative hand-off policy that avoids suboptimal, and thus, unsafe states. It may be interesting in future work to combine these ideas with scenarios in which we have sparse reward signals to make more informed blind spot predictions.

We chose to learn a hand-off policy rather than update the agent's policy because estimating the true policy can require much more data than just learning blind spot regions. Further, the agent may not be capable of representing the correct policy. For example, if the agent does not have the full feature set, it may be able to identify a blind spot region but not able to update its policy for those different situations. Blind spot detection and safe transfer of control allows for better awareness of where the policy needs to be improved.

Finally, our approach assumes access to a set of features to learn limitations of the simulator. In future work, it would be interesting to apply these ideas to deep RL approaches, which are often featureless. One idea is to form a feature library that includes high-level features extracted from deep learning models [188] (e.g., superpixels or detector for car), which can then be used by our approach. Such a library can be used across applications to identify important high-level features missed in simulation that are important for the real world.

5.7 Conclusion

We have formalized the problem of agent and human blind spot detection in reinforcement learning to do safe transfer of control in the real world. In the proposed approach, the agent first learns important features from human demonstrations, which were ignored in its simulation environment. The agent then estimates its own blind spots as well as the human's to allow for safe execution in the world. Results show that using our blind spot models, the agent can learn to transfer control to the human intelligently to avoid blind spots in the real world. In future work, the representation and blind spot models can be updated in an online fashion with real-world feedback. The agent can also use active learning approaches to guide the collection of demonstration data for learning better blind spot models.

Chapter 6

Conclusion and Future Work

The goal of this thesis was to identify agent and human errors due to representation limitations. We introduced three problems that considered different assumptions on what an agent and a human could observe about the world and proposed approaches for discovering errors in these settings. In the following section, the contributions of this thesis are described and then, multiple avenues for future work based on the insights from this thesis are included.

6.1 Contributions

First, in Chapter 2, we presented the problem of identifying agent errors due to an agent's partially observable view of the world. Even though the agent was trained in a simulation environment, due to representational deficiencies, this learned policy resulted in costly errors in the real world environment. To discover such errors, our system collected and used human feedback, as the human helper was assumed to have full observability of the world. Due to a limited representation, the agent received many different signals from the human for a single observation. Our system aggregated these signals and trained a classifier to generalize the error predictions across the agent's feature space. We showed that the learned error model was helpful for an agent to query for help in limited regions in the real world and thus, avoid dangerous areas.

In Chapter 3, we applied the above approach to real user data and a real-world application. Specifically, human demonstrations and corrections data were collected across the same domains used in the previous simulated experiments. Error models were learned using a limited amount of user data. Further, we found that people preferred providing different types of feedback based on the type of domain, which can motivate the selection of the feedback type based on the properties of the domain. We then applied the method to an autonomous car application, in which an agent was tasked to avoid an obstacle and reach a goal cone. The agent had very simple sensing that resulted in the agent being unable to distinguish between the obstacles and the cones. We collected human demonstrations and applied our approach. Results showed that an agent learned to perform the task successfully through intelligent querying at high-danger regions.

The next step was to identify the errors of people, who may have limited representations of the world. In Chapter 4, we presented the problem of identifying the cause of human errors and developed a generative modelling approach to infer latent variables that affect human decision-making. The model included two factors affecting human actions: their blind spots, or missing observational features of the world, and the amount of noise in execution. Given a set of human demonstrations of a task, these two latent factors were inferred using exact and approximate inference techniques. We showed in our evaluations that our model was able to recover human errors occurring due to representation incompleteness.

Finally, in Chapter 5, we introduced the problem of discovering agent and human errors due to inattentional blindness. In this setting, both had full observability of the world but due to limited training environments, many factors were ignored and not used in action selection. Ignoring important features, however, can sometimes lead to costly mistakes in the real world. Thus, we complemented the agent with a human, who was trained in a different training environment that covered another part of the world. Our proposed approach used action mismatches and information about which features the agent and human were selectively paying attention to to identify who should act. Our experiments showed that an agent using our system learned to

handoff control intelligently for safe joint execution in the real world.

These three problems together provide a foundation for more exploration of representational limitations as a cause for errors. The proposed approaches enable AI systems and people to use the strengths of the other to discover failures and ultimately fix them.

6.2 Future Directions

We now outline a variety of future directions based on the problems and methods presented in this thesis. Specifically, we discuss:

- Combining the methods in this thesis with **other AI safety techniques** for increased robustness in real-world execution.
- **Augmenting agent and human representations** to enable long-term fixing of representational deficiencies.
- Developing **interpretable methods of communication** between agents and humans when representations of the two are not shared.
- Applying the proposed error discovery methods to **collaborative team tasks.**
- Exploring **real-world applications** for identifying agent and human representation errors.

We describe each of these ideas in further detail below.

6.2.1 Complementing with other AI safety techniques

This thesis provides methods for identifying errors of AI systems due to representation incompleteness when deploying a simulation policy in the real world. The proposed approaches help to develop more safe and self-aware AI systems. Many recent works [73, 102, 142, 77, 47] also aim to improve safety of trained AI systems when transferring from simulation to the real world. Combining our ideas with these works can enable stronger, more robust agents.

For example, training agents on more realistic training environments [200, 178] that more closely match the real world can result in higher-quality policies. Domain randomization in simulation training further [231, 187] provides stronger and more generalizable policies than training an agent on one variation of a task. Estimating an agent's uncertainty about the real world [133, 113] can also be incredibly useful for developing more self-aware agents. Incorporating different types of human feedback [66, 70] can lead to safe and efficient learning of AI systems as well. All of these techniques, combined with our approaches, can increase the robustness of agents when deployed into the real world.

6.2.2 Fixing and augmenting representations

Approaches for training more robust agents and for deploying agents safely in the world are useful for reducing catastrophes in real-world execution. However, if the errors are occurring due to representation incompleteness, the agent or human will always need external help in regions where the representation is insufficient. In this thesis, the discovered errors were avoided by transfer of control, but this can lead to excessive querying with no improvement over time. As a more long-term solution, agent and human representations can be augmented based on the identified error regions. Further, training environments can be iteratively improved based on the discovered failure cases.

In work by Unhelkar et al [235], agents work with humans in collaborative settings. The agent may not observe the world fully, including both hidden variables of its environment and latent states of the human teammate. The authors propose a Bayesian approach to recover latent decision factors. Another formulation that is commonly used for modelling uncertainties in the environment is POMDPs [131, 39]. In [62], the POMDP model is generalized to an infinite POMDP, which does not require prior knowledge of the state space size. In the work, a nonparametric approach is used to enable the agent to first model a small number of states and grow this space as the agent learns more. These generative approaches provide methods for augmenting representation when the true state of the world is unknown.

Related to augmenting representations, in deep learning literature, it is desirable to learn representations automatically. Representation learning [24, 48, 60] can thus be important for identifying important explanatory factors using low-level data. Even in these cases, once representations are learned based on a large amount of training data, it may be important to revise these representations online as agents are introduced to novel scenarios.

6.2.3 Agent-human interpretable communication

Another avenue for future work is developing methods for agent-human communication when representations of the two do not match. In team settings, communication can be incredibly important for high performance. The approach taken in this thesis is to share information in the form of actions. In some settings, however, it may be useful to communicate state information rather than simple observing the actions of other teammates. If the representations of the agent and human are different, however, it becomes challenging to communicate intent, goals, behavior, etc. Natural language can be one method of sharing information [12, 145]. However, other forms of interpretable communication can be used to support human-agent teaming.

One approach is to consider representations that are meaningful to both AI sytems and people. For example, representing the world in terms of objects can be an interpretable medium for communicating with a person. Our prior work proposed using objects mappings as a way to transfer knowledge between tasks through an interpretable medium [183, 185, 184]. We showed that agents transferred information among related tasks quickly and effectively when representing the world in terms of objects and determining relationships between similar objects [183]. We also demonstrated that these mappings were interpretable to people [185]. The downside is that while object mappings can be a concise and interpretable way of adapting to novel settings, the method needs to incorporate more information to describe the rich relationships of the world in a meaningful way to people.

In recent work [34, 83], object representations are learned that can generalize to new scenes with multiple objects. The proposed system is able to identify disentangled

representations that are interpretable and can potentially serve as a medium for better communication with people. In addition to objects as an interpretable medium, other interpretable machine learning techniques can be used to inform agent-human communication [5, 118, 116]. For instance, prototypical examples can be a meaningful way of interacting with people and can be one way of bridging the gap between different representations [117].

6.2.4 Human-agent collaborative team tasks

In this thesis, we considered settings where either a human helps an agent learn its errors or visa versa and a case where agents and humans share control. In all of these scenarios, however, only a single actor was required to act at each point in time. In collaborative team tasks, many of the same representational problems apply, but there are additional challenges when both agents and humans have important roles that cannot be replaced.

In cooperative tasks, it is important for teammates to not only learn about errors of each other, but also to work effectively as a team. If an error is identified, it may not be sufficient to transfer control because the task requires both the agent and the human to be involved. Discovering errors and acting appropriately for safe joint execution becomes more challenging in these complex collaborative scenarios. Team training techniques can be useful for making sure that teams are robust and avoid errors while jointly performing tasks. Cross training and perturbation training are two techniques used in human-human team training that were adopted to help robots and humans work together effectively across a range of scenarios [186, 156, 159]. In these problems, there was no emphasis on error discovery and refinement. Extending these training frameworks to explicitly identify and fix errors due to representation limitations can result in more robust human-robot teams.

Going beyond discovery of errors by augmenting and fixing representations can be particularly useful in collaborative tasks. For example, if latent states about the environment or human mental state are estimated, an agent can act to better support cooperative behavior. In prior work [158], we developed approaches for identifying

latent human preferences so that a robot can act appropriately and be an adaptive teammate. Applying the approaches from this thesis to human-robot team settings can be an important direction for enabling safer and more robust collaboration.

6.2.5 Additional applications in the real world

In future work, the methods from this thesis can be applied to several real-world applications. First, autonomous driving is one of the most relevant and applicable domains for our approaches. Current self-driving cars often make errors, and many can be attributed to poor observability or lack of awareness of what is important in the real world. Identifying when a car missed detecting a pedestrian can be useful for augmenting the car with more complex and rich sensors. One way of iteratively testing machine learning models on autonomous driving scenarios is AirSim, a high-fidelity driving simulator [200]. Other rich car simulators [63, 45] can be used as well to train and test algorithms for detecting errors in more realistic driving settings.

Another interesting application for understanding errors of AI systems and people is healthcare [82]. Doctors are tasked with treating patients given limited knowledge of the patient's health. A doctor generally has access to only a small set of features denoting the patient's state, including blood tests, radiology images, qualitative descriptions of symptoms, etc. Given this information, treatments need to be given appropriately for maximizing long-term patient health. Ideas from this thesis can be used to identify when more information could have been useful for making better decisions. In work by Li et al, for example, POMDPs are used to model partial observability in doctor-patient interactions, and the proposed approach is used to learn effective sequential policies to treat patients in this context [127, 128, 246, 226].

The methods can also be applied to educational settings, in which students may not pay attention to all important factors in a problem. Understanding student errors can be valuable for modifying training procedures for improved learning. Prior work [76, 115] has explored approaches for visualizing similarities and differences among student code snippets for programming assignments. Clustering student solutions based on similarities can enable teachers to understand students' thinking processes.

The methods introduced in this thesis can be helpful in categorizing human errors and identifying when a student is not paying attention to important features that matter for various assignments.

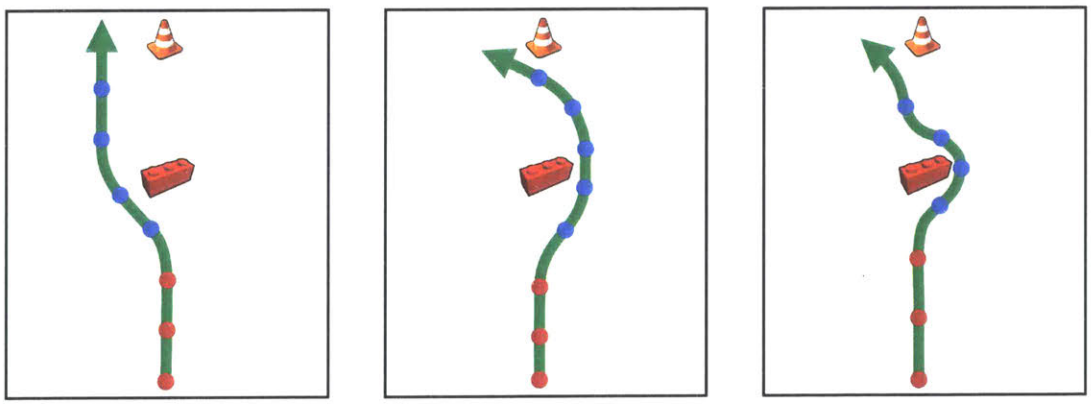
We hope this thesis provides insights to enable further exploration of AI and human safety, specifically on identifying failures of agents and people. With this awareness, new technologies can be introduced for iteratively reducing catastrophes. This continuous process of safely discovering errors and ultimately refining them will enable human-agent teams to be stronger and take advantage of each other's strengths.

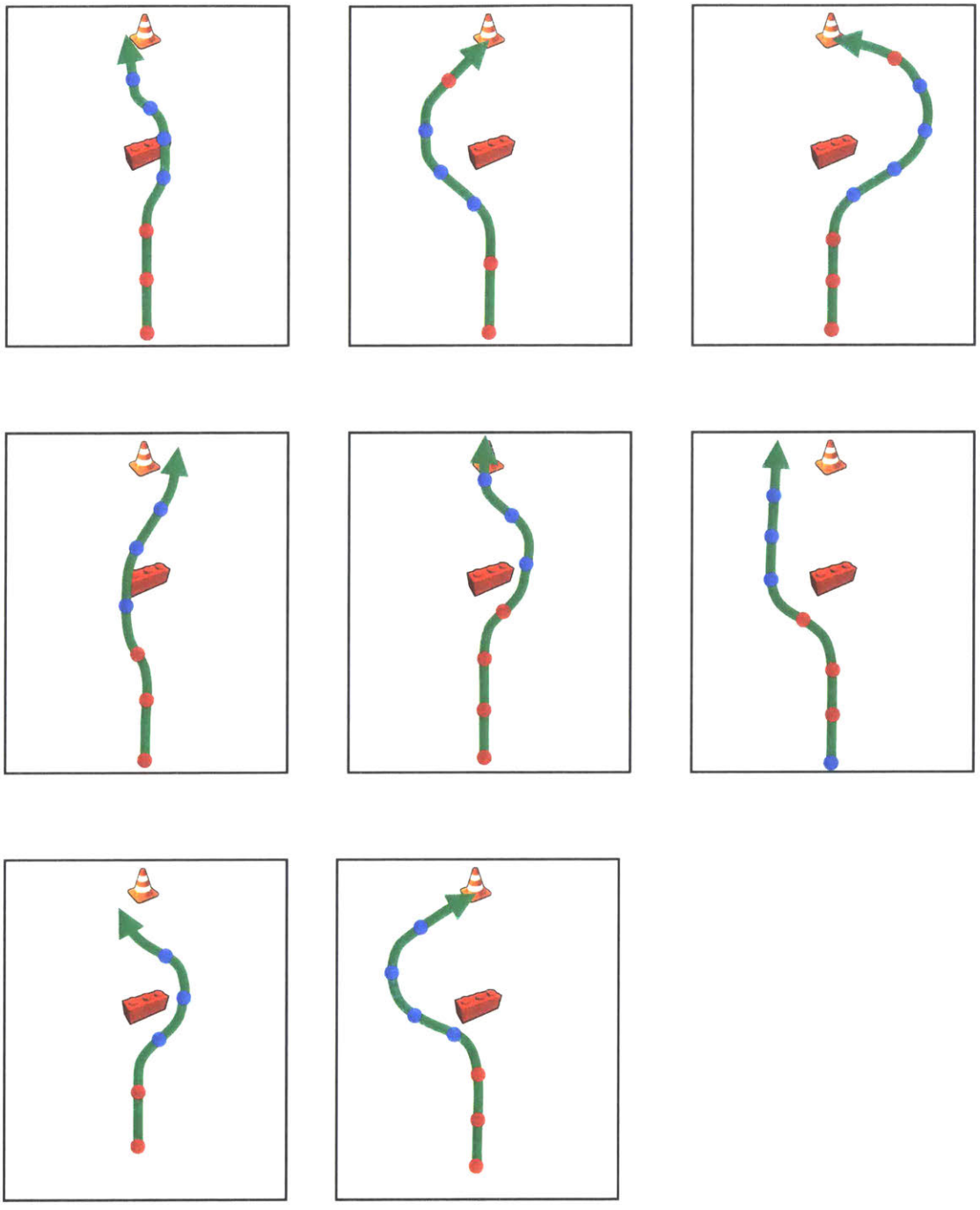
Appendix A

Human-in-the-loop Experiments and Demonstrations

Final demonstration trials for the autonomous car domain:

This appendix contains trajectories for all 11 final trials using the blind spot model to query for expert help on the autonomous car domain. The green arrow highlights the continuous path. Red dots denote discrete states where the agent queried for help (blind spots), while blue dots represent discrete states where the agent used the simulation policy to act autonomously.





Appendix B

Discovering Human Errors

Derivation of $P(\mathbf{b}, \eta | D)$

$$
\begin{aligned}
P(\mathbf{b}, \eta | D) &= \frac{P(D|\mathbf{b}, \eta) P(\mathbf{b}, \eta)}{P(D)} \\
&\propto P(\mathbf{b}, \eta) \prod_{(\mathbf{s}^i_{real}, a^i_h, e^i) \in D} P(\mathbf{s}^i_{real}, a^i_h, e^i | \mathbf{b}, \eta) \\
&\propto P(\mathbf{b}) P(\eta) \prod_i P(\mathbf{s}^i_{real}, a^i_h, e^i | \mathbf{b}, \eta) \\
&\propto P(\mathbf{b}) P(\eta) \prod_i \sum_{\mathbf{o}^i_h, \mathbf{s}^i_h} P(\mathbf{s}^i_{real}, a^i_h, e^i | \mathbf{o}^i_h, \mathbf{s}^i_h, \mathbf{b}, \eta) P(\mathbf{o}^i_h, \mathbf{s}^i_h | \mathbf{b}, \eta) \\
&\propto P(\mathbf{b}) P(\eta) \prod_i \sum_{\mathbf{o}^i_h, \mathbf{s}^i_h} \frac{P(\mathbf{s}^i_{real}, a^i_h, e^i, \mathbf{o}^i_h, \mathbf{s}^i_h, \mathbf{b}, \eta)}{P(\mathbf{o}^i_h, \mathbf{s}^i_h, \mathbf{b}, \eta)} P(\mathbf{o}^i_h, \mathbf{s}^i_h | \mathbf{b}, \eta) \\
&\propto P(\mathbf{b}) P(\eta) \prod_i \sum_{\mathbf{o}^i_h, \mathbf{s}^i_h} P(\mathbf{s}^i_{real}) P(\mathbf{o}^i_h | \mathbf{s}^i_{real}, \mathbf{b}) P(\mathbf{s}^i_h | \mathbf{o}^i_h) \pi^{*s}(a^i_h | \mathbf{s}^i_h, \eta) f_{acc}(e^i | \mathbf{s}^i_{real}, a^i_h) \\
&\propto P(\mathbf{b}) P(\eta) \prod_i P(\mathbf{s}^i_{real}) f_{acc}(e^i | \mathbf{s}^i_{real}, a^i_h) \sum_{\mathbf{o}^i_h} P(\mathbf{o}^i_h | \mathbf{s}^i_{real}, \mathbf{b}) \sum_{\mathbf{s}^i_h} P(\mathbf{s}^i_h | \mathbf{o}^i_h) \pi^{*s}(a^i_h | \mathbf{s}^i_h, \eta) \\
&= \frac{\cancel{\prod_i P(\mathbf{s}^i_{real}) f_{acc}(e^i | \mathbf{s}^i_{real}, a^i_h)} P(\mathbf{b}) P(\eta) \prod_i \sum_{\mathbf{o}^i_h} P(\mathbf{o}^i_h | \mathbf{s}^i_{real}, \mathbf{b}) \sum_{\mathbf{s}^i_h} P(\mathbf{s}^i_h | \mathbf{o}^i_h) \pi^{*s}(a^i_h | \mathbf{s}^i_h, \eta)}{\cancel{\prod_i P(\mathbf{s}^i_{real}) f_{acc}(e^i | \mathbf{s}^i_{real}, a^i_h)} \sum_{\mathbf{b}, \eta} P(\mathbf{b}) P(\eta) \prod_i \sum_{\mathbf{o}^i_h} P(\mathbf{o}^i_h | \mathbf{s}^i_{real}, \mathbf{b}) \sum_{\mathbf{s}^i_h} P(\mathbf{s}^i_h | \mathbf{o}^i_h) \pi^{*s}(a^i_h | \mathbf{s}^i_h, \eta)} \\
&= \frac{P(\mathbf{b}) P(\eta) \prod_i \sum_{\mathbf{o}^i_h} P(\mathbf{o}^i_h | \mathbf{s}^i_{real}, \mathbf{b}) \sum_{\mathbf{s}^i_h} P(\mathbf{s}^i_h | \mathbf{o}^i_h) \pi^{*s}(a^i_h | \mathbf{s}^i_h, \eta)}{\sum_{\mathbf{b}, \eta} P(\mathbf{b}) P(\eta) \prod_i \sum_{\mathbf{o}^i_h} P(\mathbf{o}^i_h | \mathbf{s}^i_{real}, \mathbf{b}) \sum_{\mathbf{s}^i_h} P(\mathbf{s}^i_h | \mathbf{o}^i_h) \pi^{*s}(a^i_h | \mathbf{s}^i_h, \eta)}
\end{aligned}
$$

Derivation of $P(\mathbf{s}_h^i|D)$

$$
\begin{aligned}
P(\mathbf{s}_h^i|D) &= \sum_{\mathbf{b},\eta} P(\mathbf{s}_h^i, \mathbf{b}, \eta|D) \\
&= \sum_{\mathbf{b},\eta} P(\mathbf{s}_h^i|\mathbf{b}, \eta, D) P(\mathbf{b}, \eta|D) \\
&= \sum_{\mathbf{b},\eta} P(\mathbf{s}_h^i|\mathbf{b}, \eta, \mathbf{s}_{real}^i, a_h^i, e^i) P(\mathbf{b}, \eta|D) \\
&= \sum_{\mathbf{b},\eta} \frac{P(\mathbf{s}_h^i, \mathbf{s}_{real}^i, a_h^i, e^i|\mathbf{b}, \eta)}{P(\mathbf{s}_{real}^i, a_h^i, e^i|\mathbf{b}, \eta)} P(\mathbf{b}, \eta|D) \\
&= \sum_{\mathbf{b},\eta} \frac{\sum_{\mathbf{o}_h^i} P(\mathbf{s}_h^i, \mathbf{s}_{real}^i, \mathbf{o}_h^i, a_h^i, e^i|\mathbf{b}, \eta)}{\sum_{\mathbf{s}_h^i}\sum_{\mathbf{o}_h^i} P(\mathbf{s}_h^i, \mathbf{s}_{real}^i, \mathbf{o}_h^i, a_h^i, e^i|\mathbf{b}, \eta)} P(\mathbf{b}, \eta|D) \\
&= \sum_{\mathbf{b},\eta} \frac{\sum_{\mathbf{o}_h^i} P(\mathbf{s}_{real}^i) P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) P(\mathbf{s}_h^i|\mathbf{o}_h^i) \pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta) f_{acc}(e^i|\mathbf{s}_{real}^i, a_h^i)}{\sum_{\mathbf{s}_h^i}\sum_{\mathbf{o}_h^i} P(\mathbf{s}_{real}^i) P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) P(\mathbf{s}_h^i|\mathbf{o}_h^i) \pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta) f_{acc}(e^i|\mathbf{s}_{real}^i, a_h^i)} P(\mathbf{b}, \eta|D) \\
&= \frac{\cancel{P(\mathbf{s}_{real}^i) f_{acc}(e^i|\mathbf{s}_{real}^i, a_h^i)}}{\cancel{P(\mathbf{s}_{real}^i) f_{acc}(e^i|\mathbf{s}_{real}^i, a_h^i)}} \sum_{\mathbf{b},\eta} \frac{\sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) P(\mathbf{s}_h^i|\mathbf{o}_h^i) \pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta)}{\sum_{\mathbf{s}_h^i}\sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) P(\mathbf{s}_h^i|\mathbf{o}_h^i) \pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta)} P(\mathbf{b}, \eta|D) \\
&= \sum_{\mathbf{b},\eta} P(\mathbf{b}, \eta|D) \left[\frac{\pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta) \sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) P(\mathbf{s}_h^i|\mathbf{o}_h^i)}{\sum_{\mathbf{s}_h^i} \pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta) \sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) P(\mathbf{s}_h^i|\mathbf{o}_h^i)} \right]
\end{aligned}
$$

Collapsed Gibbs Sampling Equations

$$
P(\mathbf{b}|D, \eta) = \frac{P(\mathbf{b}) \prod_i \sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) \sum_{\mathbf{s}_h^i} P(\mathbf{s}_h^i|\mathbf{o}_h^i) \pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta)}{\sum_{\mathbf{b}} P(\mathbf{b}) \prod_i \sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i, \mathbf{b}) \sum_{\mathbf{s}_h^i} P(\mathbf{s}_h^i|\mathbf{o}_h^i) \pi^{*s}(a_h^i|\mathbf{s}_h^i, \eta)} \tag{B.1}
$$

$$P(\eta|D,\mathbf{b}) = \frac{P(\eta)\prod_i \sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i,\mathbf{b}) \sum_{\mathbf{s}_h^i} P(\mathbf{s}_h^i|\mathbf{o}_h^i)\pi^{*s}(a_h^i|\mathbf{s}_h^i,\eta)}{\sum_\eta P(\eta)\prod_i \sum_{\mathbf{o}_h^i} P(\mathbf{o}_h^i|\mathbf{s}_{real}^i,\mathbf{b}) \sum_{\mathbf{s}_h^i} P(\mathbf{s}_h^i|\mathbf{o}_h^i)\pi^{*s}(a_h^i|\mathbf{s}_h^i,\eta)} \tag{B.2}$$

Bibliography

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

[2] David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 10–19, 2018.

[3] Baris Akgun, Maya Cakmak, Jae Wook Yoo, and Andrea Lockerd Thomaz. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398. ACM, 2012.

[4] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] David Alvarez-Melis and Tommi S Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7786–7795. Curran Associates Inc., 2018.

[6] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4):105–120, 2014.

[7] Haitham Bou Ammar, Eric Eaton, José Marcio Luna, and Paul Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *International Joint Conference on Artificial Intelligence*, pages 3345–3351, 2015.

[8] Haitham Bou Ammar, Rasul Tutunov, and Eric Eaton. Safe policy search for lifelong reinforcement learning with sublinear regret. In *International Conference on Machine Learning*, pages 2361–2369, 2015.

[9] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

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

[11] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *arXiv preprint arXiv:1806.06877*, 2018.

[12] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Edward C Williams, Mina Rhee, Lawson LS Wong, and Stefanie Tellex. Grounding natural language instructions to semantic goal representations for abstraction and generalization. *Autonomous Robots*, 43(2):449–468, 2019.

[13] Anish Athalye and Ilya Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[14] Chris Baker, Rebecca Saxe, and Joshua B Tenenbaum. Bayesian models of human action understanding. In *Advances in neural information processing systems*, pages 99–106, 2006.

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

[16] Chris L Baker and Joshua B Tenenbaum. Modeling human plan recognition using bayesian theory of mind. *Plan, activity, and intent recognition: Theory and practice*, pages 177–204, 2014.

[17] Bikramjit Banerjee and Matthew E Taylor. Coordination confidence based human-multi-agent transfer learning for collaborative teams. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, 2018.

[18] Gagan Bansal, Besmira Nushi, Ece Kamar, Daniel S Weld, Walter S Lasecki, and Eric Horvitz. Updates in human-ai teams: Understanding and addressing the performance/compatibility tradeoff. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2429–2437, 2019.

[19] André Barreto, Rémi Munos, Tom Schaul, and David Silver. Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*, 2016.

[20] Samuel Barrett, Noa Agmon, Noam Hazon, Sarit Kraus, and Peter Stone. Communicating with unknown teammates. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1433–1434. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[21] Samuel Barrett, Avi Rosenfeld, Sarit Kraus, and Peter Stone. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, October 2016.

[22] Samuel Barrett, Peter Stone, Sarit Kraus, and Avi Rosenfeld. Learning teammate models for ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, pages 57–63, 2012.

[23] Samuel Barrett, Matthew E Taylor, and Peter Stone. Transfer learning for reinforcement learning on a physical robot. In *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, 2010.

[24] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[25] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

[26] Erdem Bıyık and Dorsa Sadigh. Batch active preference-based learning of reward functions. *arXiv preprint arXiv:1810.04303*, 2018.

[27] Leonid Blouvshtein and Daniel Cohen-Or. Outlier detection for robust multi-dimensional scaling. *arXiv preprint arXiv:1802.02341*, 2018.

[28] Paul Bodesheim, Alexander Freytag, Erik Rodner, Michael Kemmler, and Joachim Denzler. Kernel null space methods for novelty detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3374–3381, 2013.

[29] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[30] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[31] Cynthia Breazeal, Cory D Kidd, Andrea Lockerd Thomaz, Guy Hoffman, and Matt Berlin. Effects of nonverbal communication on efficiency and robustness in human-robot teamwork. In *2005 IEEE/RSJ international conference on intelligent robots and systems*, pages 708–713. IEEE, 2005.

[32] Gordon Briggs and Matthias Scheutz. Facilitating mental modeling in collaborative human-robot interaction through adverbial cues. In *Proceedings of the SIGDIAL 2011 Conference*, pages 239–247, 2011.

[33] Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 316–324, 2014.

[34] Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

[35] Maya Cakmak, Crystal Chao, and Andrea L Thomaz. Designing interactions for robot active learners. *IEEE Transactions on Autonomous Mental Development*, 2(2):108–118, 2010.

[36] Maya Cakmak and Andrea L Thomaz. Designing robot learners that ask good questions. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 17–24. ACM, 2012.

[37] Colin Campbell and Kristin P Bennett. A linear programming approach to novelty detection. In *Advances in neural information processing systems*, pages 395–401, 2001.

[38] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30(4):891–927, 2016.

[39] Anthony Rocco Cassandra. *Exact and approximate algorithms for partially observable Markov decision processes*. Brown University, 1998.

[40] Tathagata Chakraborti and Subbarao Kambhampati. Algorithms for the greater good! on mental modeling and acceptable symbiosis in human-ai collaboration. *arXiv preprint arXiv:1801.09854*, 2018.

[41] Tathagata Chakraborti, Sarath Sreedharan, and Subbarao Kambhampati. Human-aware planning revisited: A tale of three models. In *IJCAI-ECAI XAI/ICAPS XAIP Workshops*, 2018.

[42] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[43] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Outlier detection: A survey. *ACM Computing Surveys*, 2007.

[44] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[45] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Nießner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

[46] Crystal Chao, Maya Cakmak, and Andrea L Thomaz. Transparent active learning for robots. In *Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on*, pages 317–324. IEEE, 2010.

[47] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *arXiv preprint arXiv:1810.05687*, 2018.

[48] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[49] Joannes El Chliaoutakis, Panayotis Demakakos, Georgia Tzamalouka, Vassiliki Bakou, Malamatenia Koumaki, and Christina Darviri. Aggressive behavior while driving as predictor of self-reported car crashes. *Journal of safety Research*, 33(4):431–443, 2002.

[50] Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*, 2017.

[51] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

[52] John D Co-Reyes, YuXuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. *arXiv preprint arXiv:1806.02813*, 2018.

[53] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[54] Gergely Csibra. Action mirroring and action understanding: An alternative account. *Sensorymotor foundations of higher cognition. Attention and performance XXII*, pages 435–459, 2008.

[55] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[56] Wenyuan Dai, Yuqiang Chen, Gui-Rong Xue, Qiang Yang, and Yong Yu. Translated learning: Transfer learning across different feature spaces. In *Advances in neural information processing systems*, pages 353–360, 2009.

[57] Alexander Philip Dawid and Allan M Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied statistics*, pages 20–28, 1979.

[58] Coline Devin, Pieter Abbeel, Trevor Darrell, and Sergey Levine. Deep object-centric representations for generalizable robot learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7111–7118. IEEE, 2018.

[59] Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247. ACM, 2008.

[60] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

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

[62] Finale Doshi-Velez. The infinite partially observable markov decision process. In *Advances in neural information processing systems*, pages 477–485, 2009.

[63] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[64] Kenji Doya, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural computation*, 14(6):1347–1369, 2002.

[65] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[66] Ishan Durugkar, Matthew Hausknecht, Adith Swaminathan, and Patrick MacAlpine. Multi-preference actor critic. *arXiv preprint arXiv:1904.03295*, 2019.

[67] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *arXiv preprint arXiv:1711.06782*, 2017.

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

[69] Dieter Fox, Sebastian Thrun, Wolfram Burgard, and Frank Dellaert. Particle filters for mobile robot localization. In *Sequential Monte Carlo methods in practice*, pages 401–428. Springer, 2001.

[70] Christopher Frye and Ilya Feige. Parenting: Safe reinforcement learning from human input. *arXiv preprint arXiv:1902.06766*, 2019.

[71] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[72] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[73] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[74] Samuel J Gershman. Reinforcement learning and causal models. *The Oxford handbook of causal reasoning*, page 295, 2017.

[75] Miriam Gil, Vicente Pelechano, Joan Fons, and Manoli Albert. Designing the human in the loop of self-adaptive systems. In *International Conference on Ubiquitous Computing and Ambient Intelligence*, pages 437–449. Springer, 2016.

[76] Elena L Glassman, Jeremy Scott, Rishabh Singh, Philip J Guo, and Robert C Miller. Overcode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 22(2):7, 2015.

[77] Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. Sim-to-real transfer with neural-augmented robot simulation. In *Conference on Robot Learning*, pages 817–828, 2018.

[78] Noah J Goodall. Can you program ethics into a self-driving car? *IEEE Spectrum*, 53(6):28–58, 2016.

[79] Noah D Goodman, Chris L Baker, Elizabeth Baraff Bonawitz, Vikash K Mansinghka, Alison Gopnik, Henry Wellman, Laura Schulz, and Joshua B Tenenbaum. Intuitive theories of mind: A rational approach to false belief. In *Proceedings of the twenty-eighth annual conference of the cognitive science society*, volume 6. Cognitive Science Society Vancouver, 2006.

[80] Noah D Goodman, Joshua B Tenenbaum, Jacob Feldman, and Thomas L Griffiths. A rational analysis of rule-based concept learning. *Cognitive science*, 32(1):108–154, 2008.

[81] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Toward supervised anomaly detection. *Journal of Artificial Intelligence Research*, 46:235–262, 2013.

[82] Omer Gottesman, Fredrik Johansson, Matthieu Komorowski, Aldo Faisal, David Sontag, Finale Doshi-Velez, and Leo Anthony Celi. Guidelines for reinforcement learning in healthcare. *Nat Med*, 25(1):16–18, 2019.

[83] Klaus Greff, Raphaël Lopez Kaufmann, Rishab Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. *arXiv preprint arXiv:1903.00450*, 2019.

[84] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2625–2633, 2013.

[85] Thomas L Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in cognitive sciences*, 14(8):357–364, 2010.

[86] Thomas L Griffiths and Joshua B Tenenbaum. Optimal predictions in everyday cognition. *Psychological science*, 17(9):767–773, 2006.

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

[88] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, 2014.

[89] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018.

[90] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning. In *Advances in neural information processing systems*, pages 3909–3917, 2016.

[91] Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe exploration for reinforcement learning. In *ESANN*, pages 143–148, 2008.

[92] Brent Harrison, Upol Ehsan, and Mark O Riedl. Guiding reinforcement learning exploration using natural language. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1956–1958. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[93] Zhanpeng He, Ryan Julian, Eric Heiden, Hejia Zhang, Stefan Schaal, Joseph Lim, Gaurav Sukhatme, and Karol Hausman. Zero-shot skill composition and simulation-to-real transfer by learning task representations. *arXiv preprint arXiv:1810.02422*, 2018.

[94] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[95] Janine Hoelscher, Dorothea Koert, Jan Peters, and Joni Pajarinen. Utilizing human feedback in pomdp execution and specification. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 104–111. IEEE, 2018.

[96] Heiko Hoffmann. Kernel pca for novelty detection. *Pattern recognition*, 40(3):863–874, 2007.

[97] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. *arXiv preprint arXiv:1810.02912*, 2018.

[98] Shariq Iqbal and Fei Sha. Coordinated exploration via intrinsic rewards for multi-agent reinforcement learning. *arXiv preprint arXiv:1905.12127*, 2019.

[99] David Isele, José Marcio Luna, Eric Eaton, V Gabriel, James Irwin, Brandon Kallaher, and Matthew E Taylor. Lifelong learning for disturbance rejection on mobile robots. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3993–3998. IEEE, 2016.

[100] David Isele, Mohammad Rostami, and Eric Eaton. Using task features for zero-shot knowledge transfer in lifelong learning. In *IJCAI*, pages 1620–1626, 2016.

[101] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

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

[103] Eric Jang, Coline Devin, Vincent Vanhoucke, and Sergey Levine. Grasp2vec: Learning object representations from self-supervised grasping. *arXiv preprint arXiv:1811.06964*, 2018.

[104] Emilie MD Jean-Baptiste, Pia Rotshtein, and Martin Russell. Pomdp based action planning and human error detection. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 250–265. Springer, 2015.

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

[106] Ryan Julian, Eric Heiden, Zhanpeng He, Hejia Zhang, Stefan Schaal, Joseph Lim, Gaurav Sukhatme, and Karol Hausman. Scaling simulation-to-real transfer by learning composable robot skills. *arXiv preprint arXiv:1809.10253*, 2018.

[107] Sebastian Junges, Nils Jansen, Christian Dehnert, Ufuk Topcu, and Joost-Pieter Katoen. Safety-constrained reinforcement learning for mdps. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 130–146. Springer, 2016.

[108] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[109] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[110] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[111] Ece Kamar. Directions in hybrid intelligence: Complementing ai systems with human intelligence. In *IJCAI*, pages 4070–4073, 2016.

[112] Sertac Karaman, Ariel Anders, Michael Boulet, Jane Connor, Kenneth Gregson, Winter J Guerra, Owen Guldner, Mubarik Mohamoud, Brian Plancher, Robert Shin, and John Vivilecchia. Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at MIT. March 2017.

[113] Zachary Kenton, Angelos Filos, Owain Evans, and Yarin Gal. Generalizing from a few environments in safety-critical reinforcement learning. *arXiv preprint arXiv:1907.01475*, 2019.

[114] Ramtin Keramati, Jay Whang, Patrick Cho, and Emma Brunskill. Strategic object oriented reinforcement learning. *arXiv preprint arXiv:1806.00175*, 2018.

[115] Been Kim, Elena Glassman, Brittney Johnson, and Julie Shah. ibcm: Interactive bayesian case model empowering humans via intuitive interaction. 2015.

[116] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. In *Advances in Neural Information Processing Systems*, pages 2280–2288, 2016.

[117] Been Kim, Cynthia Rudin, and Julie A Shah. The bayesian case model: A generative approach for case-based reasoning and prototype classification. In *Advances in Neural Information Processing Systems*, pages 1952–1960, 2014.

[118] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). *arXiv preprint arXiv:1711.11279*, 2017.

[119] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM, 2009.

[120] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016.

[121] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[122] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Eric Horvitz. Identifying unknown unknowns in the open world: Representations and policies for guided exploration. In *AAAI*, pages 2124–2132, 2017.

[123] Himabindu Lakkaraju and Cynthia Rudin. Learning cost-effective and interpretable treatment regimes. In *Artificial Intelligence and Statistics*, pages 166–175, 2017.

[124] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer, 2000.

[125] Christian Lebiere, Florian Jentsch, and Scott Ososky. Cognitive models of decision making processes for human-robot interaction. In *International Conference on Virtual, Augmented and Mixed Reality*, pages 285–294. Springer, 2013.

[126] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Feature construction for inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1342–1350, 2010.

[127] Luchen Li, Matthieu Komorowski, and Aldo A Faisal. Optimizing sequential medical treatments with auto-encoding heuristic search in pomdps. *arXiv preprint arXiv:1905.07465*, 2019.

[128] Michael Li, D Ladner, S Miller, and D Classen. Identifying hospital patient safety problems in real-time with electronic medical record data using an ensemble machine learning model. *Int J Clin Med Inform*, 1(1):43–58, 2018.

[129] Yuezhang Li, Katia Sycara, and Rahul Iyer. Object-sensitive deep reinforcement learning. *arXiv preprint arXiv:1809.06064*, 2018.

[130] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[131] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. Efficient dynamic-programming updates in partially observable markov decision processes. 1995.

[132] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

[133] Björn Lütjens, Michael Everett, and Jonathan P How. Safe reinforcement learning with model uncertainty estimates. *arXiv preprint arXiv:1810.08700*, 2018.

[134] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2970–2977, 2019.

[135] Chen Ma, Junfeng Wen, and Yoshua Bengio. Universal successor representations for transfer reinforcement learning. *arXiv preprint arXiv:1804.03758*, 2018.

[136] Arien Mack. Inattentional blindness: Looking without seeing. *Current Directions in Psychological Science*, 12(5):180–184, 2003.

[137] Arien Mack, Irvin Rock, et al. *Inattentional blindness*. MIT press, 1998.

[138] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[139] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, pages 7047–7058, 2018.

[140] Markos Markou and Sameer Singh. Novelty detection: a reviewâĂŤpart 1: statistical approaches. *Signal processing*, 83(12):2481–2497, 2003.

[141] Markos Markou and Sameer Singh. Novelty detection: a reviewâĂŤpart 2:: neural network based approaches. *Signal processing*, 83(12):2499–2521, 2003.

[142] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018.

[143] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Vivian Weller. Concrete problems for autonomous vehicle safety: advantages of bayesian deep learning. International Joint Conferences on Artificial Intelligence, Inc., 2017.

[144] R Andrew McCallum. Hidden state and reinforcement learning with instance-based state identification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(3):464–473, 1996.

[145] Nikhil Mehta and Dan Goldwasser. Improving natural language interaction with robots using advice. *arXiv preprint arXiv:1905.04655*, 2019.

[146] Josh Merel, Yuval Tassa, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017.

[147] Fernand Meyer. Color image segmentation. pages 303–306, 1992.

[148] Ida Momennejad, Evan M Russek, Jin H Cheong, Matthew M Botvinick, Nathaniel Douglass Daw, and Samuel J Gershman. The successor representation in human reinforcement learning. *Nature Human Behaviour*, 1(9):680, 2017.

[149] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

[150] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. *arXiv preprint arXiv:1810.01257*, 2018.

[151] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pages 9191–9200, 2018.

[152] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Grounding language for transfer in deep reinforcement learning. *arXiv preprint arXiv:1708.00133*, 2017.

[153] Andrew Y Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

[154] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

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

[156] Stefanos Nikolaidis, Przemyslaw Lasota, Ramya Ramakrishnan, and Julie Shah. Improved human–robot team performance through cross-training, an approach inspired by human team training practices. *The International Journal of Robotics Research*, 34(14):1711–1730, 2015.

[157] Stefanos Nikolaidis, Swaprava Nath, Ariel D Procaccia, and Siddhartha Srinivasa. Game-theoretic modeling of human adaptation in human-robot collaboration. In *Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction*, pages 323–331. ACM, 2017.

[158] Stefanos Nikolaidis, Ramya Ramakrishnan, Keren Gu, and Julie Shah. Efficient model learning from joint-action demonstrations for human-robot collaborative tasks. In *Proceedings of the tenth annual ACM/IEEE international conference on human-robot interaction*, pages 189–196. ACM, 2015.

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

[160] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[161] OpenAI, :, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning Dexterous In-Hand Manipulation. *ArXiv e-prints*, August 2018.

[162] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pages 4026–4034, 2016.

[163] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[164] Peter C Pantelis, Chris L Baker, Steven A Cholewiak, Kevin Sanik, Ari Weinstein, Chia-Chien Wu, Joshua B Tenenbaum, and Jacob Feldman. Inferring the intentional states of autonomous virtual agents. *Cognition*, 130(3):360–379, 2014.

[165] Sungrae Park, JunKeon Park, Su-Jin Shin, and Il-Chul Moon. Adversarial dropout for supervised and semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[166] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[167] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[168] Xuefeng Peng, Yi Ding, David Wihl, Omer Gottesman, Matthieu Komorowski, Li-wei H Lehman, Andrew Ross, Aldo Faisal, and Finale Doshi-Velez. Improving sepsis treatment strategies by combining deep and kernel-based reinforcement learning. In *AMIA Annual Symposium Proceedings*, volume 2018, page 887. American Medical Informatics Association, 2018.

[169] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[170] Joelle Pineau, Nicholas Roy, and Sebastian Thrun. A hierarchical approach to pomdp planning and execution. In *Workshop on hierarchy and memory in reinforcement learning (ICML)*, volume 65, page 51, 2001.

[171] Bharat Prakash, Mohit Khatwani, Nicholas Waytowich, and Tinoosh Mohsenin. Improving safety in reinforcement learning using model-based architectures and human intervention. *arXiv preprint arXiv:1903.09328*, 2019.

[172] David V Pynadath, Michael J Barnes, Ning Wang, and Jessie YC Chen. Transparency communication for machine learning in human-automation interaction. In *Human and Machine Learning*, pages 75–90. Springer, 2018.

[173] Siyuan Qi and Song-Chun Zhu. Intent-aware multi-agent reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7533–7540. IEEE, 2018.

[174] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. 3(3.2):5, 2009.

[175] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[176] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. Reverse nearest neighbors in unsupervised distance-based outlier detection. *IEEE transactions on knowledge and data engineering*, 27(5):1369–1382, 2015.

[177] Aniruddh Raghu, Matthieu Komorowski, Imran Ahmed, Leo Celi, Peter Szolovits, and Marzyeh Ghassemi. Deep reinforcement learning for sepsis treatment. *arXiv preprint arXiv:1711.09602*, 2017.

[178] Aniruddh Raghu, Matthieu Komorowski, Leo Anthony Celi, Peter Szolovits, and Marzyeh Ghassemi. Continuous state-space models for optimal sepsis treatment-a deep reinforcement learning approach. *arXiv preprint arXiv:1705.08422*, 2017.

[179] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. Modeling others using oneself in multi-agent reinforcement learning. *arXiv preprint arXiv:1802.09640*, 2018.

[180] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *Urbana*, 51(61801):1–4, 2007.

[181] Ramya Ramakrishnan, Ece Kamar, Debadeepta Dey, Julie Shah, and Eric Horvitz. Discovering blind spots in reinforcement learning. *Proceedings of the 17th international joint conference on Autonomous agents and multiagent systems*, 2018.

[182] Ramya Ramakrishnan, Ece Kamar, Besmira Nushi, Debadeepta Dey, Julie Shah, and Eric Horvitz. Overcoming blind spots in the real world: Leveraging complementary abilities for joint execution. 2019.

[183] Ramya Ramakrishnan, Karthik Narasimhan, and Julie Shah. Interpretable transfer for reinforcement learning based on object similarities. 2016.

[184] Ramya Ramakrishnan and Julie Shah. Towards interpretable explanations for transfer learning in sequential tasks. In *2016 AAAI Spring Symposium Series*, 2016.

[185] Ramya Ramakrishnan and Julie A Shah. Knowledge transfer from a human perspective. *AAMAS Transfer in Reinforcement Learning Workshop*, 2017.

[186] Ramya Ramakrishnan, Chongjie Zhang, and Julie Shah. Perturbation training for human-robot teams. *Journal of Artificial Intelligence Research*, 59:495–541, 2017.

[187] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv preprint arXiv:1906.01728*, 2019.

[188] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

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

[190] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[191] Avraham Ruderman, Richard Everett, Bristy Sikder, Hubert Soyer, Jonathan Uesato, Ananya Kumar, Charlie Beattie, and Pushmeet Kohli. Uncovering surprising behaviors in reinforcement learning via worst-case analysis. 2018.

[192] Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[193] Paul Ruvolo and Eric Eaton. Ella: An efficient lifelong learning algorithm. In *International Conference on Machine Learning*, pages 507–515, 2013.

[194] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In *Machine Learning Proceedings 1992*, pages 385–393. Elsevier, 1992.

[195] William Saunders, Girish Sastry, Andreas Stuhlmueller, and Owain Evans. Trial without error: Towards safe reinforcement learning via human intervention. *arXiv preprint arXiv:1707.05173*, 2017.

[196] Rebecca Saxe. Against simulation: the argument from error. *Trends in cognitive sciences*, 9(4):174–179, 2005.

[197] Kristin E Schaefer, Edward R Straub, Jessie YC Chen, Joe Putney, and Arthur W Evans III. Communicating intent to develop shared situation awareness and engender trust in human-agent teams. *Cognitive Systems Research*, 46:26–39, 2017.

[198] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000.

[199] Ankit Shah, Pritish Kamath, Julie A Shah, and Shen Li. Bayesian inference of temporal task specifications from demonstrations. In *Advances in Neural Information Processing Systems*, pages 3804–3813, 2018.

[200] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[201] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*, 2015.

[202] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[203] Aashish Sheshadri and Matthew Lease. Square: A benchmark for research on computing crowd consensus. In *First AAAI Conference on Human Computation and Crowdsourcing*, 2013.

[204] Rowland R Sillito and Robert B Fisher. Semi-supervised learning for anomalous trajectory detection. In *BMVC*, volume 1, pages 035–1, 2008.

[205] Felipe Leno Da Silva and Anna Helena Reali Costa. Object-oriented curriculum generation for reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1026–1034. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[206] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[207] Daniel J Simons and Christopher F Chabris. Gorillas in our midst: Sustained inattentional blindness for dynamic events. *perception*, 28(9):1059–1074, 1999.

[208] Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 520–527. AUAI Press, 2004.

[209] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[210] Bradly C Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *arXiv preprint arXiv:1703.01703*, 2017.

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

[212] Andreas Stuhlmüller and Noah D Goodman. Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs. *Cognitive Systems Research*, 28:80–99, 2014.

[213] Yanan Sui, Alkis Gotovos, Joel Burdick, and Andreas Krause. Safe exploration for optimization with gaussian processes. In *International Conference on Machine Learning*, pages 997–1005, 2015.

[214] Umar Syed, Michael Bowling, and Robert E Schapire. Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, pages 1032–1039. ACM, 2008.

[215] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[216] Kartik Talamadupula, Gordon Briggs, Tathagata Chakraborti, Matthias Scheutz, and Subbarao Kambhampati. Coordination in human-robot teams using mental modeling and plan recognition. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2957–2962. IEEE, 2014.

[217] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[218] Matthew E Taylor, Gregory Kuhlmann, and Peter Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 283–290. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[219] Matthew E Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pages 879–886. ACM, 2007.

[220] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[221] Matthew E Taylor, Peter Stone, and Yaxin Liu. Transfer learning via intertask mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(Sep):2125–2167, 2007.

[222] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, volume 3, page 6, 2017.

[223] Andrea Thomaz, Guy Hoffman, Maya Cakmak, et al. Computational human-robot interaction. *Foundations and Trends® in Robotics*, 4(2-3):105–223, 2016.

[224] Sebastian Thrun. Monte carlo pomdps. In *Advances in neural information processing systems*, pages 1064–1070, 2000.

[225] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017.

[226] Eric J Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine*, 25(1):44, 2019.

[227] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[228] Lisa Torrey and Matthew Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1053–1060. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[229] Eric Christopher Townsend. Estimating short-term human intent for physical human-robot co-manipulation. 2017.

[230] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[231] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 969–977, 2018.

[232] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration in finite markov decision processes with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 4312–4320, 2016.

[233] Jonathan Uesato, Ananya Kumar, Csaba Szepesvari, Tom Erez, Avraham Ruderman, Keith Anderson, Nicolas Heess, Pushmeet Kohli, et al. Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures. *arXiv preprint arXiv:1812.01647*, 2018.

[234] Tomer Ullman, Chris Baker, Owen Macindoe, Owain Evans, Noah Goodman, and Joshua B Tenenbaum. Help or hinder: Bayesian models of social goal inference. In *Advances in neural information processing systems*, pages 1874–1882, 2009.

[235] Vaibhav V Unhelkar and Julie A Shah. Learning models of sequential decision-making without complete state specification using bayesian nonparametric inference and active querying. 2018.

[236] Tanja Urbancic and Ivan Bratko. Reconstructing human skill with machine learning. In *Proceedings of the 11th european conference on artificial intelligence*, pages 498–502. John Wiley & Sons, Inc., 1994.

[237] J van Baar, R Corcodel, A Sullivan, D Jha, D Romeres, and DN Nikovski. Simulation to real transfer learning with robustified policies for robot tasks. 2018.

[238] Frank Van Overwalle and Kris Baetens. Understanding others' actions and goals by mirror and mentalizing systems: a meta-analysis. *Neuroimage*, 48(3):564–584, 2009.

[239] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

[240] Thomas J Walsh, István Szita, Carlos Diuk, and Michael L Littman. Exploring compact reinforcement-learning representations with linear regression. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 591–598. AUAI Press, 2009.

[241] Weiran Wang, Raman Arora, Karen Livescu, and Jeff Bilmes. On deep multi-view representation learning. In *International Conference on Machine Learning*, pages 1083–1092, 2015.

[242] Kyle Hollins Wray, Luis Pineda, and Shlomo Zilberstein. Hierarchical approach to transfer of control in semi-autonomous systems. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1285–1286. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[243] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 4:6, 2000.

[244] Dan Xu, Elisa Ricci, Yan Yan, Jingkuan Song, and Nicu Sebe. Learning deep representations of appearance and motion for anomalous event detection. *arXiv preprint arXiv:1510.01553*, 2015.

[245] X Jessie Yang, Anuj Pradhan, Dawn Tilbury, Lionel Robert, et al. Human autonomous vehicles interactions: An interdisciplinary approach. 2018.

[246] Kun-Hsing Yu and Isaac S Kohane. Framing the challenges of artificial intelligence in medicine. *BMJ Qual Saf*, 28(3):238–241, 2019.

[247] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018.

[248] Bianca Zadrozny, John Langford, and Naoki Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 435–442. IEEE, 2003.

[249] Eric Zhan, Stephan Zheng, Yisong Yue, Long Sha, and Patrick Lucey. Generative multi-agent behavioral cloning. *arXiv*, 2018.

[250] Fangyi Zhang, Jürgen Leitner, Zongyuan Ge, Michael Milford, and Peter Corke. Adversarial discriminative sim-to-real transfer of visuo-motor policies. *arXiv preprint arXiv:1709.05746*, 2017.

[251] Qingchen Zhang, Laurence Tianruo Yang, Zhikui Chen, Peng Li, and Fanyu Bu. An adaptive dropout deep computation model for industrial iot big data learning with crowdsourcing to cloud computing. *IEEE Transactions on Industrial Informatics*, 15(4):2330–2337, 2018.

[252] Shao Zhifei and Er Meng Joo. A survey of inverse reinforcement learning techniques. *International Journal of Intelligent Computing and Cybernetics*, 5(3):293–311, 2012.

[253] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.

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

[255] Sixiang Zuo, Zhiyang Wang, Xiaorui Zhu, and Yongsheng Ou. Continuous reinforcement learning from human demonstrations with integrated experience replay for autonomous driving. In *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2450–2455. IEEE, 2017.