Inverse Reinforcement Learning for Robotic Applications: Hidden Variables, Multiple Experts and Unknown Dynamics

by

Kenneth Bogert

(Under the Direction of Prashant Doshi)

Abstract

Robots deployed into many real-world scenarios are expected to face situations that their designers could not anticipate. Machine learning is an effective tool for extending the capabilities of these robots by allowing them to adapt their behavior to the situation in which they find themselves. Most machine learning techniques are applicable to learning either static elements in an environment or elements with simple dynamics. We wish to address the problem of learning the behavior of other intelligent agents that the robot may encounter. To this end, we extend a well-known Inverse Reinforcement Learning (IRL) algorithm, Maximum Entropy IRL, to address challenges expected to be encountered by autonomous robots during learning. These include: occlusion of the observed agent's state space due to limits of the learner's sensors or objects in the environment, the presence of multiple agents who interact, and partial knowledge of other agents' dynamics. Our contributions are investigated with experiments using simulated and real world robots. These experiments include learning a fruit sorting task from human demonstrations and autonomously penetrating a perimeter patrol. Our work takes several important steps towards deploying IRL alongside other machine learning methods for use by autonomous robots.

Index words: robotics, inverse reinforcement learning, machine learning, Markov decision process

Inverse Reinforcement Learning for Robotic Applications: Hidden Variables, Multiple Experts and Unknown Dynamics

by

Kenneth Bogert

B.S., The University of North Carolina at Asheville, Asheville, NC, 2004

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

Doctor of Philosophy

Athens, Georgia

2016

Inverse Reinforcement Learning for Robotic Applications: Hidden Variables, Multiple Experts and Unknown Dynamics

by

Kenneth Bogert

Dedicated to the world, for allowing me run around on it for a little while.

Acknowledgments

First and foremost, I want to thank my advisor, Dr. Prashant Doshi, for all he has done these past five years. His motivation and commitment to excellence has prepared me for success as a professor of computer science. Our lengthy conversations and collaboration on my research have significantly contributed to my growth as a scholar and establishing international recognition of our work.

I would like to thank my lab-mates for the debates and discussion that helped me along my journey, including Ekhlas Sonu, BJ Wimpey, Muthu Chandrasekaran, Roi Ceren, William Richardson, Fadel Adoe, Shervin Shahryari, Sina Solaimanpour, Indrajit Das, Kedar Marathe, Maulesh Trivedi, Shibo Li, Yu Qiu, Shan Khan, Anuja Nagare, Anousha Mesbah, and Xia Qu.

Thanks to my close friends, the Family and Norwood, for the support and encouragement I needed to see this through; to Laura, for your patience, wisdom and understanding poured out at a moment's notice; to Kyle Johnsen for providing endless advice and expertise that I will likely make use of for the rest of my life; to my family for your love and encouragement; and finally, thanks to Dr. Bert and Hertha for pushing me towards graduate school, none of this would have happened if not for you!

Publication List:

1. Kenneth Bogert and Prashant Doshi. ”Multi-Robot Inverse Reinforcement Learning under Occlusion with Estimation of State Transitions” Under Review.

2. Kenneth Bogert and Prashant Doshi. ”Inverse Reinforcement Learning with Hidden Data: Scaling to Multiple Robots” In Preparation.

3. Kenneth Bogert, Jonathan Feng-Shun Lin, Prashant Doshi, and Dana Kulic. ”Expectation-Maximization for Inverse Reinforcement Learning with Hidden Data” In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 1034-1042, 2016, Singapore.

4. Kenneth Bogert and Prashant Doshi. ”Toward Estimating Others Transition Models Under Occlusion for Multi-Robot IRL” In *Proceedings of the 24th International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 1867-1873, 2015, Buenos Aires, Argentina.

5. Kenneth Bogert, Sina Solaimanpour, Prashant Doshi. ”(Demonstration) Aerial Robotic Simulations for Evaluation of Multi-Agent Planning in GaTAC” In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 1919-1920, 2015, Istanbul, Turkey.

6. Kenneth Bogert and Prashant Doshi. ”(Extended abstract) Multi-Robot Inverse Reinforcement Learning under Occlusion with State Transition Estimation” In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent systems (AAMAS)*, pp. 1837-1838, 2015 Istanbul, Turkey.

7. Kenneth Bogert and Prashant Doshi. ”Multi-Robot Inverse Reinforcement Learning Under Occlusion with State Transition Estimation” *Workshop on Autonomous Robots and Multirobot Systems (ARMS)*, In the International Conference on Autonomous Agents and Multiagent Systems, 2015, Istanbul, Turkey.

8. Kenneth Bogert and Prashant Doshi. ”Multi-Robot Inverse Reinforcement Learning under Occlusion with Interactions” In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 173-180, 2014, Paris, France.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In many contexts an autonomous robot is likely to come across situations that its designer did not or could not anticipate. This problem is increasingly relevant as robots are forecast to be deployed in unpredictable or dynamic scenarios, for instance: disaster recovery, autonomous vehicles, or health care. In these uncontrolled or partially controlled scenarios a robot may be limited by:

- Situations the designer did not or could not anticipate
- Incomplete knowledge, for instance of the current state of the world due to sensor limitations
- Inaccurate actuators causing uncertainty in the robot's actions
- Presence of other intelligent agents that the robot must interact with in some way

Machine learning solution

Rather than attempt to pre-program robots with responses to all possible situations it might find itself in (an intractable approach by far) machine learning can be employed to extend the robot's useful range beyond what the designer anticipated. Examples include SLAM [56], which can be used in unknown environments to learn the map of an area while the robot is moving through it, and reinforcement learning [23] which learns the dynamics of uncertain actions, even when the state of the world is only partially observable [34]. This dissertation will address the problem of a robot learning from other agents in the environment. Our goal is produce a generally applicable machine learning technique which could be employed alongside other machine learning methods on autonomous robots to produce an intelligent agent capable of operating in its environment without the prior knowledge it needs to accomplish its tasks. We will first discuss the unique problems and challenges encountered by robots which motivate developing and deploying these machines as learning agents.

1.1 Motivation: Robots as Learning Agents

Though robots come in a vast variety of body types and capabilities all share in common the ability to effect a physical change in the world. In environments where actuators are imperfect the robot may be given access to sensors to provide information about the state of the world it needs to complete its task. In many scenarios the relevant state of the world is continuous, for example in navigation tasks the state is the robot's pose on a map. Actions available to the robot may also be continuous, such as the speed with which to turn its wheels.

In some cases it is beneficial to discretize the robot's state and action spaces. The cost of finding action plans in the full continuous space may be computationally prohibitive, for example, or the robot may lack resources with which to store and utilize the complete high-dimensional data. In this document we will consider only discrete states and actions unless otherwise noted.

1.1.1 Challenges of robotic domains

Noisy sensors

Autonomous robots may be expected to rely on their on-board sensors to produce information about the world. These sensors are often limited in the range, quality, and completeness of the information they produce. For example, laser scanners attached to a mobile robot can produce accurate distance and bearing measures of external objects but suffer from minimum and maximum range as well as occlusion by objects in the environment. This problem is particularly present with inexpensive robots such as the Turtlebot 2 (figure 1.1, used in our robotic experiments) due to the use of inexpensive, limited range sensors with many unexpected failure modes. Furthermore, if a robot is interested in knowing its pose on a map a laser scanner alone will likely not reveal such information un-ambiguously. While external references, such as GPS, could be queried to fill in the gaps of on-board sensors these suffer from communication issues that may be unacceptable for the robot's intended task, for instance during movement indoors or real time decision making.

Sensors, in fact, rarely produce noise-free data. Autonomous cars, for instance, cannot rely solely on GPS data to make second-to-second driving decisions due to the need to filter successive GPS readings which results in the car's position belief lagging behind the car's true position. When at speed such a situation is dangerous as the car may not respond correctly to external events such as the movement of other cars or pedestrians. Ideally, autonomous robots must fuse limited, low-latency on-board sensor data with available external data to make use of both sources of information effectively.

As a result of these issues, the true state of the world that a robot needs to complete its task may be hidden with only partial information about it available. In spite of this many other agents with similar or less sensor data (humans, for example) regularly accomplish the tasks we desire robots to do. This strongly indicates that robots have the necessary information needed to do the same but lack the intelligence, energy, or actuators needed.

Imperfect actuators

In most cases the actuators robots use to effect change do not perform perfectly. This may be due to the physical system it relies on being fundamentally stochastic or could arise due to wear or damage to the actuator. Examples include the wheels on a small, differential drive robot wearing down over time, or a legged robot slipping in mud.

As a result even if the robot were to be gifted with perfect knowledge of the state of the world it could not be certain as to what the outcome of its actions would be. By contrast, in contexts where the environment can be fully controlled and the robot's actuators perform relatively noise free the designer can confidently predict the range of states the robot might find itself in, for example an assembly line robot bolted in to place in a factory.

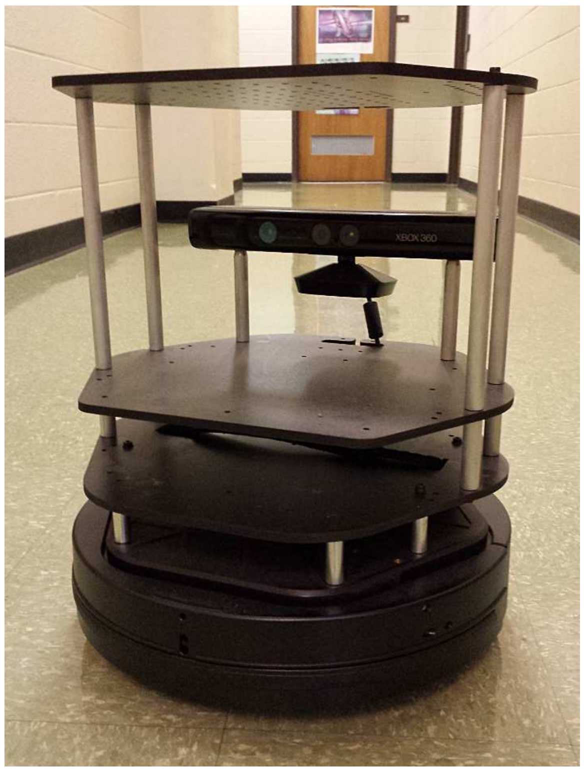


Figure 1.1: A Turtlebot 2, used in our robotic experiments. As an example of an inexpensive autonomous differential drive robot it exhibits noisy, limited sensors and imperfect actuators.

Unanticipated environments

Additionally, robots are often expected to perform in environments in which the designer did not or could not anticipate. Examples include household robots where the layout of the home it operates in is not known ahead of time or disaster robots entering areas inaccessible to humans. It is unreasonable to assume these issues could be solved by making every possible environment's data available to the robot due to the intractable amount of information it would have to store, unreliable access to large external data stores, or the authentic unknowns of an environment.

Presence of other agents

Lastly, a robot may operate in environments in which it will interact with other agents, for instance other robots, humans, or animals. Multi-agent scenarios greatly complicate planning and reduce the predictability of the environment, necessitating advanced techniques to allow the robot to accomplish its goals. This is especially the case when the goals include interaction with other agents as the possible results now depend upon the choices of the other agents which the robot cannot perfectly control. As a result, multi-agent solutions have significantly higher computational time and storage costs than single agent.

1.1.2 Machine learning with robots

Because of the above challenges it is desirable to endow robots with the ability to adapt to the situation they find themselves in. The field of machine learning provides a number of solutions to many of these issues. For example, a robot can employ SLAM [56] to autonomously learn the map of its environment while keeping itself localized and reinforcement learning [23] allows the robot to learn the mechanics of the world while attempting to reach its goals. However, these and most current methods are only applicable to the static elements of the environment.

1.2 Contributions of this Dissertation

Application of IRL to autonomous robots

Dynamic elements of an environment include non-agent elements which move such as flowing water or loose dirt. These elements are usually governed by static laws and therefore existing techniques can be adapted to learn the way in which they change, dynamic Bayesian networks can be employed to learn the behavior of weather events for example. Agents, however, present a much larger challenge in that they have the ability to select an action which then influences their dynamics. When attempting to predict or understand the agent's motion some explanation for the action choice is required or else the information learned about the agent is not generalizable to novel situations. In this dissertation we will model agents to be learned from (”experts”) as attempting to optimize their rewards in a Markov Decision Process (MDP), thus explaining their action choices.

To avoid having to provide the robot with one MDP for every agent it may encounter we wish to give the robot the ability to learn these MDPs by observing agent behavior. Inverse Reinforcement Learning (IRL) provides one such method, as given all components of a MDP except for the reward function (MDP/R) and observations of the agent's behavior IRL finds a reward function that results in a completed MDP whose optimal behavior matches observations. IRL makes a number of assumptions, however, that are difficult to satisfy in the challenging robotic domains previously described. Particularly, IRL assumes: the learner can obtain perfect observations of the expert's behavior, the learner has full knowledge of the expert's dynamics, the expert is perfectly rational and its states conform to the Markov assumption, and the expert behaves solely to optimize the rewards it receives from its MDP without distraction from other agents.

Relaxing assumptions of IRL

In this work we partially relax a number of these assumptions with the goal of utilizing IRL on-board an autonomous mobile robot to learn MDPs that accurately describe the agents it encounters in its environment. We focus first on the difficulty of the learning robot constantly observing the expert using its on-board sensors. Due to the limits of sensor range or objects in the environment the learner may fail to observe the expert during portions of its demonstrated behavior, and this failure may be persistent resulting in no information received for a specific subset of states the expert may traverse through. We call this style of observation failure *occlusion* and develop techniques to account for the missing data that results.

IRL assumes the expert is rational and, being an expert, has arrived at the optimal policy for the given MDP that describes it. Then, the behavior observed must be due to the expert trying to maximize its rewards over time, an assumption that may not be valid in the presence of multiple experts as some of their behavior may be due to interaction between them that is not captured by their MDPs. We develop a model of this *sparse interaction* and extend a well known IRL algorithm to account for this interaction behavior in order to prevent the interactions from becoming a source of persistent noise that biases the learned reward functions.

In real-world contexts the learner may not have complete knowledge of the expert's dynamics. This is represented in MDPs by the transition function $T()$ and describes the (potentially stochastic) changes in state influenced by the actions of the agent. This lack of knowledge could be due to many mundane causes, for instance, damaged or worn components on a robot may change its ability to move through a space and such effects would not plausibly be known ahead of time to the learner. Commonly a method such as estimating the parameters of a dynamic Bayesian network that describes the transition function would be employed to estimate the transition probabilities from the observed behavior. However, as the expert is acting optimally it is not expected to visit every state and perform every action in them, making learning the transition function from observations with a model free approach difficult. Worse, in the presence of occlusion some transitions critical to the expert's behavior may never be observed. We remedy this problem by developing a model based upon features or sub-components of the transitions, once learned this model rigorously projects the transition information from observed transitions to those that have not been seen, recovering a complete transition function.

Experimental validation of our methods

Finally, we evaluate the performance of our new techniques in a number of robotic domains, both real and simulated. Algorithms intended for deployment on real robots must work in the face of sensor noise and state uncertainty, imperfect actuators, and limits on available computation time or power. Many of our experiments require time-sensitive, accurate learning in order to successfully accomplish the task given to the learning robot. All computation is done on-board the robots using small laptop computers and our favorable results show the viability of our methods in these challenging domains.

1.3 Dissertation Organization

This dissertation is organized into chapters as follows:

Chapter 1	Introduction to the robotic domain, motivation and overview of the contributions of this work.
Chapter 2	Review of background material this work makes use of including probabilistic reasoning over time and graphical frameworks, Markov Decision Processes, overview of some Inverse Reinforcement Learning algorithms we extend, and numeric methods used to solve the problems in this dissertation.
Chapter 3	Review of related work in the fields of Inverse Reinforcement Learning, robotic patrolling methods, and learning transition probabilities from trajectories of agents. Comparisons are made to the contributions of this dissertation.
Chapter 4	Methods to compensate for hidden variables as a result of occlusion during IRL. Two methods are presented, a simple fix that ignores the missing data and an Expectation-Maximization approach that completes the missing data with its expectation.
Chapter 5	Method for estimating the transition function of an external agent from observed behavior under occlusion prior to performing IRL. Convex approximation for large problems given.
Chapter 6	Model and solution for performing IRL in the presence of multiple possibly interacting experts. Consideration is given for the case where the interaction behavior of the experts is unknown due to occlusion.
Chapter 7	Experimental evaluation of the methods in chapters 5 and 6 in a robotic patrolling domain.
Chapter 8	Conclusion, discussion of the work presented, and consideration of future work directions.

Additionally, extra details on our robotic experiments and proofs presented in this work are provided in Appendix A.

A sequential reading of this dissertation will provide: the motivation and background for using Inverse Reinforcement Learning on autonomous robots, an exploration of work related to this concept, followed by a number of chapters each focused on relaxing an assumption or requirement of IRL. Experiments and evaluation follow the methods section in chapter 4 while a number of other experiments involving the penetration of a perimeter patrol are grouped together in chapter 7. Lastly, we provide a short discussion of the work presented as well as future work directions.

Chapter 2

Background

The techniques developed in this work build upon existing Inverse Reinforcement Learning techniques. Here we review these precursor works as well as the probabilistic reasoning frameworks and optimal decision making theory they depend on. Applying IRL to robotic domains entails efficiently solving the problems presented, in most cases with a numerical function minimizer. For completeness we provide a short background on these algorithms and briefly consider their capabilities and requirements.

2.1 Probabilistic Reasoning over Time

2.1.1 Bayesian Networks

Bayesian networks are a graphical way of modeling the relationship between random variables. They are represented as directed-acyclic graphs (DAG) in which nodes may be observable values, hidden (latent) variables, or even hypotheses whose values are determined according to an associated probability distribution. The edges represent conditional dependencies between the nodes such that nodes which have no connection are conditionally independent of each other. We commonly use family language to describe the relations between nodes, for example B is a parent of A, A is a child of C, etc. We show an example Bayesian network in figure 2.1

Markov Blanket

Many algorithms exist for performing efficient inference in Bayesian networks in addition to learning both the structure and distribution parameters underlying the nodes from data. One property we make use of in this dissertation is the Markov Blanket. A node is conditionally independent of all other nodes in the network given its Markov Blanket which is defined as the node's parents, its children, and its children's parent nodes. Formally, we have: $Pr(X|MB(X), Y) = Pr(X|MB(X))$ where the set $MB(X)$ is all the nodes in X's Markov Blanket, we illustrate this concept in figure 2.2.

2.1.2 Dynamic Bayesian Networks

Bayesian networks can be extended to settings involving time by assuming that the structure of the network and node distributions remain fixed and "unrolling" the network out by timestep. These are referred to as dynamic Bayesian networks (DBN) and are illustrated in figure 2.3. We use

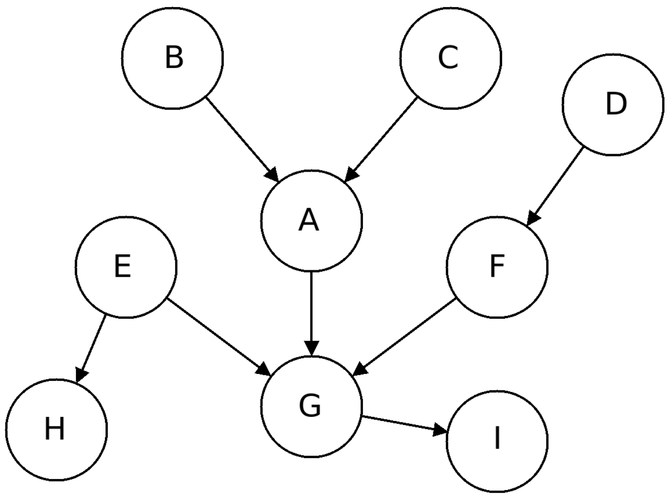


Figure 2.1: Example of a Bayesian network. Each node represents a random variable, edges indicate conditional dependencies, e.g. $Pr(A|B)$

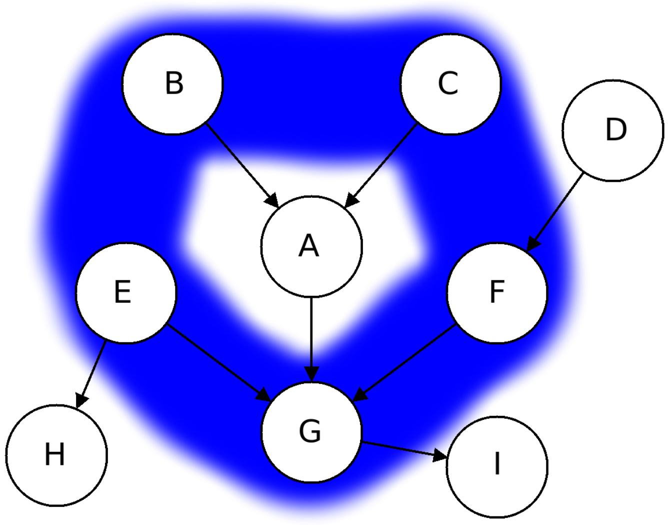


Figure 2.2: The Markov blanket of node A is all nodes in the blue area. A is conditionally independent of all other nodes given its Markov blanket, which is defined as its parents, its children, and its children's parents.

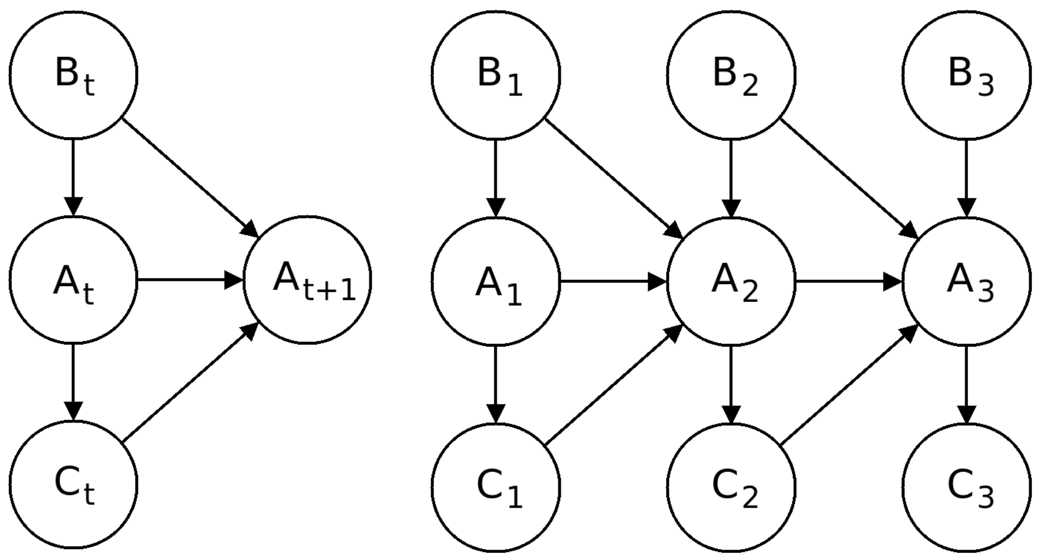


Figure 2.3: Example dynamic Bayesian network (left) and the DBN unrolled for three time steps (right)

a subscript on each node's variable to indicate the timestep relations between them, for instance $Pr(X_{t+1}|X_t)$ gives the probability of the X variable at the next timestep $(t+1)$ given its value at the current timestep (t).

Hidden Markov models

One common form of DBN is known as a hidden Markov model (HMM). In general, at each timestep in a HMM there is one latent variable representing the state of the system and one observed variable which depends only upon that state; we illustrate this design in figure 2.4. As each state variable depends only upon the state at t-1 this model exhibits the Markov property: all other previous states have no influence on its associated probability distribution.

2.1.3 Expectation-Maximization

Given a sequence of observed variables the joint state probabilities of a HMM may be calculated efficiently using the Baum-Welch algorithm, also called the forward-backward algorithm for its two main steps:

Forward Step:

$$f(Z_{t+1}|y_{1:t+1}) \quad \propto Pr(y_{t+1}|Z_{t+1}) \sum_{z_t} Pr(Z_{t+1}|y_t) f(z_t|y_{1:t}) \tag{2.1}$$

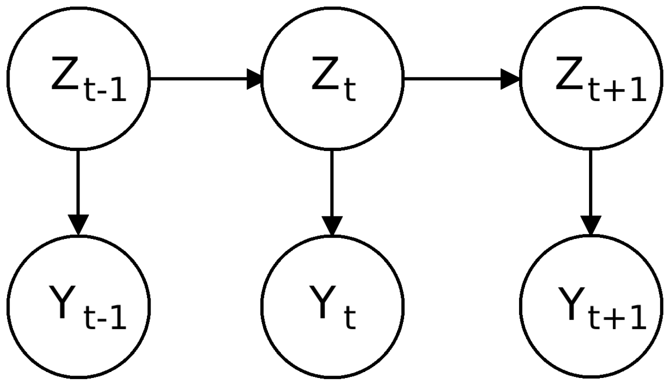


Figure 2.4: Example of a hidden Markov model (HMM) shown as an unrolled dynamic Bayesian network. Z nodes represent hidden state and Y nodes probabilistic observations received at each timestep.

Backward Step:

$$B(y_{t+1}|Z_t) \quad = \sum_{z_{t+1}} Pr(y_{t+1}|z_{t+1})B(y_{t+2}|z_{t+1})Pr(z_{t+1}|Z_t) \tag{2.2}$$

Here we use Y to denote the observed variables and Z as the hidden. As can be seen in the above equations the algorithm's steps are so named because it first moves forward in time applying a filtering step and then moves backward in time applying smoothing. This algorithm is a specialized form of the well known Expectation-Maximization (EM) algorithm and has a time complexity of O(N^2T) where N is the size of the state space and T is the number of timesteps. EM is an iterative method used to find the locally maximum likelihood parameters by taking the expectation over latent variables using the current set of parameters, then choosing parameters that maximize the log likelihood of the expectation.

E step:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) = \sum_{Y \in \mathcal{Y}} \tilde{Pr}(Y) \sum_{Z \in \mathbb{Z}} Pr(Z|Y; \boldsymbol{\theta}^{(t)}) \; log \; Pr(Y, Z; \boldsymbol{\theta}) \tag{2.3}$$

M step:

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\theta} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) \tag{2.4}$$

This is repeated until convergence ($\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)}$).

2.1.4 Monte-Carlo statistical methods

As the size of a DBN grows performing the E step may become prohibitively expensive due to the exponential growth of the size of $\mathbb{Z}$. An alternative is to perform Monte-Carlo integration to estimate the distribution over $\mathbb{Z}$ using samples. However, sampling directly from the distribution

$Pr(Z|Y)$ may be difficult; due to its size it may not be feasible to compute or represent the distribution exactly. When this method is used to estimate the E step in Expectation-Maximization the resulting algorithm is called Monte-Carlo Expectation-Maximization [33] (MCEM).

Gibbs sampling

Gibbs sampling provides a simple solution by iteratively sampling one node in Z each iteration to produce a Markov-chain of samples (each sample's probability depends only on its parent). Under mild assumptions the samples converge to the desired distribution as the number of samples increases.

Gibbs sampling is a Markov-Chain Monte-Carlo method for approximating the distribution of a Bayesian network with hidden variables [49]. It is a special case of the Metropolis-Hastings algorithm in which the probabilities of each individual node are known and can be sampled from, but the distribution over the entire network is intractable. Sampling proceeds by first randomly assigning all hidden nodes and then repeatedly sampling each hidden node conditioned upon the current value of all other nodes. This procedure generates a Markov chain of samples where each complete network generated depends only upon the previous one, and over time the sequence of networks approaches the true joint distribution desired.

2.2 Markov Decision Processes

Decision theoretic models treat agents differently from the simple state changes of a HMM in that agents have the ability to choose an action that influences their dynamics. While many such models exist we focus here on Markov Decision Processes [46] as MDPs are the framework used by Inverse Reinforcement Learning. Markov Decision Processes describe agents in scenarios where the agent has the ability to fully observe the state of the world. In situations with partial state observability, the agent is accurately described by a Partially Observable MDP (POMDP) [22]. Though they describe many robotic scenarios more accurately than MDPs, POMDP's will not be discussed at length here as they are not used in this work.

Formally a MDP is defined by the tuple $< S, A, T, R, \gamma >$ where S is a (possibly infinite) set of states (we will assume a finite set of discrete states from here forward), A is a set of actions (again, finite discrete), $T : S \times A \times S' \rightarrow [0, 1]$ is the transition function which returns the probability of the agent transitioning to state S' given it is in state S and performs action A, $R : S \times A \rightarrow \mathbb{R}$ is the reward received for being in state S and performing action A, and γ is the discount factor which describes the trade-off in value between immediate rewards and the expected future ones. The agent using an MDP solves it by finding an optimum policy $\pi^* : S \rightarrow A$ which, when followed, produces the maximum rewards over time. The optimum policy may be found by solving the Bellman equation:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')U(s') \tag{2.5}$$

and then greedily choosing the action whose expected utility is the highest for each state. The Bellman equation may be solved using a value iteration, a dynamic programming method:

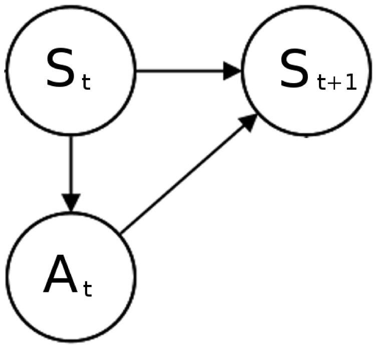


Figure 2.5: Two time step dynamic Bayesian network modeling the state transitions of a Markov Decision Process.

$$
\begin{aligned}
V^0(s) &= \max_a R(s,a) \\
V^k(s) &= \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V^{k-1}(s') \right)
\end{aligned}
\tag{2.6}
$$

This recursive function may be solved for H iterations, providing the expected value for all states looking ahead H timesteps (the horizon). It can also be iterated until the change in value of all states relative to the previous step is less than some tolerable error ϵ, which provides an infinite horizon guarantee to an error of ϵ of the expected value of each state.

MDPs require the use of state transition distributions that exhibit the Markov property, meaning that the probability of an outcome state is dependent only on the current state and action. Formally:

$$
Pr(S_{t+1}|S_{1:t}, A_{1:t}) \quad = Pr(S_{t+1}|S_t, A_t) \tag{2.7}
$$

These transitions are modeled as a simple 2 timestep DBN as shown in figure 2.5.

Though MDP's are not accurate models of many robotic domains they may still provide useful approximations as long as the uncertainty of the state is controlled by some other means. For example, the use of a localization system to provide the single most likely position a robot is currently in on a map allows an MDP's optimal policy to dictate the optimal action to perform for all possible positions provided the localization is reasonably certain (such a system does not provide a way of intelligently increasing the certainty, and likely will defer to pre-programmed recovery behaviors when its location is too uncertain).

In this work the expert agent is modeled as being a rational agent executing the optimum policy of an MDP, though perhaps imperfectly. A rational agent monotonically prefers policies that produce higher expected rewards. Given a complete MDP tuple, the reinforcement learning problem is to find the optimum policy by way of solving the value function, as described above. An agent may then execute the policy by performing the action specified by the policy at whichever state it finds the world in. This produces a trajectory $Traj = [< s, a, r >, < s, a, r >, ...]$ describing the sequence of states arrived at, actions chosen, and rewards received by the agent over time.

2.3 Inverse Reinforcement Learning

In Inverse Reinforcement Learning a learning agent desires to learn the motivations of another agent who is assumed to be an expert in the task it is performing. For example, a robot could desire the value an expert places on various parts of a task that the robot itself is expected to perform, an application called apprenticeship learning. By modeling the expert as executing a MDP, we can develop an inverse problem that supports this type of learning.

Assume that the learner has an incomplete MDP which describes the expert, specifically the learner is missing the expert's reward function ($< S, A, T, \gamma >$ are known). Now assume that the expert reveals to the learner the policy it is using, π^E. The learner's task becomes finding a reward function to complete the MDP under which π^E is the optimum policy, π^*. Unfortunately, this problem is ill-posed, there are an infinite number of reward functions which match this criteria for any given π^E.

Ng and Russell [41] provide an early solution to this issue by constructing a linear program that maximizes the margin of value between the action chosen by the expert in each state and all other actions. They further use a penalty term which encourages the use of smaller magnitude rewards. This Inverse Reinforcement Learning problem can then be solved as a linear program producing a reward value for each state.

While possible in some contexts, the requirement of receiving a single optimum policy from the expert may be difficult to meet. For instance the state space may be large or infinite putting a burden on the expert to enumerate the policy. Or there may be no communication between the learner and expert but the learner is capable of observing the expert's trajectory with the exception of the reward received. In other cases, there may be multiple agents providing expert policies which indicate different actions in some states. Finally, we note that Ng and Russell's [41] formulation does not allow for the rewards to depend on the action taken in a given state.

2.3.1 Feature Expectations

To solve these issues, we will restrict the reward function to be a linear combination of feature functions $\phi_k : S \times A \to \{0, 1\}$, $R(s, a) = \sum\limits_k \theta_k \phi_k(s, a)$ where θ_k is a real valued variable. We assume that the reward function the expert is using is of this form and all of the feature functions are known exactly to the learner. The learner's task becomes finding a vector of weights Θ that complete the reward function. To aid in the search we can calculate an expectation of features that any given policy would receive during its execution: $\Phi_\pi = \sum\limits_s \mu_\pi(s)\phi(s, \pi(s))$ where $\mu_\pi(s)$ is the state visitation frequency of state s, as given in equation 2.8 (μ_π^0 is the initial state distribution). Then we calculate the feature expectations of the expert given a policy or from an observed trajectory directly as: $\hat{\phi} = \frac{1}{|traj|} \sum\limits_{(s,a) \in traj} \phi_k(s, a)$. When we arrive at a set of weights which completes the reward function such that the resulting optimum policy's feature expectations match the expert's ($\Phi_\pi = \hat{\phi}$) we have solved the IRL problem. Additionally, we are able to calculate the weight gradient as $\nabla\Theta = \Phi_\pi - \hat{\phi}$ noting that if the optimum policy results in a feature expectation that is higher than the expert's we should reduce the corresponding weight, and vice versa. This formula is still not complete, however, as the set of weight vectors that produce the policy which matches this criteria is infinite, and depending upon the objective function used may never converge.

$$\mu_\pi(s') = \mu_\pi^0(s') + \gamma \sum_s T(s, \pi(s), s')\mu_\pi(s) \tag{2.8}$$

To make progress, Abbeel and Ng [3] follow a max-margin approach to maximize the value difference between the expert policy and all previously found policies arriving at the following quadratic program:

$$\begin{aligned} &\max_{t,\theta} \ t \\ &\textbf{subject to} \\ &\theta^T \hat{\phi} \geq \theta^T \Phi^{(j)} + t, \quad j = 0, \ldots, i-1 \\ &\parallel \theta \parallel_2 \ \leq \ 1 \end{aligned} \tag{2.9}$$

Where $\Phi^{(j)}$ is the feature expectations from the policy found at iteration j. This program can be solved with any quadratic solver, such as SVM, to obtain the weights. A convex combination of all produced policies is then generated, weighted such that the feature expectation of the distribution over these policies exactly matches the expert's feature expectations.

Abbeel and Ng also provide an approximate technique that projects the feature expectations from previous iterations along an orthogonal line with the expert's feature expectations iteratively to produce candidate feature weights for the next iteration.

2.4 Principle of Maximum Entropy

The principle of maximum entropy, while existing in various forms since at least the early 1900's, was formalized by Jaynes [21] in 1957 when he argued that the entropy in information theory and entropy in statistical mechanics are principally the same. The principle concerns the choice of a probability distribution constrained to match feature expectations with some observed data. While there are usually infinite probability distributions that match a given set of these constraints, only one of them will have the maximum entropy. This one distribution should be chosen as it makes no additional assumptions other than what is required to satisfy the constraints. In other words, since it has the maximum entropy possible this distribution contains the least amount of information and can be viewed as an encoding of the information provided in the constraints and nothing else.

When a distribution is constrained to match observed feature expectations a non-linear, convex program to find the distribution with maximum entropy is given by:

$$\begin{aligned} &\max_{\Delta} \left(- \sum_{Y \in \mathbb{Y}} Pr(Y) \ log \ Pr(Y) \right) \\ &\textbf{subject to} \quad \sum_{Y \in \mathbb{Y}} Pr(Y) = 1 \\ &\sum_{Y \in \mathbb{Y}} Pr(Y) \ f_k(Y) = \sum_{Y \in \mathbb{Y}} \tilde{Pr}(Y) \ f_k(Y) \qquad \forall k \end{aligned} \tag{2.10}$$

Where $f_k(Y)$ is a function that returns feature k of Y and $\tilde{Pr}(Y)$ is the empirical distribution over Y.

2.4.1 Convex Programs

Maximum entropy programs of the type above are in a class of programs called *convex*. To be convex, its objective function must be a convex function, inequality constraints must be convex functions of relevant variables, and equality constraints must be linear in the relevant variables [15].

A convex function is one in which its epigraph (the set of points $\geq$ the points of the function) is a convex set. A convex set is one in which all the points of a line segment drawn between any two points within the set must also be in the set. Broadly speaking, a set which contains "indents" or interior holes that are not part of the set are not convex.

2.4.2 Principle of Latent Maximum Entropy

Wang et. al. [59] introduce the principle of Latent Maximum Entropy to extend the principle of maximum entropy to problems with hidden variables. The program in equation 2.10 is modified to take into account an expectation over the missing data in the primary constraint:

$$\begin{gathered}\max_{\Delta}\left(-\sum_{Y\in\mathbb{Y}} Pr(X)\ log\ Pr(X)\right)\\ \textbf{subject to}\quad \sum_{X\in\mathbb{X}} Pr(X)=1\\ \sum_{X\in\mathbb{X}} Pr(X)\ f_k(X)=\sum_{Y\in\mathbb{Y}}\tilde{Pr}(Y)\sum_{Z\in\mathbb{Z}} Pr(Z|Y)\ f_k(X)\qquad\forall k\end{gathered}\tag{2.11}$$

Here, $X = Y \cup Z$. This new program is non-convex due to the presence of $Pr(Z|Y)$; an approximate solution in which $Pr(X)$ is assumed to be log-linear is provided which leads to the Lagrangian dual:

$$\mathcal{L}^{\text{dual}}(\boldsymbol{\theta}) \approx log\ \Xi(\boldsymbol{\theta}) - \sum_k \theta_k \sum_{Y\in\mathbb{Y}} \tilde{Pr}(Y) \sum_{Z\in\mathbb{Z}} Pr(Z|Y) f_k(X) \tag{2.12}$$

Where $\Xi(\boldsymbol{\theta})$ is the partition function.

2.5 Maximum Entropy IRL

We noted in section 2.3 that the constraint $\phi_\pi = \hat{\phi}$ is not sufficient to ensure convergence onto a single set of weights. Abbeel and Ng's [3] max-margin assumption may not be accurate for all scenarios where IRL could be applied. This could result in an inaccurate set of weights found which negatively impacts the generalizability of the algorithm. In fact, any assumptions made about the structure of the rewards beyond the feature expectation constraint will distort the results in some way. To overcome this issue Ziebart et. al. [61] propose a solution using the principle of maximum entropy.

As first formulated in Ziebart et. al. [61], MaxEnt IRL forms a probability distribution over all possible expert trajectories. Formally, we have:

$$\begin{aligned}
&\max_{\Delta} \left(-\sum_{Y \in \mathbb{Y}} Pr(Y)\ log\ Pr(Y) \right) \\
&\textbf{subject to} \quad \sum_{Y \in \mathbb{Y}} Pr(Y) = 1 \\
&\sum_{Y \in \mathbb{Y}} Pr(Y) \sum_{\langle s,a \rangle \in Y} \phi_k(s,a) = \hat{\phi}_k \qquad \forall k
\end{aligned} \tag{2.13}$$

Here, Δ is the space of all distributions $Pr(Y)$.

We may apply Lagrangian relaxation bringing both the constraints into the objective function and then solving the dual. The relaxed objective function becomes:

$$\begin{aligned}
\mathcal{L}(Pr, \boldsymbol{\theta}, \eta) = &-\sum_{Y \in \mathbb{Y}} Pr(Y)\ logPr(Y) \\
&+\eta \left(\sum_{Y \in \mathbb{Y}} Pr(Y) - 1 \right) \\
&+\sum_k \theta_k \left(\sum_{Y \in \mathbb{Y}} Pr(Y) \sum_{\langle s,a \rangle \in Y} \phi_k(s,a) - \hat{\phi}_k \right)
\end{aligned} \tag{2.14}$$

As equation 2.14 is convex taking the derivative with respect to $Pr(Y)$ and setting it to zero gives us the optimum:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial Pr(Y)} &= -log\ Pr(Y) - 1 + \sum_k \theta_k \sum_{\langle s,a \rangle \in Y} \phi_k(s,a) + \eta = 0 \\
Pr(Y) &= \frac{e^{\sum_k \theta_k \sum_{\langle s,a \rangle \in Y} \phi_k(s,a)}}{\Xi(\boldsymbol{\theta})}
\end{aligned} \tag{2.15}$$

where $\Xi(\boldsymbol{\theta})$ is the normalization constant $e^{\eta} e^{-1}$; as such η may easily be obtained. Plugging equation 2.15 back into the Lagrangian (equation 2.14), we arrive at the dual $\mathcal{L}^{\text{dual}}(\boldsymbol{\theta})$:

$$\mathcal{L}^{\text{dual}}(\boldsymbol{\theta}) = log\ \Xi(\boldsymbol{\theta}) - \sum_k \theta_k \hat{\phi}_k \tag{2.16}$$

With gradient:

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

This technique is based upon the observation that the probability of the expert following a given trajectory is proportional to the rewards received along it. This is defined exactly for deterministic MDPs and an approximation is provided in the case of stochastic transition functions in equation 2.17. Though this behavior would indicate the expert is sub-rational, we interpret this distribution as modeling a rational agent's behavior perturbed by spurious noise, perhaps due to un-modeled effects in the environment. An example would be robots forced to deviate from their optimum behavior due to the presence of humans blocking their way.

$$Pr(Y) \approx \frac{\prod\limits_{<s,a,s'>\in Y} T(s,a,s') e^{\sum\limits_k \theta_k \sum\limits_{\langle s,a\rangle \in Y} \phi_k(s,a)}}{\Xi(\boldsymbol{\theta})} \tag{2.17}$$

Later formulas, such as those found in Boularias et. al. [14], define the maximum entropy distribution over policies. We note here that given sufficient trajectories supplied by the expert this formula is approximately correct. However, it violates the correct from of a maximum entropy problem as the empirical data should be provided in the form of a probability distribution that the maximum entropy distribution is constrained to match.

$$\begin{aligned}
&\max_{\Delta} \left(-\sum_{\pi\in\Pi} Pr(\pi)\ logPr(\pi) \right) \\
&\text{subject to} \\
&\sum_{\pi\in\Pi} Pr(\pi) = 1 \\
&\sum_{\pi\in\Pi} Pr(\pi) \sum_{s\in S} \mu_\pi(s)\phi_k(s,\pi(s)) = \hat{\phi}_k \qquad \forall k
\end{aligned} \tag{2.18}$$

Approximations and Efficient Solutions

These problems may be solved with gradient descent techniques, for instance in Ziebart et. al. [61] Exponentiated Gradient Descent [27] is used. A common issue that appears during numerical optimization is the large size of the probability distributions which renders the problem intractable for large state/action spaces or long trajectories. Numerous solutions exist to address this issue and we briefly describe those used in this dissertation and relevant background works.

- Calculating the state visitation frequency directly using a forward-backward algorithm and replacing $\triangledown\theta_k = \sum\limits_{Traj} Pr(Traj|\theta)\phi_k(Traj) - \hat{\phi}_k$ with $\sum\limits_s \mu(s)\phi_k(s) - \hat{\phi}_k$. Ziebart et. al. [61] introduces this method to avoid calculating the large probability distribution over all possible trajectories taxis may take in a city. We use this approach as part of the *Hidden Data EM* algorithm in chapter 4.

- Calculating a measure of the similarity between the expert's trajectory and a given policy as $\sum\limits_{(s,a)\in Traj} Q^\pi(s,a) - V^\pi(s)$. This approach is used in the IRL* algorithm presented in chapter 4 and we utilize it in the experiments in chapter 6 (see section 6.2 for more details). This approach was first described in Ng and Russell [41].

- Reducing the distribution space by only considering policies near the optimum for the current set of weights rather than every possible policy. Boularias et. al. [14] describes this technique and suggests using all policies that are one action away from the optimum.

2.6 Evaluation of IRL

Evaluation of the performance of an IRL method is somewhat difficult. Direct comparison of the learned reward function to the true reward function is ill-advised as two very different reward functions may produce the same optimum policy and different IRL techniques may return different rewards for the same behavior. If the true optimum policy of the expert is known then one evaluation method is to calculate the proportion of states in which the optimum policy for the learned reward function gives the correct action. We label this technique **Learned Behavior Accuracy**.

2.6.1 Inverse Learning Error

However, evaluation with **Learned Behavior Accuracy** risks overemphasizing incorrect actions in unimportant states. If the expert's true reward function can be obtained for evaluation purposes we may utilize a metric presented by Choi and Kim [18] which compares the value function of the policy obtained from optimally solving the expert's MDP with the true reward function with the value of policy that is found by using the learned reward function. In other words, let π^L be the policy obtained by optimally solving the MDP using the reward function learned by the learner and π^* be the optimal policy of the expert obtained utilizing the true rewards. Then, the *inverse learning error (ILE)* metric is calculated as:

$$\mathsf{ILE} = ||V^{\pi^*} - V^{\pi^L}||_1$$

where V^{π^*} is the optimal value function of the expert's MDP and V^{π^L} is the value function due to utilizing policy π^L on the expert's MDP. In the event that the learned reward function results in a policy identical to the expert's actual optimal policy, $\pi^* = \pi^L$ and ILE will be 0; ILE monotonically increases as the two diverge. Furthermore, use of the difference in values also allows the error to grow proportionately to the value of the states where the learned policy diverges thereby placing emphasis on higher-valued states.

As an example, we show an evaluation of IRL^* in randomly generated MDPs. We generate MDPs with six states, four actions, and random transition and reward functions. We optimally solve each one using value iteration and repeatedly sample trajectories of length 10 timesteps using the optimal policy. As more trajectories are provided we expect the accuracy of the learning (LBA) to increase while ILE to approach zero. As can be seen in figure 2.6 this is indeed the case. Each data point is the average of averages from the output of 1000 trials grouped into blocks of size 10. Standard error is from 0.018 (1 trajectory, ILE) to 0.0021 (100 trajectories, ILE).

Notice the large standard deviation with fewer provided trajectories of ILE compared to LBA. This indicates the improved sensitivity to the true reward function that ILE offers: LBA simply counts the number of correct actions in the learned policy and as a result two different learned policies can produce the same LBA but will be unlikely to have the same ILE. In fact, this will only be the case when two differently learned states have the same optimum value.

2.7 Learning transition functions

Given that IRL algorithms are commonly given expert trajectories, if the learner does not have the expert's transition function then it stands to reason that the learner can estimate the transition function from the transitions within the trajectories. As the transitions of a MDP are describable

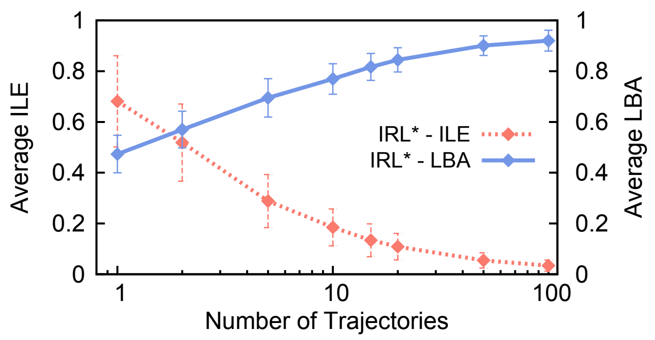


Figure 2.6: Average Inverse Learning Error (ILE) and Learned Behavior Accuracy (LBA) of IRL* as the number of provided trajectories increases. Random MDPs with 6 states and 4 actions, trajectories of length 10. Error bars are one standard deviation, standard error is between 0.018 (1 trajectory, ILE) to 0.0021 (100 trajectories, ILE)

with a simple three node DBN (see figure 2.3) the parameters of this transition DBN may be learned by employing a Maximum a Posteriori approach:

$$\begin{aligned} \theta_{MAP} &= \arg\max_{\theta} Pr(\theta|Trajectory) \\ &= \arg\max_{\theta} Pr(Trajectory|\theta)Pr(\theta) \\ &= \arg\max_{\theta} \prod_{s,a,s' \in Trajectory} Pr(s'|s,a;\theta)Pr(\theta) \end{aligned} \tag{2.19}$$

This approach requires the expert to cover a substantial portion of the state space during its demonstration and, given it is assumed the expert is optimal, this may be difficult to satisfy. Particularly, the expert will not perform all actions with the same frequency, otherwise it would not be optimal, increasing the amount of observations needed to adequately learn the transition DBN's parameters.

Alternatively, a Dirichlet distribution (see chapter 49 [7]), parameterized with transition counts from observations, could be used to form a distribution over possible transition DBNs and this distribution sampled (or the expectation taken) to arrive at candidate transition DBN for use with IRL.

In the event that some transitions are missing due to spurious data loss, Expectation-Maximization as described in section 2.1.3 may be employed to complete the missing data with an expectation given the current parameters. This algorithm is iterated to convergence at a local extrema to arrive at a likely parameterization for the transition DBN.

2.8 Numerical optimization

In certain cases constrained non-linear programs may be solved analytically, however, this will not be the case in general and numerical solvers are most often employed to produce an answer. A common approach is to perform Lagrangian relaxation to arrive at a single, unconstrained function with added Lagrange multiplier variables. This function may then be minimized using a number of function minimizer algorithms, we review those used in this dissertation below.

2.8.1 Nelder-Mead

The Nelder-Mead or downhill simplex method was proposed in [39] and uses a number of heuristics to move a simplex, a shape containing D+1 points where D is the number of dimensions of the problem, through the solution space until a minima is found. The points of the simplex are used to estimate the gradient relative to each other and as a result this algorithm does not require the gradient to be specified. Many variations exist but common implementations may be thought of as moving the simplex proportionally to the estimated gradient thereby approximating steepest gradient descent. Proof of convergence of Nelder-Mead is an open problem but in certain limited cases it has been shown to converge [32, 29].

2.8.2 L-BFGS

The limited memory Broyden-Fletcher-Goldfarb-Shanno [30] algorithm is a quasi-Newton line search method in which the Hessian of the problem is estimated through repeated gradient evaluations. The limited memory variant uses a linear amount of space to approximate the Hessian improving performance with problems of many variables. Line search quasi-Newton methods iteratively project a line based upon the gradient and Hessian approximations at a given point to estimate where the zero point of a function may be found and as a result are expected to converge faster than gradient descent methods. BFGS is known to converge to the global optimum if the function being optimized is strictly convex [43].

2.8.3 Gradient Descent

Gradient descent and its online variant stochastic gradient descent (SGD) are common numerical solving techniques in which the parameters of the problem are updated in proportion to the gradient until a minima is reached. This algorithm is simply:

$$w_{t+1} = w_t - \nu \nabla F(w_t) \tag{2.20}$$

With ν as the learning rate.

SGD is an approximation of gradient descent useful when the gradient is decomposable into a sum of parts:

$$F(w_t) = \sum_i F_i(w_t) \tag{2.21}$$

SGD iterates through the individual parts and updates its parameters after each one. Though the parameter updates may be chaotic this algorithm has an advantage in that the entire gradient does not have to be re-calculated with each parameter update. It therefore offers improved expected performance for large data sets where calculating the full gradient would be prohibitively expensive.

2.8.4 Exponentiated Gradient Descent

Exponentiated gradient descent [27] is a variant of standard gradient descent in which the parameter updates are exponential in the gradient:

$$w_{t+1} = w_t \times e^{\nu \nabla F(w_t)} \tag{2.22}$$

This algorithm offers improved performance over standard gradient descent in the presence of noisy data and has a lower theoretical loss function. One drawback is the parameters are restricted to the normalized plane and the components of $\nabla F(w_t)$ to $[-1, 1]$, though some solutions to alleviate the parameter restriction are offered. This algorithm is oriented towards online stochastic gradient descent where F_i is equal to the difference in binary feature values between the parameterized model and the provided data's at timestep i.

2.8.5 Improved Adaptive Exponentiated Gradient Descent

For sparse data sets having a single learning rate may cause over emphasis on data points that are more common at the expense of the sparse points, slowing convergence. To remedy this, adaptive gradient descent algorithms adjust the learning rate per weight in response to the associated error. In [54] a version of this method is developed for exponentiated gradient descent that offers improved worse case loss. A simple version of its update procedure (with path-length bounds) is as follows:

$$\begin{aligned} w_t &= e^{\beta_t - \nu \nabla F(w_{t-1})} \\ \beta_{t+1} &= \beta_t - \nu \nabla F(w_t) - \nu^2 (\nabla F(w_t) - \nabla F(w_{t-1}))^2 \end{aligned} \tag{2.23}$$

This algorithm allows parameters in the range $[0, \inf]$, with a simple transformation this can be converted to the range $[-\inf, \inf]$.

2.8.6 Primal-Dual Interior Point

For constrained, non-linear but convex problems interior point methods offer fast, scalable solutions over methods developed for general non-linear programs (such as sequential quadratic programming [12]). Numerous algorithms in this class exist, one of the most successful is the primal-dual method [12] (also called central-path algorithms) in which the gradient of a barrier function based upon the primal problem is found by utilizing the dual problem. This allows for polynomial time complexity.

2.8.7 Penalty Method

One possible way of solving non-convex programs is to treat the Langrangian multipliers as penalty terms. These terms are initialized to small, positive values and gradually increased each iteration of the algorithm. With the multipliers fixed the Lagrangian becomes a function of just the primal variables, which can be minimized using any function minimizer technique. After a solution is found, this solution becomes the starting point for the next iteration. This continues until the multipliers go to infinity, at which point only feasible solutions will have non-infinite values.

2.9 Notation and Terminology

To aid the reader we provide a list of common term definitions here as well as notation for reading equations found in this dissertation.

Variables

- **Random variables:** A, B, C
- **Variable values:** a, b, c or $a_1, a_2, \ldots a_n$
- **Sets:** $\{A, B, C\}$
- **Vectors:** $A_{1:n} = (A_1, A_2, ...A_n)$
- **Observed variables and values:** Y, y_n
- **Hidden variables and values:** Z, z_n
- **Full data sets:** $X = Y \cup Z$

Probability Distributions

- **Probability distribution:** $Pr(A) = Pr(A = a)$
- **Parameterized probability distribution:** $Pr(A; \theta) = Pr(A = a; \theta)$
- **Conditional probability distribution:** $Pr(A|B) = Pr(A = a|B = b)$
- **Empirical probability distribution:** $\tilde{P}r(A)$

Markov Decision Processes

- **State space and state:** S, s
- **Action space and action:** A, a
- **Policy:** π
- **Value function:** $V(s)$
- **Q-Value function:** $Q(s, a)$
- **State visitation frequency:** $\mu_\pi(s)$
- **State occupancy distribution at timestep t:** $\mu_t(S)$
- **Trajectory:** $Trajectory = Traj = \{< s_1, a_1 >, < s_2, a_2 >, \ldots < s_T, \emptyset >\}$

Inverse Reinforcement Learning

- **Expert agent:** I, J
- **Learner agent:** L
- **Features:** $\phi = (\phi_1, \phi_2, ...\phi_k)$
- **Feature expectations:** $\Phi = (\Phi_1, \Phi_2, ...\Phi_k)$
- **Expert feature expectations:** $\hat{\phi} = (\hat{\phi_1}, \hat{\phi_2}, ...\hat{\phi_k})$

2.10 Summary

In this chapter we provided a brief overview of the foundation concepts and frameworks used in our work. Inverse Reinforcement Learning is based around the Markov Decision Process optimal control framework which is itself justified by probabilistic reasoning over time methods and decision theory. Our development of Inverse Reinforcement Learning techniques for use on autonomous robots extends Maximum Entropy IRL in various ways detailed in the following chapters. However, the IRL methods presented in this chapter are by no means the entirety of those available and we explore some of them in more detail in chapter 3.

To perform and evaluate our robotic experiments we defined evaluation techniques appropriate for IRL and briefly reviewed the numerical optimization methods used in this work. Other techniques reviewed in this chapter include Expectation-Maximization and learning dynamic Bayesian networks from data, both of which are utilized in chapter 5.

Chapter 3

Related Work

In this chapter we review and discuss work related to performing Inverse Reinforcement Learning in robotic environments, particularly other methods of IRL not covered in the background. Additionally, we discuss other applications of IRL as well as other work related to patrolling robots similar to those used in our experiments and contrast them with our applications.

3.1 Inverse Reinforcement Learning

We view IRL as a way of performing inverse optimal control [44] with MDPs as its framework. Inverse optimal control includes other frameworks besides MDPs such as linear quadratic regulators [60, 35]. Linear quadratic regulators (LQR) are optimal control frameworks in which the system dynamics take the form of linear differential equations and the cost (or reward) function is quadratic. LQR's may in many cases be solved exactly in closed form to produce a controller which naturally handles continuous state spaces. This property makes them attractive for many robotic optimal control problems, such as flying a helicopter (Abbeel et. al. [2]).

As we described in more detail in section 2.3, in this work we extend Maximum Entropy IRL of Ziebart et. al. [61] which, similarly to the Max Margin algorithm from Abbeel and Ng [3], models the reward function of the expert as a linear combination of binary feature functions. The weights are manipulated until the feature expectations of the optimum policy (or distribution over trajectories) matches the feature expectations of the demonstration.

Linear programming approach

Ng and Russell [41] introduced IRL in terms of matching the expert's provided policy to the optimum policy produced by a candidate reward function and provide the following linear program. To solve the reward function degeneracy issue it chooses a reward function that maximizes the difference in value between the optimal and second most optimal policies.

$$\begin{aligned}
&max \sum_{i=1}^{N} \min_{a \, \in \, A_{/a^\pi}} \big((P_{a^\pi}(i) - P_a(i))(I - \gamma P_{a^\pi})^{-1}R\big) - \lambda ||R||_1 \\
&st. \\
&(P_{a^\pi}(i) - P_a(i))(I - \gamma P_{a^\pi})^{-1}R \succeq 0 \quad \forall a \in A_{/a^\pi} \\
&|R_i| \leq R_{max}, \quad i = 1, \ldots, N
\end{aligned} \tag{3.1}$$

Where P_a is the transition matrix for action a, a^π is the action taken by the expert in state number i, λ is a penalty coefficient, and R_{max} is a parameter specifying the maximum allowable magnitude of the reward vector's elements.

Bayesian IRL

Other IRL methods include a Bayesian approach from Ramachandran [47] which forms a probability distribution over the space of all reward functions. The probability of the state-action pairs in the observed expert's behavior is modeled with an exponential distribution:

$$Pr(s_i, a_i|R) \;\; = \tfrac{1}{Z} e^{\alpha Q(s_i, a_i, R)} \tag{3.2}$$

Where α is a parameter indicating the rationality of the expert. The distribution over R may be found through maximum a posteriori, a common method for Bayesian estimation problems.

An optimal policy can then found which minimizes the expected policy loss over the expectation of the distribution of R, i.e. the mean reward function. However, the posterior distribution over R may be complex with analytical derivation difficult, so a Monte-Carlo Markov chain sampling algorithm is provided to sample from the posterior by performing a random walk through neighbors of a given reward function (discretized into a grid). This technique contrasts with the Maximum Entropy IRL technique wherein a single reward function is found that parameterizes a distribution over all trajectories (or policies). As formulated Bayesian IRL does not use a linear combination of feature functions for $R()$.

Game-theoretic approach

A game-theoretic approach to IRL from Syed and Schapire [55] produces the MWAL algorithm, which models the process of IRL as a zero-sum game with a min player that chooses reward weights θ and a max player that chooses a mixture of policies ψ. By solving the game the following quantity is found:

$$v^* = \max_{\psi \in \Psi} \; \min_{\theta} \; [\theta * \Phi_\psi - \theta * \hat{\phi}] \tag{3.3}$$

where Φ_ψ is the feature expectations of the policy mixture and $\hat{\phi}$ is the feature expectations of the expert. Here, the optimum ψ maximizes the difference between the value functions $V_\psi - V_{\hat{\phi}}$ under the worst case weights. Notice this is similar to the minimizing $|Q - V|$ approach in section 2.5.

IRL with POMDPs

Domains in which the agent does not have the ability to fully observe the state of the world are properly described by Partially Observable MDPs (POMDP). Many robotic domains, such as navigation, fall into this domain and can only be approximated with MDPs. Inverse Reinforcement Learning with POMDPs was addressed in Choi and Kim [19] and faces a difficulty as the expert is acting based on its internal belief as to the state of the world which must be shared with the learner. Alternatively, the expert's complete optimal finite state machine policy may be shared instead. In our work, by contrast, the expert agent has full observability over its state even though the learning agent may not, making it appropriate for the learner to model the expert as an MDP.

Continuous space IRL

Performing IRL in continuous space is considered an open problem but there have been a few papers that address this challenge. Some, such as relative entropy IRL from Boularias et. al. [13], simply utilize straightforward integration in their equations with discrete features and discrete time. For continuous time, space, and actions the use of path integrals allows for a specific class of these systems to be solved with Maximum Entropy IRL as shown in Aghasadeghi and Bretl [4]. Here the policies and rewards are linear combinations of basis functions and the expert provides the optimal set of weights for the true policy (possibly by demonstration). For performance reasons we use only discrete states and actions in our experiments, and discuss the specifics per experiment.

Active Learning

Most IRL formulations, including those used in this dissertation, assume no interaction between the expert and learner. However, IRL with active learning as defined in Lopes et. al. [31] extends Bayesian IRL to allow the learner to make requests for demonstrations from the expert for states it is interested in. This is accomplished by discretizing the distribution over rewards and calculating the average entropy per state, then querying the expert for a demonstration of the state with the maximum average entropy.

3.2 Applications of Inverse Reinforcement Learning

Apprenticeship Learning

One of the first and most well-known applications of IRL is apprenticeship learning (Abbeel and Ng [3]) in which the learned reward function is transferred from the expert's MDP to another MDP/R model which describes the learning agent. Once solved, the learner's MDP allows it to accomplish the same task as the expert, but not necessarily in the same way. Apprenticeship learning is a generalized form of imitation learning (Schaal [51]) that avoids issues of action mapping due to differing body types and capabilities between the learner and expert, for example when a non-humanoid robot learns from a human, by performing any needed translation in the reward function. Many examples exist in the literature, such as learning the type of grasp to use while sorting objects (Bogert et. al. [11]), learning to drive a simulated car (Abbeel and Ng [3]) or sail a simulated boat (Neu [40]), learning to fly a helicopter (Abbeel et. al. [1]), robot path planning that can mimic teachers (Ratliff et. al. [48]), and recently learning to grasp unknown objects by their handle (Boularias et. al. [14]), play a simple ball-in-cup children's motor game (Boularias et. al. [13]), and learning to play games such as table tennis as well as an expert (Muelling et. al. [36]).

Prediction of expert motion

Because IRL completes the MDP/R model of the expert another useful application is to use the completed MDP for predicting the actions of the expert in states that have not been observed, for instance to determine the future locations of patrolling robots so they may be avoided (Bogert and Doshi [9, 10]). The learned reward function may also be used to provide predictions of the expert's behavior in new, unseen scenarios by transferring it into secondary MDP/R models that describe the expert in these new scenarios. Examples include a learning a drivers' behavior to better optimize fuel efficiency of a hybrid car (Vogel et. al. [58]), predicting driver behavior on residential roads (Shimosaka et. al. [52]), and predicting the future path of humans so that, for example, path planning by a robot can avoid collisions with them (Kitani et. al. [26], Ratliff et. al. [48], and Kim and Pineau [24]).

Others

Other applications of IRL include learning the behavior of the expert so that the learner may interact with them, for example in spoken dialog (Kim et. al. [25]) where the rules of taking turns speaking are learned. IRL has also been used to complete models in which agent decision-making is a component and important data is missing, such as the routes taken by a car when only sparse GPS point-measurements are available (Osogami and Raymond [45]).

3.3 Multiple Interacting Robots

IRL lends itself naturally to robotic learning from demonstration scenarios in which there is a single human or robot expert in a controlled environment. However, our scenario in chapter 6 requires that the learner simultaneously learn from multiple, possible interacting robots. Prior work on multiple robots learning from demonstration either as the learners or as the experts is sparse. One such domain is the scenario where a single expert teaches a team of robots (Chernova and Veloso [17]). Here, the learners consider each others' actions to facilitate coordination while the expert acts alone. Another multi-robot learning from demonstration scenario places a robot as an expert and another as the learner (Alissandrakis et. al. [6]) in an imitation learning setup. In Natarajan et. al. [37] multiple agents with non-sparse interactions are modeled with a single joint-MDP and IRL performed on this MDP. This model may not be used in our domain due to the presence of occlusion; if one agent were observed and other not the state of the joint MDP becomes partially observed, breaking an important assumption of IRL.

Robot Interaction modeling

Modeling of robot interactions has received attention recently with Spaan and Melo [53] utilizing game theory to model interactions as a component of multi-agent planning. Analogous to our setting, two robots plan their trajectories using individual MDPs and overlay it with a Markov game when conflicts arise such as when they must cross through a narrow corridor. This is a multi-agent planning problem, not inverse learning; our work in chapter 6 may be viewed as the inverse of this problem where we attempt to learn which Nash equilibrium the two interacting robots have chosen based on their observed actions. Valtazanos and Ramamoorthy [57] use demonstrations to learn a set of interaction actions, which may then be used by the learner robot when interacting with another robot. This allows it to predict the state that may result due to its actions, and utilize these to shape the actions of a second robot as a response to the learner's actions. Our work differs in that we focus on entire trajectories where the interaction is a critical part of the perceived motion, the learner cannot influence the actions of the experts in any way, and we seek to learn the behavior of multiple robots when they are not interacting as well.

When interactions between two agents are pervasive and extended the problem becomes a joint decentralized MDP (Goldman and Zilberstein [20]) whose solution is highly intractable. We instead focus our scenario on settings where interactions between robots are sparse and scattered; this allows observed robots to be individually modeled as tractable MDPs.

Patrolling robots

Finally, in the context of our multi-robot patrolling application domain, related research has predominantly focused on generating robot patrolling trajectories that are theoretically difficult to learn from observations by relying on randomized behavior (Agmon et. al. [5]). This work is motivated by building robots that could be deployed for perimeter patrolling; by contrast our application seeks to use Inverse Reinforcement Learning to build robots that may learn *simple* patrolling behavior of multiple robots, which may be viewed as the first steps towards an inverse optimal patrolling problem.

3.4 Learning Transition Probabilities of an External Agent

Few approaches investigate relaxing IRL's requirement of full knowledge of the expert's transition function. Boularias et. al. [13] propose model-free IRL with a single expert, learning the reward function by minimizing the relative entropy between distributions over trajectories generated by a baseline and target policies. Klein et. al. [28] propose a method to avoid solving the forward reinforcement learning problem by instead estimating the Q function from the expert's feature expectations directly and using this as a score function for a linearly parameterized classifier that outputs the rewards. In contrast, in chapter 5 we explicitly first learn the transition function under occlusion making this a first semi-model based method.

Bard [8] used maximum entropy to estimate state probabilities when event probabilities, which are aggregation of states, are known. This is challenging because there are fewer event probabilities than the number of unknown state probabilities. Our method, $\mathsf{mIRL}^*_{/T}$+Int builds on Bard's approach with several additional challenges. These include our use of features that are random variables, trajectories that may contain several events (aggregate features), occlusion, and its novel extension toward learning transition probabilities.

3.5 Expectation-Maximization with IRL

In chapter 4 we apply Expectation-Maximization to IRL in order to learn in the presence of hidden variables. An expectation is utilized to fill in missing data, parameterized with the current iteration's distribution over trajectories. EM was also applied to IRL in Nguyen et. al. [42] in the context of multiple locally-consistent reward functions. As the reward functions' parameters and the identity of the reward function in use at each timestep of the expert's demonstration is unknown EM is used to find the maximum likelihood parameterization. A number of component functions including priors over reward functions, reward function weights, and reward function transitions must be maximized during the M step. Once EM converges, the trajectories of the expert are partitioned using the Viterbi algorithm into the most likely segments each learned reward function generated.

3.6 Summary and Discussion

In our application domain, autonomous robots performing IRL on agents they encounter in the world with the goal of using the learned rewards as part of their task, some of the assumptions inherent in IRL are expected to be challenged. Particularly, occlusion is typically unavoidable due to the reality of on-board sensors. Kitani et. al. [26] model the expert's state as partially observable but this model may not work when the expert moves completely out of view. In contrast our occlusion model compensates for these completely hidden conditions but assumes that the expert is fully observable otherwise.

Interaction is common in environments with multiple agents and must be modeled to prevent the interactions being mis-attributed to the individual agents' reward function. With dense interactions causing interruptions in each agent's ideal behavior, Natarajan et. al. [37] build one large joint MDP that models the complete system and can account for the joint state during interactions. Or, if the interaction is critical to the agents' behavior they may be modeled as playing a game rather than executing separate MDPs; unfortunately this scenario is therefore outside the realm of IRL. We take a different approach: if the interaction is a temporary distraction from their individual tasks then the interaction itself may be modeled as a game and the non-interaction behavior as a MDP which may be inversely solved with IRL. This avoids the performance penalty associated with a joint MDP and issues that arise in the presence of occlusion.

Complete knowledge of the expert's transition function is difficult to attain for unknown agents. In response, model free approaches (Boularias et. al. [13]) and approaches that avoid calculating the reinforcement learning problem when performing IRL (Klein et. al. [28]) offer methods to obtain the expert's reward function without needing the transition function. However, for applications in which an expert is modeled as a MDP which is then utilized as part of the learner's task, for instance to predict the future positions of the expert, the transition function must be estimated from the data. We present a method for doing so in chapter 5 that is orthogonal to the specific IRL method used by utilizing side knowledge of the structure of the transitions.

Chapter 4

Learning in the Presence of Hidden Variables

In common IRL scenarios it is assumed the learner has full view of the entire state space the expert's behavior occurs in. This is the case in controlled scenarios, for instance in a motion capture lab when the expert is a human. When a robot must rely on its on-board sensors to observe the expert, however, it is likely that portions of the state space will be occluded from view due to sensor limits or objects in the environment. In this chapter we will describe this problem in detail and our solutions to it.

4.1 Effects of occlusion

Many solutions exist to the problem of *spurious* missing data. That is, a data set is randomly missing a small portion of fields in a few entries. For instance, if the missing entries are rare one could simply remove the offending instances. A more advanced technique is to simply set the missing fields to the mean of samples. These methods fail if the missing data is persistent, for instance if a field has no values whatsoever due to a recording error.

In many situations involving robotic learners state space occlusion will result in persistent missing data for some portion of the expert's behavior. Examples range from the expert has moved out of range of sensors, the expert moves behind an object or self-occludes, to more complicated examples like false observations due to distractions like mirrors or offending color blobs in the environment. Extreme occlusion may prevent the learner from being able to observe the expert taking a critical action, if multiple experts are present it precludes the use of a joint-MDP to model them as the states become partially observable in the event one expert is observed and the other isn't.

While it is tempting to simply remove the occluded states from the learner's model of the expert this does not work in general as the expert may in fact receive rewards in the occluded states or these states may be critical to the expert's trajectory. In these cases at least, removing occluded states completely from consideration distorts the reward function learned.

4.2 Simple solution

A simple and straightforward solution is to limit the feature expectation calculation to sum over only those states that are visible to the learner. We define the observable state set as $Obs(S)$, and for Maximum Entropy IRL (MaxEnt IRL) we update the feature expectation constraint accordingly:

$$\sum_{\pi \in \Pi} Pr(\pi) \sum_{k} \sum_{s \in Obs(S)} \mu_{\pi}(s) \phi_k(s, \pi(s)) = \hat{\phi}_k \tag{4.1}$$

As the occluded states are already removed from $\hat{\phi}$ by definition, the constraints now match. Note, however, that we still have K features and constraints, even those features which are defined **entirely** in the occluded states and thus **never** observed. This has the unfortunate effect of preventing the use of the gradient when solving, as some portion of $\hat{\phi}$ is now underrepresented, therefore we may not use a numerical solver that rely on the gradient such as L-BFGS or gradient ascent. Instead, we use the Nelder-Mead simplex technique [39] to solve this problem; this increases the time to solve as Nelder-Mead scales with the number of parameters. This cost, however, is not prohibitive as our time-limited experiments will show.

As we continue to use the non-occluded definition of the probability distribution:

$$Pr(\pi) \propto e^{\sum\limits_{k} \sum\limits_{s \in S} \mu_{\pi}(s) \phi_k(s, \pi(s))} \tag{4.2}$$

this program is now non-convex as there are potentially multiple equally likely explanations that match the feature expectations of the visible portion but differ in the occluded (proof available in appendix A.2.1). We label this algorithm IRL*, and note that we solve this program using the $Q - V$ approach described in section 6.2.

4.2.1 Evaluation

To validate our simple approach to occlusion we examine the performance of IRL* in randomly generated MDPs as we gradually increase the level of occlusion. We generate MDPs with six states, four actions, and random transition and reward functions. We optimally solve each one using value iteration and repeatedly sample 10 trajectories of length 10 timesteps using the optimal policy.

As can be seen in figure 4.1 both measures with no occlusion (occluded states = 0) match their corresponding data point in figure 2.6 (10 trajectories). As more states are occluded from view both measures smoothly approach their max/min. Notice that even in the face of mild occlusion accurate policies may still be learned and ILE is reasonable. Each data point is the average of averages from the output of 1000 trials grouped into blocks of size 10. Standard error is from 0.023 (6 occluded states, ILE) to 0.0058 (0 occluded states, LBA).

4.3 Expectation over hidden variables

Instead of simply ignoring the missing data a better solution may be to take the expectation over the missing data given the current probability distribution over the expert's trajectories. If we complete the missing data in this way it allows the use of all states in the constraint summation again and with it the calculation of the gradient. We will thus modify the MaxEnt IRL algorithm detailed in section 2.5 to account for the hidden data.

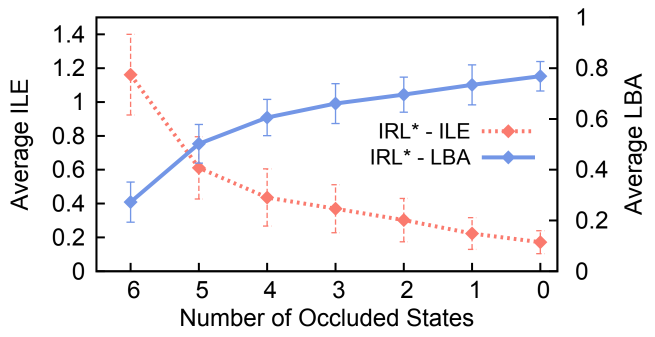


Figure 4.1: Average Inverse Learning Error (ILE) and Learned Behavior Accuracy (LBA) of IRL* as the number of occluded states decreases. Random MDPs with 6 states and 4 actions, trajectories of length 10 and 10 trajectories sampled. Error bars are one standard deviation, standard error is between 0.023 (6 occluded states, ILE) to 0.0058 (0 occluded states, LBA)

We rewrite the definition of $\hat{\phi}_k = \sum_{X \in \mathcal{X}} \tilde{Pr}(X) \sum_{\langle s,a \rangle \in X} \phi_k(s,a)$ where $\mathcal{X}$ is the set of observed trajectories and $\tilde{Pr}(X)$ is the empirical probability of a given trajectory X. We may rewrite this as $\sum_{Y \in \mathcal{Y}} \sum_{Z \in \mathbb{Z}} \tilde{Pr}(Y,Z) \sum_{\langle s,a \rangle \in X} \phi_k(s,a)$ where $X = (Y \cup Z)$, Y is the portion of each trajectory observed, Z is one way of completing the hidden portions of this trajectory and $\mathbb{Z}$ is the set of all possible Z. Now we may treat Z as a latent variable and take the expectation:

$$\hat{\phi}_k^{Z|_Y} \triangleq \sum_{Y \in \mathcal{Y}} \tilde{Pr}(Y) \sum_{Z \in \mathbb{Z}} Pr(Z|Y) \sum_{\langle s,a \rangle \in Y \cup Z} \phi_k(s,a) \tag{4.3}$$

The revised program from equation 2.13 is now:

$$\begin{aligned} &\max_{\Delta} \left(- \sum_{X \in \mathbb{X}} Pr(X) \; log \; Pr(X) \right) \\ &\textbf{subject to} \quad \sum_{X \in \mathbb{X}} Pr(X) = 1 \\ &\sum_{X \in \mathbb{X}} Pr(X) \sum_{\langle s,a \rangle \in X} \phi_k(s,a) = \hat{\phi}_k^{Z|_Y} \qquad \forall k \end{aligned} \tag{4.4}$$

Notice that in the case of no occlusion $\mathbb{Z}$ is empty and $\mathcal{X} = \mathcal{Y}$, therefore equation 4.4 reduces to equation 2.13. Thus, the above problem generalizes the original MaxEnt IRL problem.

This new program is non-convex due to the presence of $Pr(Z|Y)$ and therefore presents some challenges in solving it as we detail in the next section. It may be viewed as an application of Wang et. al.'s [59] generalization of the maximum entropy method to allow for hidden variables.

4.3.1 Method of Latent Maximum Entropy

As a first step towards a solution to equation 4.4, we proceed as is a common procedure with non-linear programs by taking the Lagrangian and finding the derivative with respect to $Pr(X)$

with the goal of finding a definition of $Pr(X)$. Unfortunately, due to the presence of $Pr(Z|Y)$ we do not arrive at a closed form solution:

$$\frac{\partial \mathcal{L}}{\partial Pr(X)} = -log\ Pr(X) - 1 + \sum_k \theta_k \sum_{\langle s,a\rangle \in X} \phi_k(s,a)$$
$$+ \sum_k \theta_k \left(\sum_{Y \in \mathcal{Y}} \tilde{Pr}(Y) \frac{\sum_{Z' \in \mathbb{Z}} [\sum_{\langle s,a\rangle \in X'} \phi_k(s,a) - \sum_{\langle s,a\rangle \in X} \phi_k(s,a)] Pr(X')}{Pr(Y)^2} \right) + \eta$$

where $X' = (Y \cup Z')$. Wang *et. al.* [59] suggest approximating the above partial derivative with the following simpler form:

$$\frac{\partial \mathcal{L}}{\partial Pr(X)} \approx -log\ Pr(X) - 1 + \sum_k \theta_k \sum_{\langle s,a\rangle \in X} \phi_k(s,a) + \eta$$

Setting the above to 0 and solving for $Pr(X)$ yields an optimum for $Pr(X)$ that is log linear:

$$Pr(X) \approx \frac{e^{\sum_k \theta_k \sum_{\langle s,a\rangle \in X} \phi_k(s,a)}}{\Xi(\boldsymbol{\theta})} \tag{4.5}$$

Now that we have an (approximate) closed form definition of $Pr(X)$, we proceed by substituting this definition back into equation 4.4. We then arrive at the objective function to minimize.

$$\mathcal{L}^{\text{dual}}(\boldsymbol{\theta}) \approx log\ \Xi(\boldsymbol{\theta}) - \sum_k \theta_k \sum_{Y \in \mathcal{Y}} \tilde{Pr}(Y) \sum_{Z \in \mathbb{Z}} Pr(Z|Y) \sum_{\langle s,a\rangle \in Y \cup Z} \phi_k(s,a) \tag{4.6}$$

To assist in minimization, we attempt to find the gradient of equation 4.6 with respect to the Lagrangian parameter vector Θ. Unfortunately we again do not arrive at a closed form solution for the gradient due to the presence of $Pr(Z|Y)$.

Expectation-Maximization Solution

Wang et. al. [59] offer an EM-based solution which we will show produces a valid solution to equation 4.6. Due to the non-convex nature of the original problem and the use of a log-linear approximation, this solution is not guaranteed to be the optimal to equation 4.4. Thus, multiple restarts with varying initial parameters are required and among the found solutions the one with the highest entropy is chosen as the final solution.

We seek to maximize the likelihood of Lagrangian parameters $\boldsymbol{\theta}$, where the log likelihood is defined as $LL(\boldsymbol{\theta}|\mathcal{Y}) = log\ Pr(\mathcal{Y}; \boldsymbol{\theta})$. This becomes,

$$\begin{aligned} LL(\boldsymbol{\theta}|\mathcal{Y}) &= log \prod_{Y \in \mathcal{Y}} Pr(Y; \boldsymbol{\theta}) = \sum_{Y \in \mathcal{Y}} \tilde{Pr}(Y)\ log\ Pr(Y; \boldsymbol{\theta}) \\ &= \sum_{Y \in \mathcal{Y}} \tilde{Pr}(Y)\ log\ Pr(Y; \boldsymbol{\theta}) \sum_{Z \in \mathbb{Z}} Pr(Z|Y; \boldsymbol{\theta}) \\ &= \sum_{Y \in \mathcal{Y}} \tilde{Pr}(Y) \sum_{Z \in \mathbb{Z}} Pr(Z|Y; \boldsymbol{\theta})\ log\ Pr(Y; \boldsymbol{\theta}) \end{aligned}$$

Rewriting $Pr(Y;\boldsymbol{\theta}) = \frac{Pr(Y,Z;\boldsymbol{\theta})}{Pr(Z|Y;\boldsymbol{\theta})}$ in the above equation we get,

$$\begin{aligned} LL(\theta|\mathcal{Y}) &= \sum_{Y\in\mathcal{Y}} \tilde{Pr}(Y) \sum_{Z\in\mathbb{Z}} Pr(Z|Y;\boldsymbol{\theta})\ log\ \frac{Pr(Y,Z;\boldsymbol{\theta})}{Pr(Z|Y;\boldsymbol{\theta})} \\ &= \sum_{Y\in\mathcal{Y}} \tilde{Pr}(Y) \sum_{Z\in\mathbb{Z}} Pr(Z|Y;\boldsymbol{\theta})\,(log\ Pr(Y,Z;\boldsymbol{\theta}) - log\ Pr(Z|Y;\boldsymbol{\theta})) \end{aligned}$$

The above likelihood may be iteratively improved until convergence (possibly to a local optima) by casting it in an EM scheme. Specifically, the likelihood may be rewritten as, $Q(\boldsymbol{\theta},\boldsymbol{\theta}^{(t)})+C(\boldsymbol{\theta},\boldsymbol{\theta}^{(t)})$ where,

$$Q(\boldsymbol{\theta},\boldsymbol{\theta}^{(t)}) = \sum_{Y\in\mathcal{Y}} \tilde{Pr}(Y) \sum_{Z\in\mathbb{Z}} Pr(Z|Y;\boldsymbol{\theta}^{(t)})\ log\ Pr(Y,Z;\boldsymbol{\theta}) \tag{4.7}$$

$$C(\boldsymbol{\theta},\boldsymbol{\theta}^{(t)}) = -\sum_{Y\in\mathcal{Y}} \tilde{Pr}(Y) \sum_{Z\in\mathbb{Z}} Pr(Z|Y;\boldsymbol{\theta}^{(t)})\ log\ Pr(Z|Y;\boldsymbol{\theta})$$

Notice that in equation 4.7, we may replace $Pr(Y,Z;\boldsymbol{\theta})$ with $Pr(X;\boldsymbol{\theta})$, and we may now substitute in equation 4.5. This gives us,

$$\begin{aligned} Q(\boldsymbol{\theta},\boldsymbol{\theta}^{(t)}) &= \sum_{Y\in\mathcal{Y}} \tilde{Pr}(Y) \sum_{Z\in\mathbb{Z}} Pr(Z|Y;\boldsymbol{\theta}^{(t)}) \sum_k \theta_k \sum_{\langle s,a\rangle\in X} \phi_k(s,a) - log\ \Xi(\boldsymbol{\theta}) \\ &= -\left(log\ \Xi(\boldsymbol{\theta}) - \sum_k \theta_k \sum_{Y\in\mathcal{Y}} \tilde{Pr}(Y) \sum_{Z\in\mathbb{Z}} Pr(Z|Y;\boldsymbol{\theta}^{(t)}) \sum_{\langle s,a\rangle\in X} \phi_k(s,a) \right) \end{aligned}$$

Notice that the Q function above is almost the negative of the dual Lagrangian presented in equation 4.6 with the only difference being $\boldsymbol{\theta}^{(t)}$. On maximizing Q to convergence, we have $\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)}$, due to which we conclude that maximizing the Q function minimizes the dual Lagrangian function. Therefore, approximately solving the maximum entropy IRL of equation 4.4 for $\boldsymbol{\theta}$ is equivalent to maximizing the Q function; Theorem 1 states this formally.

Theorem 1. *Under the assumption that the distribution of the complete trajectories is log linear, the reward function that maximizes the log likelihood of the observed incomplete trajectories also maximizes the entropy of the distribution constrained by the conditional expectation of the feature functions.*

This gives us another path to a feasible solution to the generalized IRL in equation 4.4 subject to the constraints and the log-linear model for $Pr(X)$. However, due to the non-convexity of the Lagrangian this solution may not necessarily be the optimal one.

Using this insight, we may now split the primary constraint in equation 4.4 into two portions and solve each separately.

E-step

Consequently, the E-step involves obtaining a conditional expectation of the K feature functions using the parameter $\boldsymbol{\theta}^{(t)}$ from the previous iteration. We initialize the parameter vector randomly. For all $k = 1 \ldots K$ we obtain,

$$\hat{\phi}_k^{Z|_Y,(t)} = \sum_{Y\in\mathcal{Y}} \tilde{Pr}(Y) \sum_{Z\in\mathbb{Z}} Pr(Z|Y;\boldsymbol{\theta}^{(t)}) \sum_{\langle s,a\rangle\in Y\cup Z} \phi_k(s,a) \tag{4.8}$$

M-step

This involves solving a simpler version of the constrained maximum entropy program of equation 4.4 by utilizing $\hat{\phi}_k^{*,(t)}$ from the E-step above and the log linear model for $Pr(X)$ to obtain $\boldsymbol{\theta}$.

$$\begin{aligned}
&\max_{\Delta} \left(- \sum_{X \in \mathbb{X}} Pr(X)\ log\ Pr(X) \right) \\
&\textbf{subject to} \quad \sum_{X \in \mathbb{X}} Pr(X) = 1 \\
&\sum_{X \in \mathbb{X}} Pr(X) \sum_{\langle s,a \rangle \in X} \phi_k(s,a) = \hat{\phi}_k^{Z|_Y,(t)} \qquad \forall k
\end{aligned} \tag{4.9}$$

4.3.2 Time to solve considerations

Equation 4.8 requires a summation over every possible Z for each Y. As Z is one possible way of completing the missing data to result in X the number of possible $|\mathbb{Z}|$ grows exponentially with the number of timesteps of missing data: $|\mathbb{Z}| = (|S_{occ}| \times |A|)^t$ where $S_{occ} = S - Obs(S)$. As the E step must be computed repeatedly over the course of the expectation-maximization algorithm the number of missing timesteps quickly becomes the determining factor for run length.

An alternative to this exact approach is to estimate $\hat{\phi}_k^{Z|_Y,(t)}$ using Monte-Carlo Integration. The distribution is sampled using Gibbs sampling to produce a Markov chain whose equilibrium distribution matches, producing the Monte-Carlo Expectation-Maximization algorithm (MCEM) as described in section 2.1.4. We thus treat X as a Bayesian network in which some nodes are known completely and others completely hidden. There are two types of nodes to sample: states and actions. As each node in a Bayesian net is independent given its Markov blanket, we may sample according to:

$$\begin{aligned}
Pr(s_t|MB(s_t)) &= \eta T(s_{t-1}, a_{t-1}, s_t) T(s_t, a_t, s_{t+1}) Pr(a_t|s_t) \\
Pr(a_t|MB(a_t)) &= \eta' T(s_t, a_t, s_{t+1}) Pr(a_t|s_t)
\end{aligned}$$

As the E step in MCEM is only approximate we may not rely on convergence of Q as an indicator of completion [38]. Instead we fix the number of iterations manually in our experiments.

4.4 Experiments and Results

We present two problem domains to test the performance of the algorithms described in this chapter. We utilize IRL* described in section 4.2 and we label the Expectation-Maximization approach in section 4.3 *Hidden Data EM.*

4.4.1 Domains

Fugitive reconnaissance

Our first domain is simulation only and models a scenario where a fugitive agent is being observed by a UAV as it tries to make it to a safe sector. There are two possible sectors that may be the

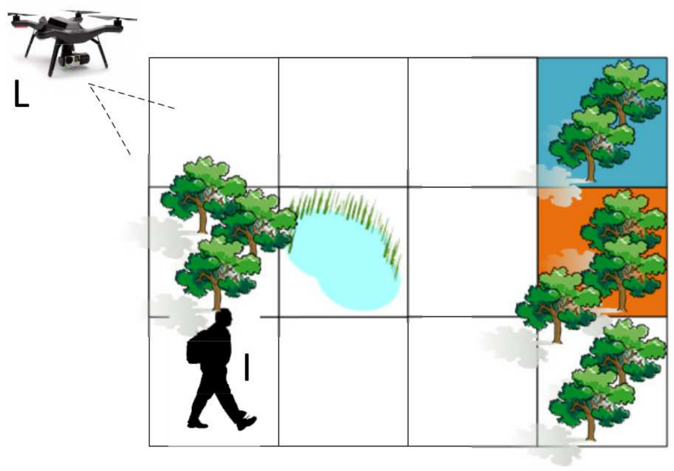


Figure 4.2: The UAV tracks the fugitive (I) as it moves through the grid of states. States may be occluded from the UAV's view by foliage. The sectors shaded blue and orange are the true safe and penalty states respectively.

safe one, but the area may be occluded from view by foliage. In another setup, the UAV changes positions between trajectories resulting in varying levels of occlusion of the state space. A diagram of the problem is shown in figure 4.2.

The states of the resulting MDP are simply the coordinates of each sector, actions are move along each of the cardinal directions. We assume the fugitive is executing a fixed, deterministic policy and that its transitions are stochastic (90% chance of arriving at the expected state with the remaining mass distributed evenly to other actions). The fugitive's reward function is modeled as a linear combination of three features:

1. **Cost of movement** is a small penalty for taking any action in all states, this encourages I to reach the goal state efficiently;

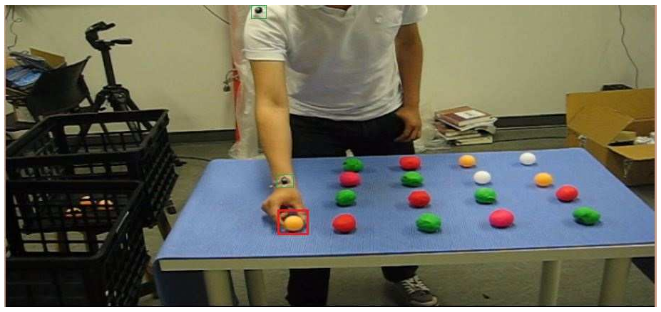
2. **Goal state reached** is activated if I reaches sector (4, 3);

3. **Penalty state reached** is activated if I reaches sector (4, 2).

Learning from human demonstration

Our second domain involves a human attempting to teach a robotic arm a fruit sorting task. The human's arm is tracked using a motion capture system as well as recorded using a color video camera. These recordings provide enough data for a learner to determine which type of fruit (here using different colored and sized balls) should be sorted into which bin. However, there is one complication: some of the fruit is ripe and therefore must be handled gently but there is no direct measurement of the force applied by the human's hand to the fruit.

Four types of balls - 6 each - are present. Two types made of soft clay represent ripe fruit that must be gently handled and the other types are ping pong balls representing unripe fruit that can be handled roughly. The sorting setup is shown in figure 4.3.


(a)

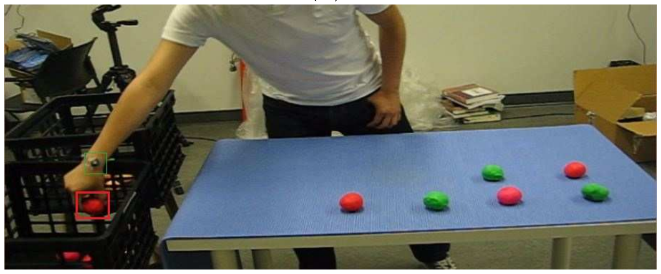

(b)

Figure 4.3: Video captures from our experiment showing a subject performing the ball sorting task. Notice that there are four types of balls in all - soft clay balls of two colors and hard ping-pong balls of two colors. Balls are sorted into two bins based on their color. (a) Subject picks a hard ball, and (b) Subject places a soft red ball in one of the bins.

To model the human, we discretize the space their hand may be in into four values { on_table, in_center, bin_1, bin_2 }. Additional variables include the ball type { 1,2,3,4 } and whether a ball is being held { yes, no }.

While we do not directly measure the force applied to each ball there is a noisy, indirect indication available in the form of the amount of time the human spent grabbing and placing the ball. Soft grabs and places are slightly slower on average but overlap significantly with hard ones. By adding a discretized "time for previous action" value to each state, we can model this by using a stochastic transition function - soft grabs result in higher probability of next states with larger "time for previous action" values and vice versa. This information is not enough to label the grab or place action as this may result in incorrect expert trajectories. Instead, we hypothesize that the *Hidden Data EM* algorithm, by taking the expectation over the missing action, can learn reward values that explain "time for previous action" values in the trajectory. This variable is discretized into 12 segments: {less than 0.3s, 0.3s, 0.4s, . . . , 1.3s, greater than 1.3s}.

The human performs one of the following seven actions:

- *Grab hard* denotes the hand grabbing a ball with default force and sets the holding_ball factor to true;
- *Grab soft* denotes the hand grabbing a ball gently and sets holding_ball to true.

On average, grabbing a ball gently is slower than grabbing it by default. Both grab actions are applicable for states where the ball position is on the table only;

- *Move To center* denotes the hand moving to the center position while holding a ball. This action applies for states where the hand has grabbed the ball that is on the table;
- *Move to bin 1* is the hand moving to bin 1 while holding the ball;
- *Move to bin 2* is the hand holding the ball moving to bin 2;

  These actions that move the ball to the bins are applicable when the ball is held and its position is in the center;
- *Release hard* places the ball with default force in a bin and sets the factor holding_ball to false; and on average is faster than grab soft, only for states in the center position
- *Release soft* places the ball gently in a bin and sets holding_ball to false.

  On average, placing a ball gently in a bin is slower than placing it with regular force. Both release actions are applicable when the ball is held and its position is in the center;

The human's reward function is modeled as having the following feature functions:

- **Release ball type X in bin 1** leads to 4 feature functions each of which is activated when a ball of that type is placed in bin 1.
- **Release ball type X in bin 2**, 4 features each of which is activated when a ball of that type is placed in bin 2.
- **Handle ball type X gently** also leads to 4 features each of which is activated when the ball of the corresponding type is grabbed and moved gently.

In total, the reward function is composed of 12 weighted feature functions.

As this is an apprenticeship learning task, we also model the robot arm, in this case a Trossen Robotics PhantomX Pincher, with an MDP. The parameters of this MDP are similar to the human's with the exception of the transition function. In the human's MDP transitions are primarily modeled as arriving at their intended next state with a probability of 0.9 with the remaining probability mass distributed to the other actions. Then the probability mass is distributed according to the "time for previous action" variable's probabilities. For the robot, we noted an increased chance of dropping a ball when using the soft grip and so modeled the transitions accordingly, specifically probability 0.9 of successfully sorting the ball and 0.1 of dropping it vs. a hard grip of 0.99 and 0.01 respectively. *As a result, the optimum policy for the robot's MDP prefers the use of a hard grip unless rewarded for using a soft grip with a specific ball per the reward function.*

4.4.2 Metrics and Baselines

When the true reward function is unknown or the learner's and expert agents' MDP differ significantly it is common to evaluate the performance of the learning agent using a performance measure related to its assigned task. Consequently, we adopt this approach for the ball sorting task.

We evaluate three variations of the algorithms presented in this chapter: MaxEnt IRL is a variation of the algorithm described by equation 2.13 in which occluded states are removed from all considered trajectories, IRL^* is the algorithm described in section 4.2, and *Hidden Data EM* is described in section 4.3. In the sorting domain all states are visible with some actions occluded, however the first two algorithms cannot correctly model this condition and so any occluded actions also cause the states to count as occluded for MaxEnt IRL and IRL^* (*Hidden Data EM* is unaffected). One additional limitation of the first two algorithms is that the occluded states must be constant throughout all observed trajectories, as such these algorithms cannot be applied to scenarios where the observability of the state space varies.

4.4.3 Results

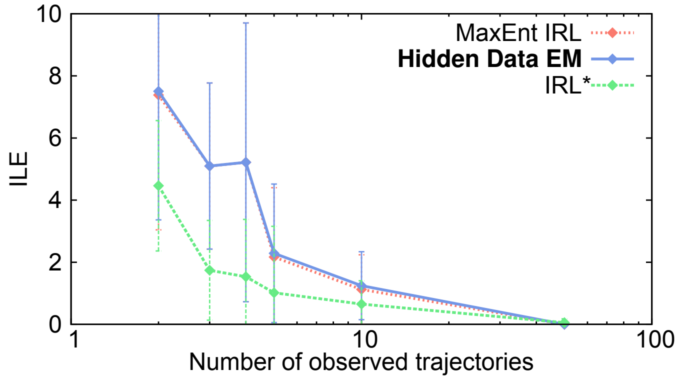
For our first experiment, we compared the ILE of all three algorithms in the fugitive domain. We first develop a control where the state space is completely visible. Here we expect the performance of *Hidden Data EM* to be identical to MaxEnt IRL. Next, we test the three algorithms when the last column of the map is occluded which presents a challenge as this includes the two terminal states. Finally, we vary the occlusion per observed trajectory simulating a UAV moving through the space. Each method received up to 50 trajectories from the fugitive with each one comprising 5 steps. The fugitive is started in a random state and executes a single policy in all samples.

Figure 4.4 (a) shows the results of the occlusion free control, notice that all methods reduce ILE as more trajectories are considered and *Hidden Data EM* is identical to MaxEnt IRL. Both of these methods are out-performed by IRL^* and this is attributed to the short trajectory length as it is possible the fugitive does not reach a goal state in 5 timesteps due to the stochastic transition function. In the event this happens, the two trajectory based methods are unable to attribute the motion of the fugitive to a goal state as the trajectory does not include them, whereas the policy based IRL* matches feature expectations of all visible states and is therefore unaffected.

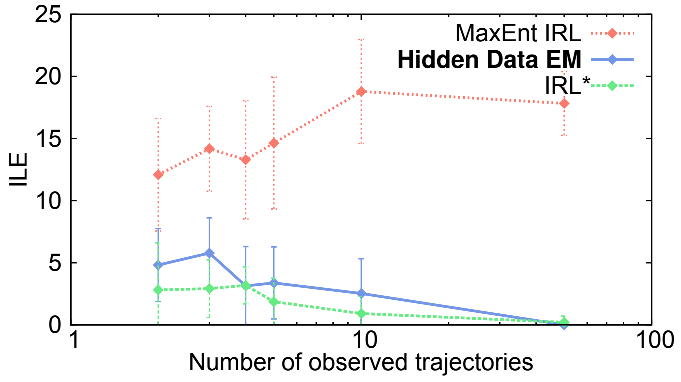
Notice the dramatic increase in error the MaxEnt IRL algorithm experiences in figure 4.4 (b). As the goal states are permanently removed from observation in this experiment this method will never consider the features associated with these states. Notice too that despite being derived from MaxEnt IRL, *Hidden Data EM* effectively overcomes this limitation and performs as well as IRL^*. Finally, in 4.4 (c) we see *Hidden Data EM* reducing ILE as the number of trajectories increases, no baseline is provided because we are unaware of any other IRL algorithm that operates with varying occlusion.

In our second experiment we used a robotic arm to sort 24 balls of varying types into two bins. Each of the three algorithms in this chapter was given the same set of recorded trajectories of a human performing this task. The reward functions found by each are transferred to the robot's MDP which was then executed repeatedly to sort all given balls. As mentioned previously, the specific action taken by the human during a grab or place is occluded from view and to provide an additional baseline we fill in the missing action with the true one execute MaxEnt IRL on the complete trajectory, we label this *Occlusion-Free control.*

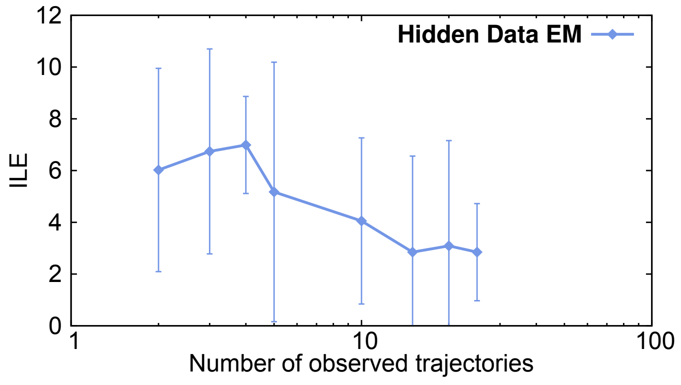
Table 4.1 shows the performance of all algorithms, while all methods learned to correctly sort the balls into the correct bin, they differ in the grasp chosen for each ball type. Here we can see *Hidden Data EM* outperformed the other methods and matched the performance of the *Occlusion-Free control.* MaxEnt IRL, unable to infer the action the human used, simply used the default hard grasp which deformed most of the soft balls. IRL* learned that one type of soft ball should be handled gently but failed to find the same for the other, additionally it incorrectly found that a hard ball should be handled gently and as a result dropped more balls than the other methods. We show some images of the arm performing a sort in figure 4.5.



(a)



(b)



(c)

Figure 4.4: Performance evaluation of all three methods in the UAV reconnaissance scenario. (a) ILE for all methods in the fully observable setting. Notice that Hidden Data EM's performance coincides with that of MaxEnt IRL as we may expect. (b) ILE the fugitive's trajectories are partially occluded, notice the failure of MaxEnt IRL due to the absence of the critical features from all observed trajectories. (c) High levels of occlusion which changes randomly between trajectories due to UAV movement.

Method	Ball correctly sorted	Balls dropped	Balls damaged
MaxEnt IRL	23	1	11
IRL*	21	3	5
Hidden Data EM	**23**	**1**	**0**
Occlusion-free control	23	1	0

Table 4.1: Performance of the various methods on the PhantomX Pincher arm on the ball sorting task. Notice that Hidden Data EM did not damage any balls and dropped just one ball while sorting. Its performance is similar to our occlusion-free control method that provides an upper bound.

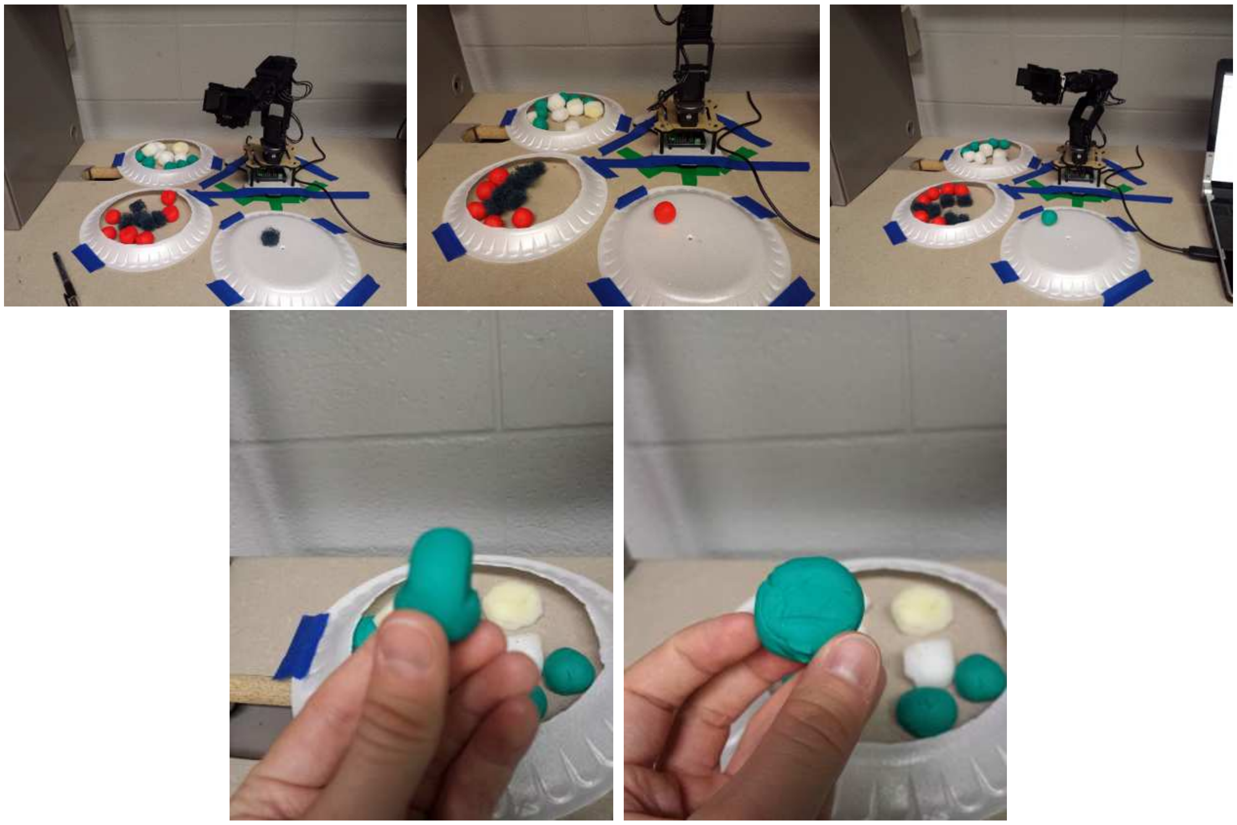


Figure 4.5: A series of snapshots of the PhantomX Pincher arm sorting the balls. We introduce balls one at a time due to lack of sufficient space for several balls and the arm being small. The two bins with balls are visible on the left side of the arm. We show the deformed clay ball due to mishandling and the non-deformed clay ball on the bottom.

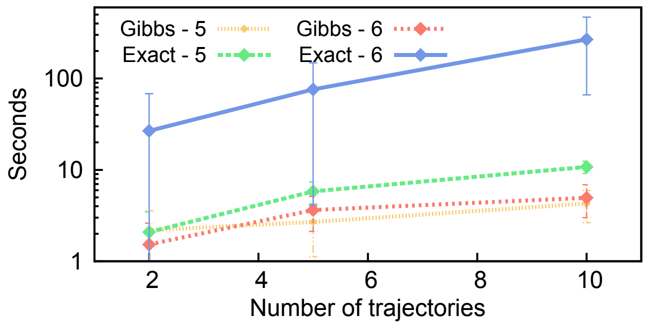


Figure 4.6: Average time to complete learning in the fugitive domain for exact Hidden Data EM and a Monte-Carlo EM variant as the number of provided trajectories increases. Note logarithmic scale. Occlusion was randomized. Error bars are one standard deviation, standard error is between 44.94s (10 trajectories of length 6, Exact method) to 0.24s (2 trajectories of length 6, Gibbs)

4.4.4 Solving speed of Exact EM vs MCEM

Experiments in the previous sections could be solved exactly due to the short lengths of the provided trajectories. As described in section 4.3.2 the size of $|Z|$ increases exponentially with the number of occluded timesteps, dramatically increasing the time required to solve *Hidden Data EM*.

We show an example of time to solve improvement of using MCEM with *Hidden Data EM* in figure 4.6 in the fugitive domain. Besides comparing trajectories of different lengths (length 5 and 6) we also increase the number of provided trajectories; the size of $\mathbb{Z}$ grows linearly with the number of trajectories and allows easy comparison between the two methods.

4.5 Summary

In the presence of occlusion the normally convex Maximum Entropy IRL program becomes non-convex as it is possible the observed behavior of the expert may not be matched with the features defined in the observable space alone. Simply limiting constraints to the effects on the observable feature expectations is sufficient in many cases to learn accurate reward functions and benefits from a solving time that does not depend upon the degree of occlusion.

By completing the hidden variables with an expectation given the currently learned reward function all states may be considered in the constraints. This new program may be approximately solved with an Expectation-Maximization based algorithm in which an expectation for the expert's feature expectations is calculated in the E step and the original Maximum Entropy IRL program is solved in the M step. While gaining improved accuracy over simply ignoring the missing data, calculating the E step exactly is difficult as the size of the potential trajectories the expert could have traversed grows exponentially in the number of timesteps it was hidden. Monte-Carlo integration may be effectively employed to estimate the E step in this case.

As all states are again considered in the constraints dynamic occlusion may naturally be handled by simply allowing $Obs(S)$ to vary with time. Dynamic occlusion is expected to be encountered when the learning agent moves through the environment resulting in different parts of the state space being occluded during the expert's demonstration. For autonomous mobile robots employing IRL this feature may prove critical to successful learning.

Chapter 5

Learning with Unknown Dynamics

One of the assumptions inherent in most IRL formulations is that the transition function of the expert agent is known to the learner. In some cases, transitions are deterministic and easily modeled. Given enough data, existing techniques can estimate a stochastic transition function an expert agent is using by modeling the transitions as a dynamic Bayesian network and employing a parameter learning method. However, this method is challenged when faced with occlusion - it is possible that transitions from some states are never observed, and further some *actions* may never be observed as the expert agent is not expected to execute all actions in every state (otherwise it would not be optimal!). In this chapter we make progress toward solving this problem by assuming a structure to the transition function and, utilizing side information about the expert agents, intelligently transfer information about transition probabilities from observable states to occluded areas.

5.1 Transition Model

When modeling an agent with an MDP one common technique is to design the state and action space as if the transition function is deterministic and then apply some sort of error model to produce stochastic transitions. With this in mind, we describe the following transition structure: assume that for every state and action there is one *intended* outcome state: ψ: $S \times A \rightarrow S$. We may view this as a deterministic transition function; to make it stochastic we must first choose a probability with which the agent arrives at the intended state and assign this probability to $T(s, a, \psi(s, a))$. The remaining probability mass, $1 - T(s, a, \psi(s, a))$, must now be distributed according to a chosen error model. This could be as simple as uniformly distributing it among all other states, assigning it all to an error state, or distributing it among the other actions' intended outcome states given the current state.

We must now assume that the learner has full knowledge of ψ and an accurate error model to use. At first glance, this might seem difficult to satisfy. There are, however, two factors which increase the likelihood of this model being applicable. First, it is the learner who is constructing an MDP to model the expert and as such may be free to add extra states and actions as it sees fit until this deterministic-core transition model applies. Second, when modeling movement in robotic domains the transitions are bounded by physical laws and as such intended next states of actions and the error model may be determined accurately.

We have now condensed the problem of learning the transition function to estimating the values of each $T(s, a, \psi(s, a))$ by simply counting the proportion of successful transitions (agent arrived at its intended next state) out of all attempted transitions. This does not, however, address the issue with occlusion as we still have no information about the transitions in occluded states.

5.2 Transition Features

We now wish to transfer the information we have about successful transition probabilities from the observable states to occluded ones. While a simple solution is to assign unobserved $T(s, a, \psi(s, a))$ the value of those that are observed, it is not always justified to do so. One situation where doing this *is* justified is when we have side information about the expert agent that indicates the transition in question has the same probability as one we can observe. We extend this concept with the use of transition features. We begin by attributing a successful transition as caused by an underlying set of features all operating successfully. In the event that any one feature fails, the agent instead follows the given error model.

We will model these features as binary random variables with each state-action pair mapped to some subset of these features. Let $\xi^{s,a} = \{\tau_1, \ldots, \tau_k\}$ be the subset of independent features mapped to a state-action pair, $\langle s, a \rangle$, where each feature, $\tau \in \mathcal{T}$, is a binary random variable whose states are τ (success) and $\bar{\tau}$ (failure), and $\mathcal{T}$ is the set of all transition features. We assume that both the set of features and mapping to states-actions is known to the learner - these could be provided by an abstract model of the type of agent being observed for instance. Note that $\bigcup\limits_{(s,a)} \xi^{s,a} = \mathcal{T}$.

We now define for all transitions

$$T(s, a, \psi(s, a)) \approx \prod_{\tau \in \xi^{s,a}} Pr(\tau) \tag{5.1}$$

This definition is approximate because the relationship between the feature variables is ideally described by a Bayesian network. In the interest of simplicity and tractability (computing the joint of a Bayesian network takes exponential time in general) we make an independence assumption between the features. Equation 5.1 is loosely analogous to the use of features in reward functions.

As an example of transition features, consider an autonomous differential drive robot. Suppose we model this robot according to the above method and want to describe a Move Forward action, with the intended next state being one meter ahead of its current position. To accomplish this task, the robot will need to move both of its wheels at the same rate, its navigation system must be localized well, and the floor it is on must be smooth and not slippery. Thus we have four transition features and the probability of moving one meter forward is: Pr(Left wheel rotating correctly) $\times$ Pr(Right wheel rotating correctly) $\times$ Pr(Navigation system localized) $\times$ Pr(Floor is traversable). Figure 5.1 illustrates the above feature mapping.

The primary advantage of this technique is that some state-action pairs may share features. If occluded state-actions share features with state-actions that are visible then by learning the features' success probabilities we can transfer transition information from the observable areas to the occluded ones. In the case that all features of occluded state-actions are shared with at least one visible state we gain the ability to recover the complete transition function.

5.2.1 Observed Probabilities

We have now reduced the task of estimating each $T(s, a, \psi(s, a))$ to estimating a much smaller set of $Pr(\tau)$. However, the challenge a learner faces is the observed trajectory of another agent likely

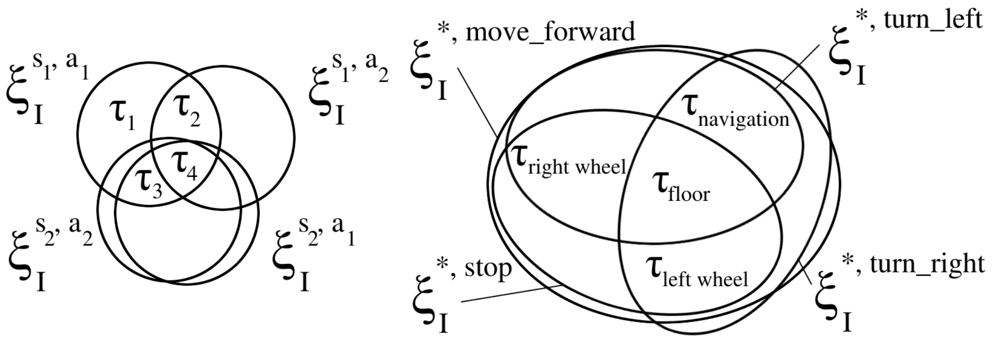


Figure 5.1: Venn diagrams showing example intersections between sets $\xi_I^{s_1,a_1}$, $\xi_I^{s_1,a_2}$, $\xi_I^{s_2,a_1}$ and $\xi_I^{s_1,a_2}$, and the mapping used by differential drive robots in our scenario.

cannot be attributed directly to specific features. Observed transitions provide only aggregated empirical distributions. In other words, given a trajectory performed by an agent of length T: $\{\langle s,a\rangle^0, \langle s,a\rangle^1, \ldots, \langle s,\varnothing\rangle^T\}$, where $\varnothing$ is the null action, we know nothing other than the proportion of times the intended next state is arrived at for the observed state-actions. This value, $q^{\psi(s,a)}$, is calculated as:

$$q^{\psi(s,a)} = \frac{\sum\limits_{t=0:\langle s^t,a^t\rangle=\langle s,a\rangle}^{T-1} \delta(s^{t+1}, \psi(s^t, a^t))}{\sum\limits_{t=0:\langle s^t,a^t\rangle=\langle s,a\rangle}^{T-1} 1}$$

where $\delta(\cdot,\cdot)$ is the indicator function that is equal to 1 when its two arguments are equal, otherwise 0.

Notice that the probability, $q^{\psi(s,a)}$, is equivalent to $T(s,a,\psi(s,a))$. As a result we can relate equation 5.1 to the observed transition probabilities:

$$\prod_{\tau\in\xi^{s,a}} Pr(\tau) = q^{\psi(s,a)} \quad \forall\ \langle s,a\rangle \tag{5.2}$$

While it is possible that a unique solution exists to this set of equations, for instance if certain transitions map to exactly one feature, in general we arrive at an ill-posed problem as there could be an infinite number of ways to assign each $Pr(\tau)$ which satisfy the constraints.

5.3 Maximum Entropy Solution

One way to make progress is to utilize the principle of maximum entropy. Under this principle we will choose the one set of feature probabilities which has the highest entropy while still satisfying the constraints. As mentioned in section 2.5 the resulting distribution encodes the available information from the constraints with no extra assumptions beyond what is needed to satisfy them. We therefore expect the results to be the most generalizable - considering all of the possible ways of assigning

$Pr(\tau)$ given a set of constraints, the one with maximum entropy will be least wrong most often. This is an important feature as we intend to project the observable transition information to unseen transitions.

The nonlinear, non-convex optimization problem for finding the transition feature distribution is:

$$\begin{aligned} &\max_{\Delta} - \left(\sum_{\tau \in \mathcal{T}} Pr(\tau) log\ Pr(\tau) + Pr(\bar{\tau}) log\ Pr(\bar{\tau}) \right) \\ &\textbf{subject to} \\ &\prod_{\tau \in \xi^{s,a}} Pr(\tau) = q^{\psi(s,a)} \quad \forall \langle s, a \rangle \\ &Pr(\tau) + Pr(\bar{\tau}) = 1 \qquad \forall \tau \in \mathcal{T} \end{aligned} \tag{5.3}$$

The Lagrangian relaxation of the above problem is:

$$\begin{aligned} \mathcal{L}(\mathcal{T}, \mathbf{v}, \boldsymbol{\lambda}) = &- \left(\sum_{\tau \in \mathcal{T}} Pr(\tau) log\ Pr(\tau) + Pr(\bar{\tau}) log\ Pr(\bar{\tau}) \right) \\ + &\sum_{j=1}^{|S||A|} v_j \left(\left(\prod_{\tau \in \xi^{s,a}} Pr(\tau) \right) - q^{\psi(s,a)} \right) + \sum_{\tau \in \mathcal{T}} \lambda_i \left(\left(Pr(\tau) + Pr(\bar{\tau}) \right) - 1 \right) \end{aligned} \tag{5.4}$$

In practice, multiple state-action pairs may map to identical sets of transition features. In this case, we need not distinguish between these state-action pairs in the above program, and may obtain a single observed probability by including the transitions for all the state-action pairs with common identical feature mappings in a single q. Thus, depending upon the complexity of the feature mapping, the number of q's may be significantly smaller than $|S||A|$. This method is similar to that used in Bard [8].

To properly handle saddle points in the Lagrangian function we take the sum of squares of the partial derivatives of $\mathcal{L}$ to arrive at the final objective function, $\mathcal{L}'$. Next, we minimize $\mathcal{L}'$ by using the penalty approach with Broyden-Fletcher-Goldfarb-Shanno (BFGS) [16] unconstrained optimization method.

We refer this new algorithm as mIRL$^*_{/T}$+Int.

5.4 Estimating a Full Transition Function

Unseen state-action pairs, whether due to being in the occluded portion of the robot's state space or due to the robot never taking a certain action in a given state, may share transition features with state-action pairs that have been observed. Thus we obtain informed probabilities (perhaps partially) for these unseen transitions. This observation highlights an advantage of the use of transition features and in the special case where the observed transitions exercise all features we gain the ability to recover the complete transition function. We note again that this principled projection into the unobserved portion of the state and action space is justified by the maximum entropy method: over all possible distributions taking into account all available information the one with the maximum entropy is expected to be the least wrong.

In the case where some features are not mapped to any observed transitions we will not obtain any information as to their probabilities. We may still, however, place an upper bound on the transitions which use these features by assigning the unknown features a probability of 1.0 for $Pr(\tau)$.

5.4.1 Error Models

mIRL$^*_{/T}$+Int proceeds by first solving the nonlinear optimization to obtain feature probabilities that maximize the total entropy. We then compute $T_I(s, a, \psi(s, a))$ for each state-action pair using Equation 5.1. In order to estimate the complete transition function, we additionally need the probability of reaching unintended states due to action errors.

The mass $1 - T_I(s, a, \psi(s, a))$ could be distributed according to many possible models such as uniformly across all states, to a dedicated error state, or among the intended states that would result due to performing actions other than a from s. While one could be chosen based on side knowledge of the agent being modeled, a better way is to choose the model which makes the observed unintended transitions most likely.

Let $Pr(s'|s, a, m)$ be the distribution over outcome states s' for error model m. The observed expert's trajectories will contain a number of triples $< s^t, a^t, s^{t+1} >$ where $s^{t+1} \neq \psi(s^t, a^t)$. We wish to choose m that maximizes the likelihood of these observed triples:

$$m^{ML} \;=\; \arg\max_m \sum_{<s^t,a^t,s^{t+1}>\in traj\;:\;s^{t+1}\neq\psi(s^t,a^t)} Pr(s^{t+1}|s^t, a^t, m) \tag{5.5}$$

We assume here only a finite number of models are available. An extension to parametric models is straightforward as each model's parameters could be estimated from the available data first before choosing the maximum likelihood parameterized model.

Subsequent to choosing an error model, the full transition function of the other robot, I, is obtained as:

$$\begin{aligned} T_I(s^t, a^t, s^{t+1}) = \;& \delta(s^{t+1}, \psi(s^t, a^t)) \prod_{\tau\in\xi_I^{s,a}} Pr(\tau) + \\ & \left(1 - \prod_{\tau\in\xi^{s,a}} Pr(\tau)\right) Pr(s^{t+1}|s^t, a^t, m) \end{aligned} \tag{5.6}$$

where $\delta(\cdot, \cdot)$ is an indicator function. In Eq 5.6 the probability of reaching s^{t+1} is due to both $T_I(s^t, a^t, \psi(s^t, a^t))$ (if s^{t+1} is the intended next state due to action a^t from state s^t) and error process m. The transition probability distribution due to m is multiplied by the probability mass left over from transitioning to the intended state.

Other alternatives to the maximum likelihood error model include fitting a mixture of models, learning a dynamic Bayesian network from the error transitions, and a recursive approach in which the transition feature technique described in this section is employed to learn the error transition feature probabilities using the residual transitions; this continues until all observed transitions are explained to within acceptable error.

5.5 Convex Approximation with Bernstein Polynomials

Equation (5.3) is non-linear and non-convex due to the presence of monomial constraints. A convex approximation provides valuable scalability benefits as there are many well-known algorithms for quickly solving convex problems. Our strategy for doing so is to find a posynomial approximation for the entropy function which will convert equation (5.3) into a geometric program, which is log-convex. We use a Bernstein polynomial approximation which is given by:

$$B_f^N(x) \;=\; \sum_{\upsilon=0}^{N} f(\tfrac{\upsilon}{N})\binom{N}{\upsilon}x^{\upsilon}(1-x)^{(N-\upsilon)}$$

Bernstein polynomials have a number of properties that make them attractive for our purposes. For instance, they are convex if the original function is convex, only-positive if the original function is only-positive, and demonstrate uniform convergence to the original function as $N \to \infty$. Unfortunately, we require all terms of the approximating polynomial to be positive in order to be a posynomial, and no such solution could be found.

We hypothesized that the downward sloping portion of the entropy function was the cause of the negative terms in the Bernstein approximation. By first negating the entropy function and adding on a constant term we arrive at an always positive convex function. We then restricted the range of $Pr(\tau)$ to $[0.5, 1]$ to remove the downward sloping portion from consideration (we set this new function to zero in the interval [0, 0.5)). Equation (5.3) now becomes:

$$\begin{array}{lll} \min\limits_{\Delta_1,\ldots,\Delta_N} \left(\sum\limits_{n=1}^{N} \sum\limits_{\tau\in\mathcal{T}_n} 0.6932 + Pr(\tau)\ log\ Pr(\tau) + (1-Pr(\tau))\ log\ (1-Pr(\tau)) \right) & & \\ \textbf{subject to} & & \\ \qquad 0.5 \quad \le Pr(\tau) \le 1 \quad \forall\tau & & \\ \prod\limits_{\tau\in\xi_n^{s,a}} Pr(\tau) \quad = q_n^{\psi(s,a)} \qquad \forall\langle s,a\rangle, n = 1\ldots N & & \end{array} \tag{5.7}$$

The Bernstein polynomial for N = 4 using the objective function for a single τ as f is:

$$B_f^N(Pr(\tau)) \approx 0.1307 * 4 * Pr(\tau)^3(1-Pr(\tau)) + 0.693 * Pr(\tau)^4 \tag{5.8}$$

$$= 0.5228\ Pr(\tau)^3 + 0.1702\ Pr(\tau)^4 \tag{5.9}$$

Now our geometric program which approximates (5.7) is:

$$\begin{array}{l} \min\limits_{\Delta_1,\ldots,\Delta_N} \left(\sum\limits_{n=1}^{N} \left(\sum\limits_{\tau\in\mathcal{T}_n} 0.5228 * Pr(\tau)^3 + 0.1702 * Pr(\tau)^4 \right) \right) \\ \textbf{subject to} \\ \qquad 0.5 \quad \le Pr(\tau) \le 1 \quad \forall\tau \\ \prod\limits_{\tau\in\xi_n^{s,a}} Pr(\tau) \quad = q_n^{\psi(s,a)} \qquad \forall\langle s,a\rangle, n = 1\ldots N \end{array} \tag{5.10}$$

Which may be made convex through a change of variables [15]. Note that this approximation is only applicable if all $Pr(\tau)$ are known to be ≥ 0.5. This is the case in our experiments and we expect it to be applicable to many more scenarios; components with such low success rates would be unlikely to be used as they would be extremely unreliable.

5.6 Summary

Estimating the transition function of an optimal agent from observations of its trajectory presents a challenge due to the low likelihood that the agent visits every available state and performs every available action in them. In scenarios that involve occlusion some transitions may never be observed preventing the use of model-free approaches. We developed a model based on transition features determining the probability of the agent reaching an intended next state that may be learned from visible transitions only. Transition features may be derived from side information about the agent in question and in physical agents such as robots may correspond to components used in the associated actions. When combined with an error model our transition feature model allows recovery of the complete transition function.

We showed a simple way of choosing the most likely error model to use given the trajectories available from the agent being observed. Many other error models are possible including a mixture of models learned from the given trajectories or recursively applying the transition feature model.

We present a convex approximation to enhance the scalability of our method to problems with large numbers of constraints. Our approach is to approximate the entropy function with a Bernstein polynomial such that the resulting function is a posynomial, making the approximation a geometric program. Geometric programs are log-convex with a straightforward change of variables.

We empirically validate our model in our robotic patrolling domain in section 7.2.2 as well as demonstrate the performance benefits of our approximation.

Chapter 6

Learning in the Presence of Multiple Experts

In this chapter we will focus on the problem of a robotic learner, which we label L to avoid confusion, attempting to perform IRL in the presence of multiple, interacting experts. The interaction here is key: if the experts are non-interacting this problem reduces to solving two IRL problems separately. We will assume that the experts, being physically embodied agents that the learner is observing, are operating in a shared space and can only interact when in close proximity. We do not place any restrictions on which state they may interact in.

When multiple experts are available, a learner may desire to learn from some subset of them. However, if the experts are interacting then the action each chooses, state transition probabilities, and/or reward each receives could depend upon the joint state and action of all agents involved. It is therefore necessary to account for the behavior of all agents present - this may involve solving a much larger joint-IRL problem that encompasses all experts.

Joint MDP

In fact, a straightforward way of proceeding is to model the complete system with a single, much larger MDP whose states are the joint states of all experts, actions are the Cartesian products of all agents' individual actions, and transition and reward functions depend upon the state and action of all agents. States in which there is interaction between agents could be modeled by using appropriate transition probabilities assuming the agents' rewards are agnostic to their interaction, i.e. the interactions are sparse and not a critical portion of each experts' task. An example of this is two robots approaching each other in a narrow hallway with their goals being arriving at the opposite ends; the temporary slowdown they experience as they sidestep each other serves only as a small distraction.

This straightforward approach, while capable of being solved with existing algorithms, has a few significant drawbacks. The size of this joint MDP could become very large as the transition function must now consider the effects of performing all joint-actions from every possible joint-state which expands with every agent added. As a result the solving time increases significantly which exacerbates the solving time of IRL since multiple MDP's must generally be solved (depending upon the method used) during the search for the correct reward parameters.

A more serious problem with this approach is if the learning agent loses sight of one of the experts due to the presence of occlusion. Then the joint state would only be partially observed, which violates the requirements of IRL. While partial observation of states is a problem addressed in [26] we take a different approach.

6.1 IRL for Multiple Mobile Robots

Due to the limitations of joint MDPs we described above we develop an alternative approach in which we will continue to model each robot as a separate, individual MDP. The behavior of the robots during an interaction will be modeled separately and overrides the behavior proscribed by these MDPs for the duration of the interaction. A non-linear program that combines the one in (2.18) for each mobile robot and is used for learning the policies of all robots is given by:

$$\max_{\Delta_1,\ldots,\Delta_N} \quad -\sum_{I=1}^{N} \sum_{\pi_I \in \Pi_I} Pr(\pi_I)\ logPr(\pi_I) \tag{6.1}$$

subject to

$$\begin{aligned}
&\sum_{\pi_1 \in \Pi_1} Pr(\pi_1) = 1\ ,\ \ \ldots\ ,\ \sum_{\pi_N \in \Pi_N} Pr(\pi_N) = 1 \\
&\sum_{\pi_1 \in \Pi_1} Pr(\pi_1) \sum_{s \in Obs(S)} \mu_{\pi_1}(s)\phi_k(s, \pi_1(s)) = \hat{\phi}_{k,1} \qquad \forall k \\
&\vdots \\
&\sum_{\pi_N \in \Pi_N} Pr(\pi_N) \sum_{s \in Obs(S)} \mu_{\pi_N}(s)\phi_k(s, \pi_N(s)) = \hat{\phi}_{k,N} \qquad \forall k
\end{aligned} \tag{6.2}$$

This program may be solved by doubling the approaches described in section 2.5, when IRL* from section 4.2 is utilized in this way we label this simple multi-agent extension as mIRL*.

6.1.1 Modeling the Interaction

The approach described above requires that any interaction between the robots be sparse and immaterial to the task they separately perform. This implies that the actions taken during interaction, which may involve some form of coordination, be chosen with the goal of resolving the interaction quickly and returning to their solitary behaviors. In our scenario, the robots must coordinate their actions when they are in close proximity to minimize the disturbance to their patrols.

At interaction states such as the one in Fig. 6.1, the subject robot L models the robots as playing a game. The strategies of this game correspond to the Cartesian product of the actions in each robot's MDP. We assume the robots solve the game before their first interaction, arriving at a solution in the form of a Nash equilibrium. This equilibrium proscribes each robot's behavior during the interaction and is not changed while L is observing.

In the event L observes an interaction taking place it simply constructs a game in which the observed behavior is the sole equilibrium. Alternatively, domain specific side information, such as knowledge of the interaction transition probabilities, can be encoded in the game as strategy payoffs. The solution of this game may result in a single Nash equilibrium, but in general may

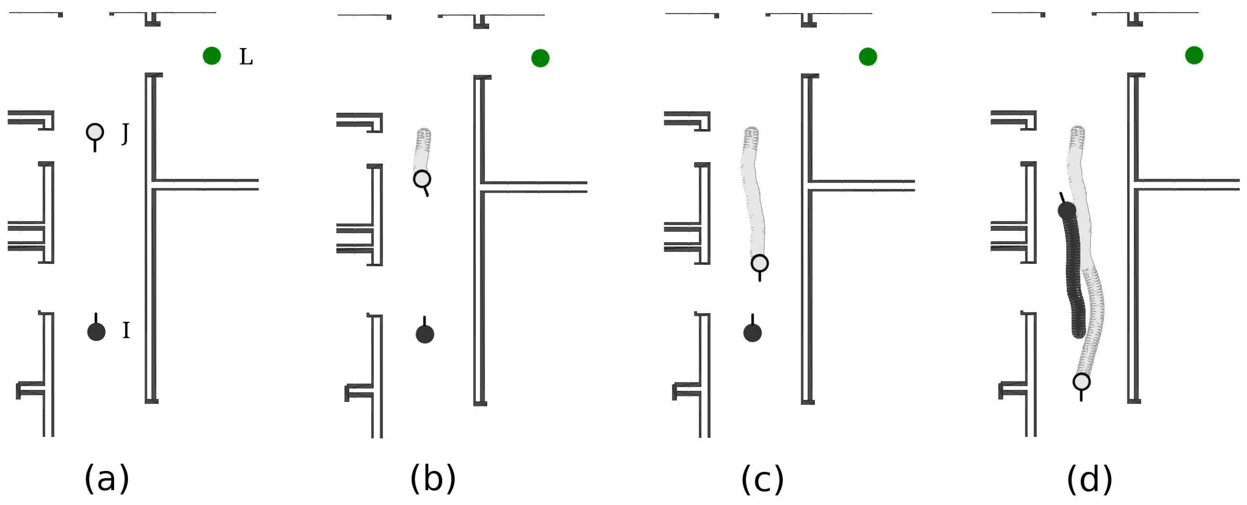


Figure 6.1: *(a)* Patrolling robots I and J approach each other, entering an interaction state *(b)* The two robots abandon their MDP-based policies and begin executing the actions dictated by the interaction game equilibrium *(c)* I stops while J sidesteps slowly. *(d)* The interaction behavior is now completed and the two robots return to the behavior specified by their individual policies.

admit a number of possible strategies as equilibria. In Table 6.1, we show an example game in the context of our patrolling application domain, which has five profiles in equilibria each representing a possible way of resolving the task of safely moving past each other when the robots approach each other from opposite directions.

Let us focus on the case where there is one interaction Nash equilibrium. As sparse interactions alter the behavior of the robots the equilibrium must be considered during IRL in order to accurately learn the policy of each robot. Otherwise the policy learned will likely be different from the actual policy of the observed mobile robot as it falsely attributes the interaction behavior to attempting to maximize its reward received from the MDP's reward function. Let $\sigma^e = \langle \sigma_1, \ldots, \sigma_N \rangle$ be an equilibrium with σ_I, $I = 1, \ldots, N$ denoting each robot's action that is in equilibrium, respectively. As *interactions impact the state-visitation frequencies of the robots*, we decompose equation 2.8 into a piece-wise function for each robot, $I = 1, \ldots, N$, as,

$$\mu^e_{\pi_I}(s) = \begin{cases} \mu^0_{\pi_I}(s) + \gamma \sum_{s'} T(s, \pi_I(s), s') \mu^e_{\pi_I}(s') & \text{if } s \text{ is not an interacting state} \\ \mu^0_{\pi_I}(s) + \gamma \sum_{s'} T(s, \sigma_I(s), s') \mu^e_{\pi_I}(s') & \text{if } s \text{ is an interacting state} \end{cases} \tag{6.3}$$

We may then replace μ_{π_I} in equations 6.2 with the above equation.

$$\sum_{\pi_I \in \Pi_I} Pr(\pi_I) \sum_{s \in \text{Obs}(S)} \mu^e_{\pi_I}(s) \phi_k(s, \pi_I(s)) = \hat{\phi}_{k,I} \quad \forall k \tag{6.4}$$

Without occlusion, L may simply note the proportion of timesteps the robots interacted in a given state in the observed trajectory. Then a corresponding proportion of iterations of equation 6.3

	Sidestep	Turn left	Turn right	Turn around	Stop
Sidestep	**5,5**	5,1	5,1	**5,5**	**5,5**
Turn left	1,5	1,1	1,1	1,2	1,0
Turn right	1,5	1,1	1,1	1,2	1,0
Turn around	**5,5**	2,1	2,1	2,2	2,0
Stop	**5,5**	0,1	0,1	0,2	0,0

Table 6.1: One example of a game that models the interaction of two robots (row player is I, column player is J) attempting to pass each other in a narrow hallway. Nash equilibria (shown in bold) are the possible ways of resolving the interaction efficiently. The payoffs displayed here are not necessarily representative of those the two robots actually receive and are chosen by L using side information about the scenario, in this case the expected transition outcome of the joint action.

are treated as interacting for the corresponding state. In the presence of occlusion, however, this equation may not be computable as the proportion of interacting states may be unknown to L.

Instead, we empirically compute the state-visitation frequencies by projecting in the full environment the policy under consideration for each robot for a large number of time steps utilizing the equilibrium behavior, σ^e, when the robots interact. The state visitations in the projections are accumulated to obtain an approximation of the state-visitation frequency, $\mu^e_{\pi_I}$. Note that this technique renders the program non-convex, as the state visitation frequency of a policy for one robot depends in part on the policy chosen for the other robot the constraint is no longer linear in $Pr(\pi_I)$ (see section 2.4.1). The degree of this non-convexity is dependent upon the sparsity of interaction.

We refer to this game-theory inspired extension of mIRL* as mIRL*+ne

6.1.2 Multiple Equilibria

Note that while the interaction game modeled by L may admit multiple equilibria, the robots pick one to resolve their interaction. In the case where L is unable to determine the equilibrium used by the robots through observation or side information how can L determine which equilibrium behavior was used?

Our approach is to retain each possible equilibrium behavior and weight it based on how close the state visitation frequency resulting from using each behavior matches observations. If $\mu^e_{\pi_I}$ is the empirical state-visitation frequency when equilibrium, σ^e, is used at the interaction and π_I elsewhere, and $\hat{\mu}_I$ is the state-visitation frequencies from the observed trajectories, then L weights the potential of this interaction behavior as an inverse function of the difference,

$$\omega^e \propto e^{-\sum\limits_{s \in \text{Obs}(S)} |\mu^e_{\pi_I}(s) - \hat{\mu}_I(s)|} \tag{6.5}$$

We calculate ω for each equilibrium behavior, and then the ω vector is normalized to sum to 1. Finally, we modify the constraint of (6.4) with a convex combination of each equilibrium-based interaction. Let $\mathcal{E}$ be the set of all equilibria behaviors, the constraint becomes:

$$\sum_{\pi_I \in \Pi_I} Pr(\pi_I) \sum_{\sigma^e \in \mathcal{E}} \omega^e \sum_{s \in \text{Obs}(S)} \mu^e_{\pi_I}(s)\phi_k(s, \pi_I(s)) = \hat{\phi}_{k,I} \quad \forall k \tag{6.6}$$

During each solving iteration of the IRL problem a new set of reward weights are generated and an optimum policy found for each robot. $\mu^e_{\pi_I}$ is found for each σ^e and the ω weights are then recomputed. Initially we expect weights may be nearly uniform because the learned policies do not correctly model the observed behavior for the most part. Eventually, as the policies improve, projecting the true interaction behavior begins to matter more and the relative weights between equilibria diverge. We explore the dynamics of the evolution of ω during solving in section 7.1.5. This new algorithm for multiple interacting robots with unknown interaction behavior is labeled mIRL*+Int.

6.2 Solving and Approximation

Our experiments will take place under occlusion and, as described in section 4.2, the complete gradient of the maximum entropy problem will be unavailable. Our approach to solving used here is to take the Lagrangian function, $\mathcal{L}(Pr, \eta, \boldsymbol{\theta})$ where η and $\boldsymbol{\theta}$ are the Lagrange multipliers and obtain its partial derivative w.r.t. $Pr(\pi_I)$ and $\boldsymbol{\theta}$ as:

$$\frac{\partial \mathcal{L}}{\partial Pr(\pi_I)} = \sum_{i=1}^{\mathcal{E}} \omega^{e_i} \cdot \sum_{s \in \text{Obs}(S)} \mu_{\pi_I}(s) \sum_k \theta_k \phi_k(s, \pi_I(s)) - logPr(\pi_I) + \eta - 1 \tag{6.7}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \sum_{\pi_I \in \Pi_I} Pr(\pi_I) \sum_{i=1}^{\mathcal{E}} \omega^{e_i} \cdot \sum_{s \in \text{Obs}(S)} \mu^{e_i}_{\pi_I}(s)\phi_k(s, \pi_I(s)) - \hat{\phi}_k \tag{6.8}$$

$\mathcal{L}(Pr, \eta, \boldsymbol{\theta})$ may have its optimal solution as a saddle point, however, many numerical optimization techniques such as hill climbing and gradient descent are instead designed to find local minima or maxima. We therefore follow an approach to ensure that the optimal solution resides at a minima by modifying the objective function of the relaxed Lagrangian. The objective function becomes $\sqrt{\frac{\partial \mathcal{L}}{\partial Pr(\pi_I)}^2 + \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}^2}$.

Our experiments have limited execution time and therefore the learning agent has at most a few minutes to perform IRL. As summation over the entire space of policies for all agents is time prohibitive we adopt an approximate solution that avoids the sum. To this end we consider only the optimum policy for a given set of weights and approximate equation 6.7 with:

$$\frac{\partial \mathcal{L}}{\partial Pr(\pi_I)} \approx \sum_{(s,a) \in traj} Q_{\pi_I}(s, a) - V_{\pi_I}(s) \tag{6.9}$$

where, $Q_{\pi_I}(s, a)$ denotes the action value of performing the observed action, a, from state, s, and following π_I thereafter; and $V_{\pi_I}(s)$ is the optimal value of (candidate) I's policy, π_I. Note that $Q_{\pi_I}(s, a) - V_{\pi_I}(s) = 0$ when $a = \pi_I(s)$. In other words, the value difference is 0 when the action from the policy under consideration matches the observed action. In this case we have found a reward function that leads to the MDP whose solution matches observations. However, it is possible that some other action could also result in a Q-value that is the same as the optimal value. Therefore, the above substitution is an approximation. If the observed action at s does not match the policy, the difference is negative increasing in magnitude as the action taken is lesser valued by the candidate reward function. This then accumulates over the trajectory. Because of these properties, we also use this measure as an approximation for $logPr(\pi_I)$ in equation 6.8 and subsequently drop the summation over all policies.

6.3 Multi-Agent Hidden Data EM

With the goal of improving the capabilities of our multi-agent model when under occlusion we now apply the sparse interaction model described in this chapter to our *Hidden Data EM* method from section 4.3. To make use of the derivation in section 4.3.1 we assume the interaction behavior between all agents is known. Then with our sparse interaction model in mind we modify the definition of $\mathbb{X}$: each trajectory is now the combined trajectories of all observed agents. Note that the state of an agent I at time t is dependent upon all agents' states and agent I's action at time $t-1$. Likewise the action chosen at time t is dependent upon all agents' states at time $t-1$. These new, interacting transition functions and policies are given by:

$$T^I(S^I, S^J, A^I, S'^I) = \begin{cases} (1-\iota)\ \delta(S^I, S'^I) + \iota\ T^I(S^I, A^I, S'^I) & \text{if interacting} \\ T^I(S^I, A^I, S'^I) & \text{otherwise} \end{cases}$$

$$Pr(A^I|S^I, S^J) = \begin{cases} \delta(A^I, A^I_{int}) & \text{if interacting} \\ Pr(A^I|S^I) & \text{otherwise} \end{cases}$$

Where ι is the probability of the interaction completing (set to 0.33 in our experiments) which models interacting transitions as having larger uncertainty than single agent transitions, δ is an indicator function which returns 1 if its parameters are equal and zero otherwise, and A^I_{int} is the action taken by agent I to resolve the interaction.

6.3.1 Scaling Multi-Agent Hidden Data EM

So long as $|\mathbb{Z}|$ remains small $Pr(Z|Y;\boldsymbol{\theta}^{(t)})$ may be calculated by enumerating every possible way in which a given Y may be completed and calculating the probability of each resulting trajectory $Pr(X)$. Unfortunately this method quickly becomes intractable as in the worst case $|\mathbb{Z}| = |S_{occ}||A|^t$ where t is the number of timesteps in which the expert was not observed, requiring $O(|S_{occ}||A|^t)$ time to calculate where $S_{occ} = S - Obs(S)$. An alternative to this brute-force method is the Forward-Backward algorithm as given in Russell and Norvig [50] pp 546. This smoothing algorithm makes use of dynamic programming to calculate the posterior marginal probabilities of the hidden states of a Markov chain and has a time complexity of $O(t(|S_{occ}||A|)^2)$.

The forward-backward algorithm requires an observation received at each timestep which is emitted by the state of the system stochastically. As the timesteps in the occluded trajectories described in this paper alternate between fully observable and fully hidden we provide a mapping to an observation model to ease implementation.

Define $o^{sa} \in O^{SA}$ to be an observation of a state s and action a. O^{SA} is the set of all possible such observations and is of size $|Obs(S)| \times |A|$. $Pr(o^{sa}|s', a') = 1$ if $s' = s$ and $a' = a$, else

0. One additional observation, o^{occ}, is needed which is equally emitted by all occluded states, $Pr(o^{occ}|s', a') = 1$ if $s' \in S_{occ}$ else 0.

Now we may map each trajectory to a simple Markov chain by creating a single random variable whose possible states are $S \times A$ and assigning the appropriate observation at each timestep (from O_{SA} when the expert agent was observed and o^{occ} when it was occluded).

The forward and backward messages for a single agent are given by:

$$\begin{aligned} f_{t+1}(s', a') &= Pr(o_{t+1}|s', a') \sum_{s,a} Pr(s', a'|s, a) f_t(s', a') \\ &= Pr(o_{t+1}|s', a') \sum_{s,a} T(s, a, s') Pr(a'|s') f_t(s, a) \end{aligned} \tag{6.10}$$

$$\begin{aligned} B_{t+1}(s, a) &= \sum_{s',a'} Pr(o_{t+1}|s', a') B_{t+2}(s', a') Pr(s', a'|s, a) \\ &= \sum_{s',a'} Pr(o_{t+1}|s', a') B_{t+2}(s', a') T(s, a, s') Pr(a'|s') \end{aligned} \tag{6.11}$$

Multi-agent trajectories require increasing the space of observations and states in the Markov chain appropriately, expanding the backward and forward messages to incorporate the state and actions of all agents, and replacing $T()$ and $Pr(a|s)$ used in the above equations with equations 6.3 and 6.3 (one for each agent). For completeness, these are:

$$\begin{aligned} f_{t+1}(s^{I'}, a^{I'}, s^{J'}, a^{J'}) = \quad & Pr(o_{t+1}|s^{I'}, a^{I'}, s^{J'}, a^{J'}) \sum_{s^I, a^I, s^J, a^J} T^I(s^I, s^J, a^I, s^{I'}) \times \\ & T^J(s^J, s^I, a^J, s^{J'}) Pr(a^{I'}|s^{I'}, s^{J'}) Pr(a^{J'}|s^{J'}, s^{I'}) f_t(s^I, a^I, s^J, a^J) \end{aligned} \tag{6.12}$$

$$\begin{aligned} B_{t+1}(s^I, a^I, s^J, a^J) = \quad & \sum_{s^{I'}, a^{I'}, s^{J'}, a^{J'}} Pr(o_{t+1}|s^{I'}, a^{I'}, s^{J'}, a^{J'}) B_{t+2}(s^{I'}, a^{I'}, s^{J'}, a^{J'}) \times \\ & T^I(s^I, s^J, a^I, s^{I'}) T^J(s^J, s^I, a^J, s^{J'}) Pr(a^{I'}|s^{I'}, s^{J'}) Pr(a^{J'}|s^{J'}, s^{I'}) \end{aligned} \tag{6.13}$$

In multi-agent settings the number of possible states and actions in occluded timesteps grows rapidly with the number of agents as for any given state-action pair for one agent there are $|S_{occ}||A|$ possible state-action pairs for the other. Then $|\mathbb{Z}| = (|S_{occ}||A|)^{t_1} \times (|S_{occ}|^2|A|^2)^{t_2} \ldots$ where t_1 is the number of timesteps in which one agent was occluded, t_2 is the number of timesteps where 2 agents were occluded and so on. As a result for any non-trivial multi-agent scenario these exact methods become prohibitively expensive.

Gibbs Sampling

As described in section 4.3.2 Gibbs sampling may be used as part of a Monte-Carlo Expectation-Maximization approach to approximation the distribution over trajectories in the E step of Hidden

Data EM. We now expand this method to our multi-agent scenario; each node in a multi-agent Z may be sampled by:

$$\begin{aligned}
Pr(s_t^I | MB(s_t^I)) = \; & \eta T^I(s_{t-1}^I, s_{t-1}^J, a_{t-1}^I, s_t^I) \times \\
& T^I(s_t^I, s_t^J, a_t^I, s_{t+1}^I) T^J(s_t^J, s_t^I, a_t^J, s_{t+1}^J) Pr(a_t^I | s_t^I, s_t^J) \times \\
& Pr(a_t^J | s_t^J, s_t^I) \\
Pr(a_t^I | MB(a_t^I)) = \; & \eta' T^I(s_t^I, s_t^J, a_t^I, s_{t+1}^I) Pr(a_t^I | s_t^I, s_t^J)
\end{aligned}$$

if the sampled node is a state or action respectively, η and η' are normalizers.

In our implementation, we initialize the nodes in Z in timestep order by first sampling from $T(s_{t-1}, a_{t-1}, s_t)$ and then $Pr(a_t|s_t)$. As full trajectories are produced during sampling we calculate the feature expectations of each one and update the mean feature expectations seen so far. Sampling is stopped once the change in the mean feature expectations is below a threshold for the last 20 samples. This mean is then added to a mean-of-means and Gibbs sampling and repeated until the mean-of-means has converged.

Blocked Gibbs Sampling

To improve the convergence rate of Gibbs sampling we may employ blocking. Here variables are grouped into blocks and the joint distribution of all nodes within the block are sampled as one unit.

We may easily calculate the joint distribution when a block size of only one timestep of the trajectory is used. However, if block sizes incorporate multiple timesteps the joint distribution becomes difficult to calculate due to its size. We use a forward-backward algorithm to calculate the distribution in this case. As multi-agent trajectories greatly increase the size of the sample space we develop variants of the above sampling methods in an attempt to improve converge rates:

- **Blocked Gibbs** - Samples the state and action of one agent as a block, alternating agents, proceeding in timestep order.
- **Multi-Agent Blocked Gibbs** - Samples the joint state and action of all agents at a given timestep of the joint trajectory, reduces to **Blocked Gibbs** if only one agent is occluded.
- **X Timestep Blocked Gibbs** - Samples X timesteps of a single agent, alternating agents. Uses a single-agent forward-backward algorithm to calculate the joint distribution to be sampled from.

A further method **X timesteps, All Agents Blocked Gibbs** which sampled X timesteps of all agents was also developed, however, using the multi-agent forward-backward algorithm was found to be prohibitively expensive in time. The Markov blankets of all the blocking methods described here are the states and actions in the timesteps surrounding them - these nodes are treated as perfect observations in the forward-backward algorithm, an uninformative dummy observation which weights all occluded states and all actions equally is used for the timesteps within the block.

6.3.2 M-Step Methods

We minimize the dual of equation 2.13 using the adaptive unconstrained exponentiated gradient descent algorithm [54] with variance bounds. The gradient involves a summation over all possible trajectories which is of size $(|S| \times |A|)^t$, a very large set. With the goal of speeding up the computation we avoid this summation by first following the approach given in Ziebart et. al. [61] and express the learner's expected features in terms of the calculated state visitation distribution at each timestep:

$$\begin{aligned}
\nabla\Theta &= \sum_X Pr(X) \sum_{(s,a)\in X} \phi(s,a) - \hat{\phi}^{Z|_Y} \\
&= \sum_s \mu(s) \sum_a Pr(a|s)\phi(s,a) - \hat{\phi}^{Z|_Y} \\
&= \sum_s \sum_t \mu_t(s) \sum_a Pr(a|s)\phi(s,a) - \hat{\phi}^{Z|_Y}
\end{aligned} \tag{6.14}$$

Where $\mu(s)$ is the state visitation frequency, $\mu_t(s)$ is the state visitation distribution at time t, $\mu_t(s) = \sum_{s'} \sum_a Pr(s|s',a)Pr(a|s')\mu_{t-1}(s')$, $\mu(s) = \sum_t \mu_t(s)$ [61], and $Pr(a|s)$ is calculated using soft-max value iteration [62]. We next decompose this gradient along timesteps to use an online stochastic gradient descent approach:

$$\begin{aligned}
\nabla\Theta = &\sum_s \sum_t \mu_t(s) \sum_a Pr(a|s)\phi(s,a) - \\
&\sum_{Y\in\mathcal{Y}} \tilde{Pr}(Y) \sum_{Z\in\mathbb{Z}} Pr(Z|Y) \sum_{\langle s,a\rangle\in Y\cup Z} \phi(s,a) \\
= &\sum_s \sum_t \mu_t(s) \sum_a Pr(a|s)\phi(s,a) - \tilde{Pr}(s,a;t)\phi(s,a) \\
= &\sum_t \sum_s \sum_a (\mu_t(s)Pr(a|s) - \tilde{Pr}(s,a;t))\phi(s,a) \\
\nabla\Theta_t = &\sum_s \sum_a (\mu_t(s)Pr(a|s) - \tilde{Pr}(s,a;t))\phi(s,a)
\end{aligned} \tag{6.15}$$

where $\tilde{Pr}(s,a;t)$ is the empirical distribution of state-action pairs at time t of all trajectories calculated from the E step. Notice that as t grows the calculation of μ_t begins to dominate the run time due to its recursive nature. One approach to optimization is to utilize the previous timestep's empirical state distribution as μ_{t-1} as shown below.

$$\begin{aligned}
\nabla\Theta_t = &\sum_s \sum_a (Pr(a|s)\mu_t(s) - \tilde{Pr}(s,a;t))\phi(s,a) \\
\nabla\Theta_t = &\sum_s \sum_a (Pr(a|s) \sum_{s'} \sum_{a'} Pr(s|s',a')Pr(a'|s')\mu_{t-1}(s') - \tilde{Pr}(s,a;t))\phi(s,a) \\
\nabla\Theta_t \approx &\sum_s \sum_a (Pr(a|s) \sum_{s'} \sum_{a'} Pr(s|s',a')Pr(a'|s') \sum_{a''} \tilde{Pr}(s',a'';t-1) - \tilde{Pr}(s,a;t))\phi(s,a)
\end{aligned} \tag{6.16}$$

Intuitively, this approximation first projects forward one timestep from the expert's previous state distribution using the learned stochastic policy and then compares the resulting state-action distribution with the observed state-action distribution of the expert at the current timestep. This is in keeping with the approach of stochastic gradient descent where only the local values of the gradient are considered, but may under-emphasize differences in state-visitation that results from the use of wrong weights, for instance if the expert traverses through a portion of the state space the current trajectory distribution deems unlikely this will be ignored except for the first timestep where the deviation occurs. This is compensated for in SGD by repeatedly iterating through the observed trajectories until convergence of the weights.

In our experiments the learner receives a single trajectory from each expert making it unlikely that the empirical state distribution is adequately estimated. We therefore further optimize this step by setting $\mu_t(s) = \sum_{a'} \tilde{Pr}(s, a'; t)$. This allows us to ignore the computation of the state-visitation distribution completely but has a side effect of disallowing the use of state-only features (we do not utilize these in our experiments). We perform stochastic gradient descent for 10,000 iterations and update $Pr(a|s)$ after blocks of 10 iterations.

6.4 Summary

In this chapter we presented a sparse interaction model that may be applied to multiple interacting robots when they are nominally performing separate behaviors except for brief interactions. This model is based on game theory and assumes the robots have played the game and arrived at some Nash equilibrium (possibly unknown to the learner) which dictates their interaction behavior prior to the learner observing their behavior.

We show how our model may be applied to mIRL* to remove the persistent noise caused by the interaction, when the interaction behavior is known we call this simple extension mIRL*+ne. We demonstrate how the interaction behavior may be searched for during IRL by weighting each possible Nash equilibrium according to how close its use brings the calculated feature expectations to the expert's, a procedure we call mIRL*+Int.

We extend *Hidden Data EM* in a straightforward way to our sparse interaction multi-robot scenarios. We note the large size of the hidden variable space makes exact solutions intractable for all but trivial contexts and describe a number of variants of Gibbs sampling to alleviate this issue. Our methods are experimentally validated in sections 7.1.4 (mIRL*) and 7.2.2 (multi-agent *Hidden Data EM*).

Chapter 7

Experiments with Penetrating a Robotic Patrol

We apply the techniques developed in chapters 5 and 6 to a problem domain in which an attacking robot L is attempting to move through a perimeter patrol of two other robots, I and J, without being detected. L has obtained a vantage point from which it can observe the other two robots without being detected, however, once it starts moving it will be detected if a patroller is nearby and facing it. The patrollers move in a shared space and interact when approaching each other to avoid collisions, their interaction mode is not known to the attacker ahead of time. L is required to 1) observe the patrollers without moving to gather data for IRL 2) use IRL to recover the patrollers' reward function and subsequently the policy they are using 3) using the found policies predict the future path of the patrollers 4) plan a route through the space that avoids the predicted path of the patrollers while still arriving at the goal state.

7.1 Evaluation of **mIRL*+Int**

We use two measures to quantify the ability of IRL to learn the behavior of the patrollers. The first is **Learned Behavior Accuracy** which, as described in section 2.6, is the proportion of states in which the optimum policy for the learned reward function gives the correct action. This gives a measure of the prediction accuracy to be expected from a given technique. The second measure is the **Success Rate** of L as a proportion of penetration attempts that result in L reaching its goal without being detected. This measures all aspects of the experiment but suffers from the possibility that L could fail to learn the patrollers' policies accurately but still reach its goal undetected by chance.

7.1.1 Penetrating a simple multi-robot patrol

Each patroller is modeled as acting according to the output of its own MDP with interactions resolved by an interaction game Nash equilibrium. The states of this MDP are the cell decomposition of the patrolled area (x, y) and an additional discretized orientation ψ. Each patroller may take one of 5 actions: Move Forward, Stop, Turn Left, Turn Right, and Turn Around. The transition function models the probability of any action succeeding at 66% with the remaining probability mass distributed to the other actions.

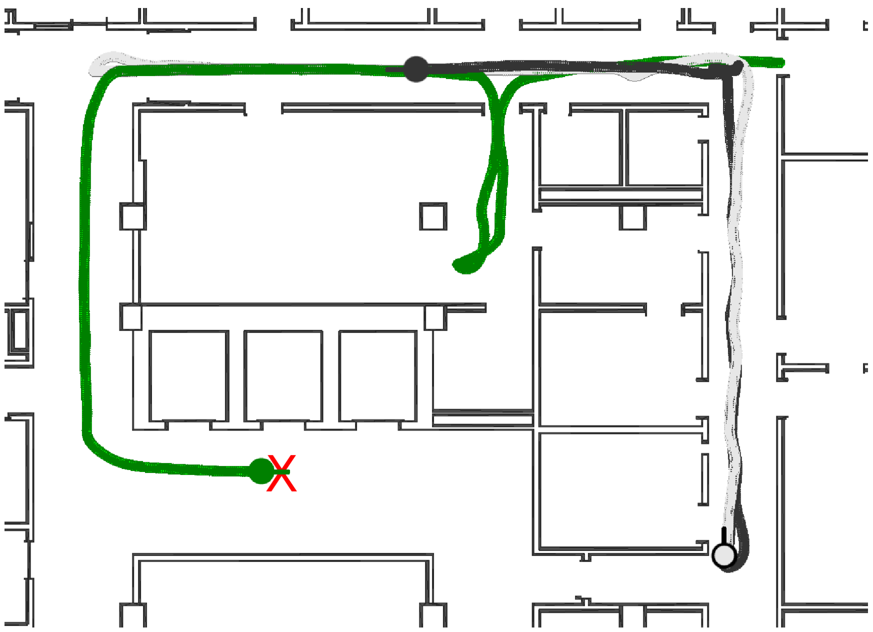


Figure 7.1: Successful penetration by L of our larger environment. In this particular instance, L chose to enter a room briefly to allow I to move past before exiting and moving towards the goal.

As L is expected to move through the same space as the patrollers its MDP is similar to theirs with the important addition of a discretized time variable. This 4 dimensional MDP is needed because the states L must plan for are dynamic as the patrollers are constantly in motion. After the policies for each patrollers have been learned, L jointly projects forward in time, starting from the last position each patroller was observed at, to arrive at a prediction for the future positions of each patroller. These positions are noted in L's MDP and any states which are visible from a patroller's position at a given time step receive a negative reward. The goal state at all time steps is given a positive reward and the MDP is solved optimally. L then searches for a positive value at any time for its starting position and if found there must be a path to the goal that avoids detection starting at that time. L waits until that time has arrived and begins following its policy. In the event that no positive value is found the positions of the patrollers are updated with any new observations and the process is repeated until one is found or the experiment runs out of time (set to a max of 20 minutes). We show an example of a successful run in figure 7.1.

7.1.2 Maps

We test two maps, both of indoor office environments, in our simulation experiments. The smaller of these is also used in our physical experiments. As can be seen in figures 7.2 and 7.3 L is located in a central location from which it can gather observations and the patrollers move in a cyclic fashion between their pre-determined way-points.

There are a number of possible interaction behaviors that each patroller could take when they approach each other. These include: stopping, turning left or right, turning back around, or slowly side-stepping the other robot. L is unaware what the choices of each robot are, however, it can eliminate certain combinations as not resolving the interaction (such as Stop, Stop). Interactions are time extended to 3 timesteps to model the slower movement of the patrollers during interaction.

Each robot in our simulations and physical experiments is a Turtlebot equipped with a video camera and laser scanner. In physical experiments the robots identify each other by color and are autonomous relying on adaptive Monte-Carlo localization and ROS's Move_base system for navigation.

In the smaller environment the reward functions of both robots are a linear combination of two types of features:

1. *Has moved*, which returns 1 if the action causes the patroller to leave its current location, otherwise 0; and
2. *Turn around at state, s*, which returns 1 if the robot turns around 180° at the location given by s, otherwise 0.

Reward functions of the patrollers in the larger environment include the following binary feature functions as their trajectories may go through rooms in the large hallways as well.

1. *Has moved*, which returns 1 if the action causes the patroller to leave its current location, otherwise 0; and
2. *Turn around in the hallway*, which returns 1 if the robot turns around 180° at a location that is in the hallways, otherwise 0.
3. *Enter room*, which returns 1 if the patroller enters a room from one of the hallways, otherwise 0;
4. *In room*, which returns 1 if the patroller is in any of the rooms in the map, otherwise 0; and
5. *Leave room*, which returns 1 if the patroller leaves a room to enter a hallway.

The true reward function penalizes turning around in the middle of the hallway or entering a room and rewards turning at the goal states. As a result, the optimum policies of the patrollers generates trajectories that move through the hallways, turn around at the ends, and move back. The Nash equilibrium chosen by the patrollers to resolve their interaction game is (Side Step, Stop).

We make use of a number of baselines to evaluate the approach introduced in this section 6.1.2 (labeled mIRL*+Int). mIRL* uses the approach described in section 6.1 which handles occlusion but does not model interaction between the robots. As an upper bound on the success rate we reveal the true policies and NE of the patrollers to L, this *Known Policy* approach evaluates the prediction abilities of L only. Finally, a *Random* approach, in which all observations are ignored and a random time is waited before L begins its attack is used to provide a lower bound on the success rate.

7.1.3 Simulation Results

We begin by evaluating the Learned Behavior accuracy of both learning algorithms as a function of the degree of observability. This data is gathered from 400 simulated runs of the larger environment and 200 from the smaller, leading to standard error from 0.022 (lowest visibility smaller map) to 0.0031 (highest visibility larger map) - in the physical experiments it is impossible to change the degree of observability and corresponds to the lowest measure here. As can be seen in figure 7.4 the accuracy increases with observability, as expected. Note, however, the divergence between the two

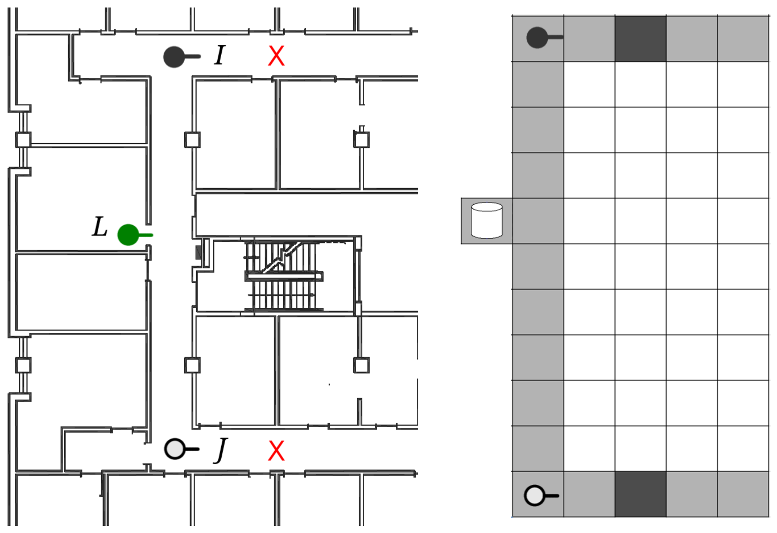


Figure 7.2: The smaller environment and the corresponding MDP state space for our experiments. The two goal cells are colored black and the white cells denote occupied locations. L's starting location is shown while I and J move in a cycle between the two goal cells.

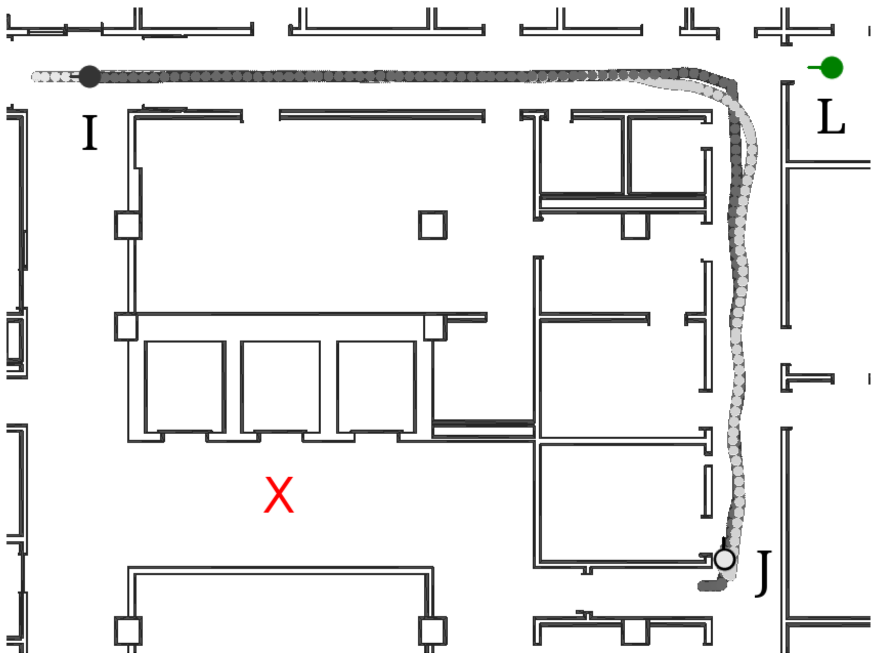


Figure 7.3: Trajectories of patrollers, I and J, in the simulated hallways of a building for our larger environment. Subject robot L is tasked with reaching the goal state at X undetected from its starting position.

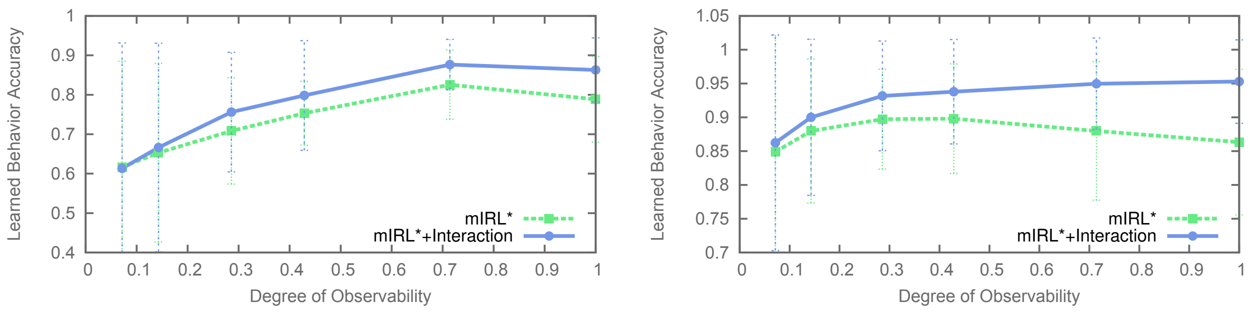


Figure 7.4: Learned behavior accuracy of our approach measured for different occlusion rates. The vertical bars represent one standard deviation from the mean, standard error is between 0.022 (lowest visibility smaller map) to 0.0031 (highest visibility larger map). Smaller map on the left and larger on the right.

methods indicating mIRL*'s failure to take the interaction into account has a detrimental effect on its ability to learn the reward functions. This effect is increased with observability and highest with no occlusion as it becomes more likely that interactions between robots are directly observed and it becomes increasingly unable to explain the observations with separate MDPs alone. The accuracy difference between these two methods is statistically significant (Student's paired, two-tailed t-test, $p < 0.025$ for both environments).

Next we evaluate the success rate of each algorithm as a function of three factors: the Learned Behavior accuracy, degree of observability, and the observation time L was given to observe the patrollers. As can be seen in figure 7.5 (a), Learned Behavior accuracy positively correlates with the success rate, particularly for mIRL*+Int. Notice, however, that the success rate decreases for mIRL* as the Learned Behavior accuracy increases to 100%, indicating the importance of modeling the patroller interaction on L's ability to predict where they will be in the future accurately. 7.5 (b) and (c) show that the success rate generally increases mIRL*+Int with more observations available but this is not necessarily true for $mIRL^*$. Also notice both methods are well bounded by *Known Policy* and *Random*, and that given enough observations mIRL*+Int performs on par with the upper bound.

7.1.4 Physical Robot results

As it was challenging to vary the degree of observability in our physical experiments, we report the success rate as a function of observation time. We performed 10 runs of each observation time setting and noted the success rate produced. Notice in figure 7.6 the success rate starts low but improves with more observations and also the close agreement with the success rate produced by simulations for equivalent parameters. Importantly, the difference between simulation and physical results is not statistically significant, as we should expect.

Finally, figure 7.7 shows photos of L, I and J at various points of our experiment. In particular, I and J are shown in an interacting state. We also illustrate L's visibility during the experiment and L exiting its vantage point as it makes its way towards the goal state.

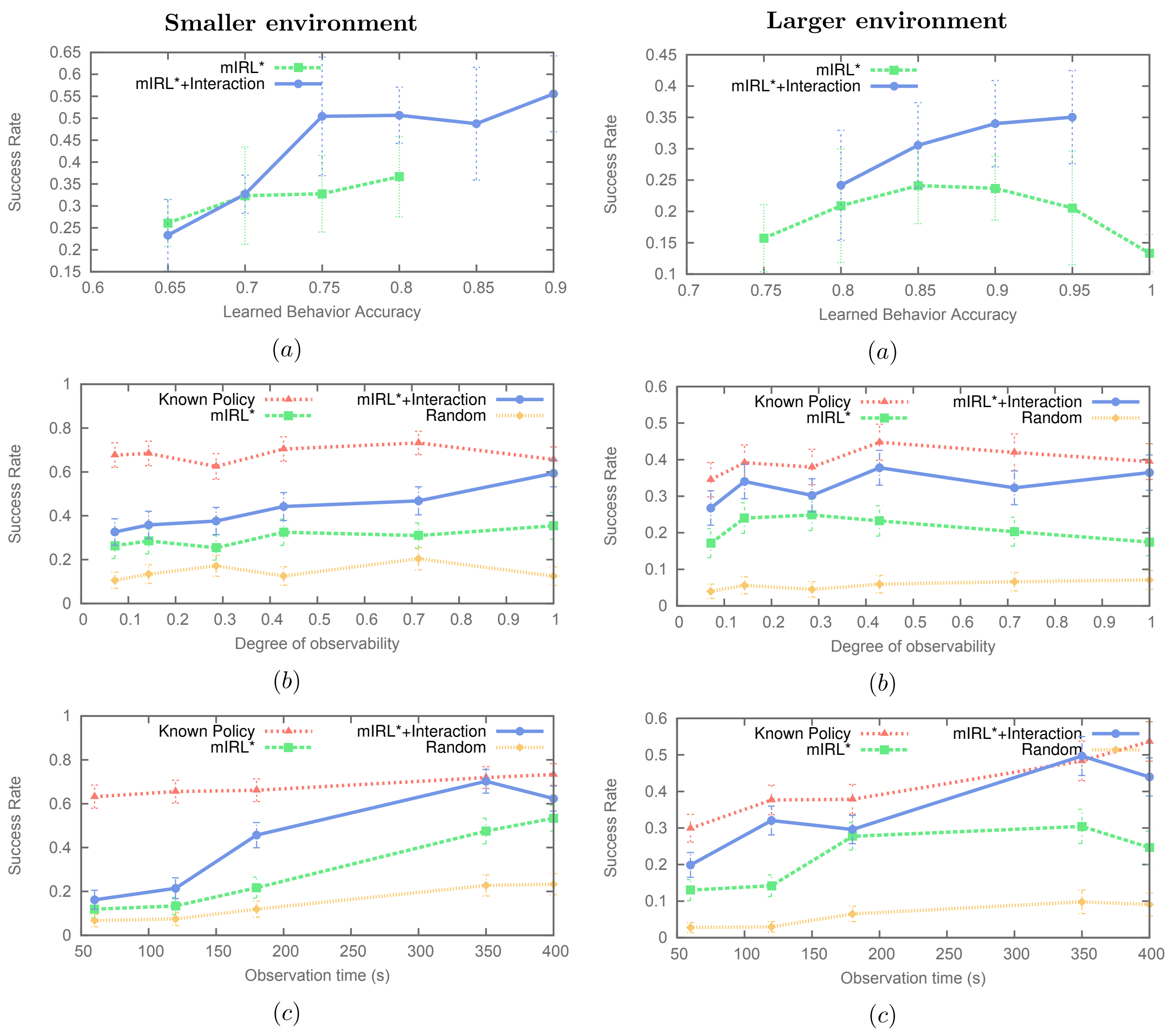


Figure 7.5: (a) The effect of the learned joint behavior accuracy on the success rate in both simulated environments. (b, c) A comparison of the success rates achieved by our approach and the baselines as a function of the amount of visibility and the time spent observing. Vertical bars indicate (a) standard deviation (standard error 0.048 (0.75, Smaller mIRL*+Int) to 0.012 (1.0, Larger mIRL*)) and (b, c) 95% confidence intervals.

7.1.5 Evolution of ω

As discussed in section section 6.1.2 we expect the Nash equilibrium weighting parameters, ω, to be relatively flat (high entropy) at the beginning of the optimization procedure since the reward weights considered will be essentially random. Then, as the numeric optimization proceeds and

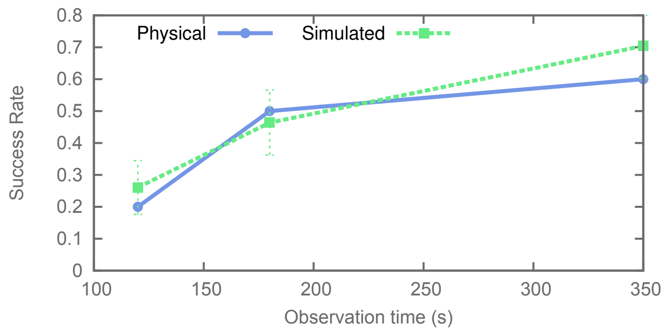


Figure 7.6: Success rates based on observation times in the physical runs. We compare these with those obtained from the simulations for the same degree of observability. The vertical bars are 95% confidence intervals.

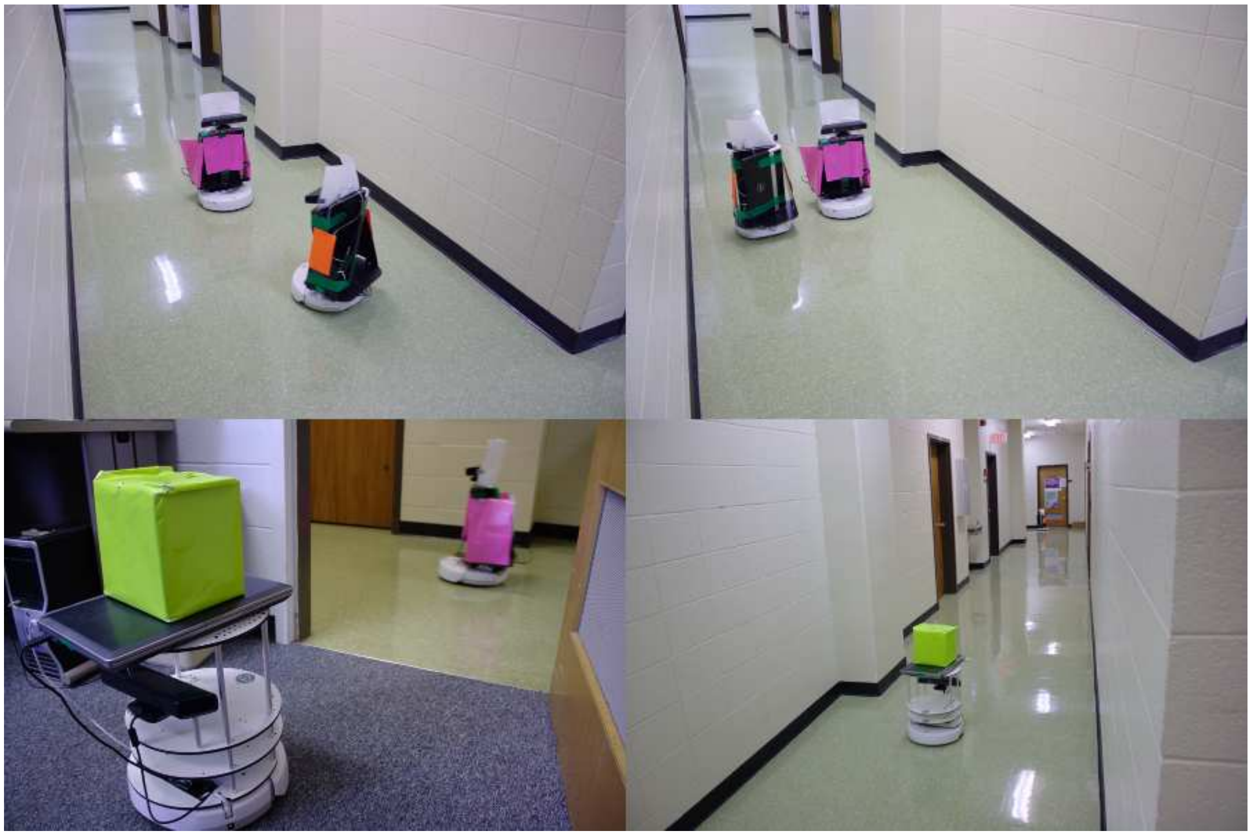


Figure 7.7: (Top) I and J in the process of interacting, J stops movement while the I moves past at a reduced speed. (Bottom) L observing a patroller from its vantage point (left) and performing a successful penetration (right)

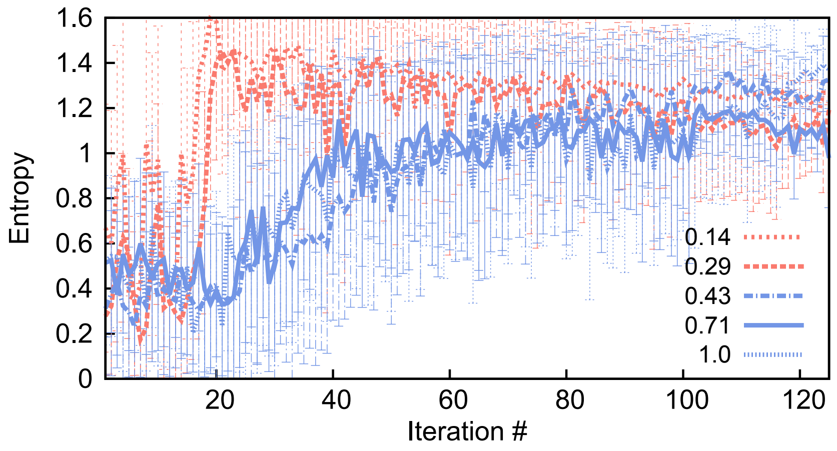


Figure 7.8: Average entropy of Nash Equilibrium weights ω during solving iterations for various levels of observability. Error bars are one standard deviation, standard error is between 0.115 (early iterations, lowest visibility) to 0.0049 (final iterations, lowest visibility).

the rewards begin to approach optimum we expect the weights to diverge towards the most likely NE's that correctly resolve the interaction. We recorded the ω distribution during the numeric optimization of mIRL*+Int for various levels of occlusion. This algorithm was limited to 125 iterations of Nelder-Mead simplex, and at the end the reward weights as well as the most likely NE used by the agents is returned. We sampled 30 times for each level of occlusion and averaged the entropy at each iteration together, standard error is between 0.115 (early iterations, lowest visibility) to 0.0049 (final iterations, lowest visibility).

Interestingly, as can be seen by figure 7.8, the ω distribution does not behave as expected and instead begins at low entropy and gradually rises to a moderate level. We observe that near the beginning of the solving process the ω distribution is erratic and may be near deterministic with one equilibrium containing nearly all the probability mass. This does not persist for long, however, and the next iteration a completely different equilibrium might contain the bulk of the probability mass, or a bad reward function might be chosen that results in ω reaching maximum entropy (1.6 in our experiment, meaning all equilibria equally likely). This is illustrated in figure 7.8 as the standard deviation error bars; notice their large size in the early iterations and gradually reducing towards the end. The vast majority of runs end with two or three equilibria having the bulk of the probability mass, with the others essentially zero, regardless of the level of observability.

Notice, however, the difference in behavior between the low observability (0.14 and 0.29) runs and the others. With few states available to compare the effects each equilibria has on the state visitation frequency in these particular runs ω oscillates more extremely and we see a jump to near maximum entropy around iteration 20, whereas runs with more visibility tend to smoothly increase entropy as the reward function converges.

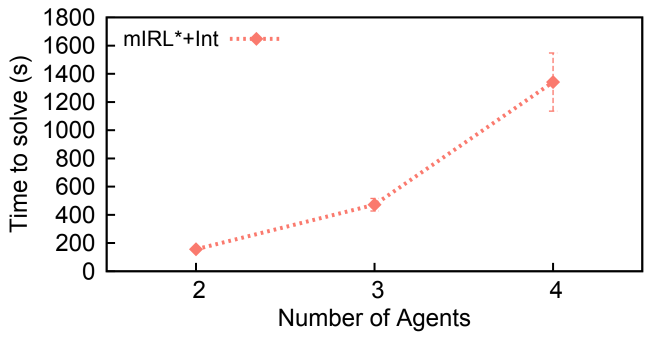


Figure 7.9: Average time to solve the patrolling domain (small map) as the number of agents is increased. Error bars are one standard deviation, standard error is between 65.16 seconds (4 agents) to 3.8 seconds (2 agents).

7.1.6 Scaling with multiple agents

An important question pertains to the scalability of mIRL*+Int as the number of robots is increased. As more agents are added to the problem, the number of reward function weights which must be solved for grows linearly (assuming similar MDPs for all agents). However, the number of Nash equilibria that resolve the interaction between these agents grows exponentially. At each solving iteration and for each equilibrium, we are required to sample repeatedly to estimate $\mu^{e_i}_{\pi_I}$. Therefore for large numbers of agents we expect the solution time to grow exponentially. For fewer agents the linear time needed by Nelder-Mead simplex technique is instead expected to dominate. We can see this happening in figure 7.9 as we add more simulated agents to our patrolling domain the solving time increases from 156 seconds to around 1,350. Comparison with greater than 4 agents is difficult in our scenario as the interactions begin to dominate the agent behavior, violating the sparse interaction assumption critical to mIRL*+Int. Each data point shown is the average of 10 runs, leading to standard error between 65.16 seconds (4 agents) to 3.8 seconds (2 agents).

7.1.7 Discussion

Our experiments demonstrate an improvement in both the learning accuracy and performance of L on its penetration task when the interaction between robots is modeled using our method. Accounting for the interaction removes a source of persistent noise from observations and enables more accurate predictions. We show our method for determining which Nash equilibrium the robots are likely using to resolve their interaction is effective, which is critical in scenarios involving occlusion as the learner may not observe the interaction.

Interestingly, though we expected the Nash equilibrium weights to be approximately equal at the start of numeric optimization and gradually converge on a single best solution we instead observe extremely dynamic changes in weights in response to changes in the reward function. These changes gradually reduce until convergence leaving two or three probable NE's.

7.2 Evaluation of Multi-Agent *Hidden Data EM*

We evaluate multi-agent *Hidden Data EM*, developed in section 6.3, in a similar multi-robot patrolling scenario to that described in section 7.1 and so to save space we will only describe here the differences from those experiments.

As before, there are two robots, I and J, patrolling a space that an attacker L wants to move through without being detected. The patrollers move in a shared space and interact when approaching each other to avoid collisions, their interaction mode is known to the attacker ahead of time. L is required to 1) observe the patrollers without moving to gather data for IRL 2) use IRL to recover the patrollers' reward function and subsequently the policy they are using 3) using the found policies predict the future path of the patrollers 4) plan a route through the space that avoids the predicted path of the patrollers while still arriving at the goal state.

As in section 7.1 we use two measures to quantify the ability of IRL to learn the behavior of the patrollers. The first is **Learned Behavior Accuracy** which is the proportion of states in which the optimum policy for the learned reward function gives the correct action. This gives a measure of the prediction accuracy to be expected from a given technique. The second measure is the **Success Rate** of L as a proportion of penetration attempts that result in L reaching its goal without being detected. This measures all aspects of the experiment but suffers from the possibility that L could fail to learn the patrollers' policies accurately but still reach its goal undetected by chance.

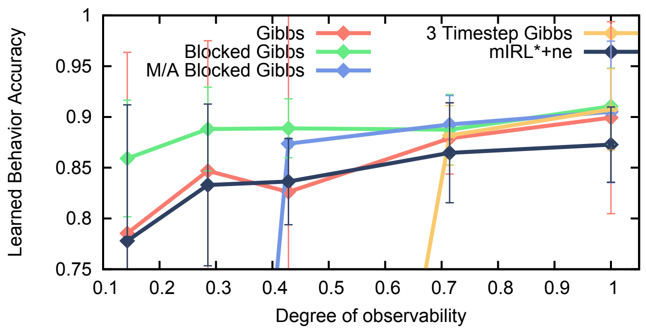
Additionally, we report the number of **Time Outs** for each solving method as a measure of L taking too long to perform its task. We time out each run of the experiment after 25 minutes and if L has failed to reach the goal before then the run is counted as a failure. Time outs can occur if 1) the IRL solving algorithm takes too much time or 2) the policies learned using IRL produce no useful predictions, causing L to believe there is no safe path to the goal.

7.2.1 Penetrating a simple multi-robot patrol

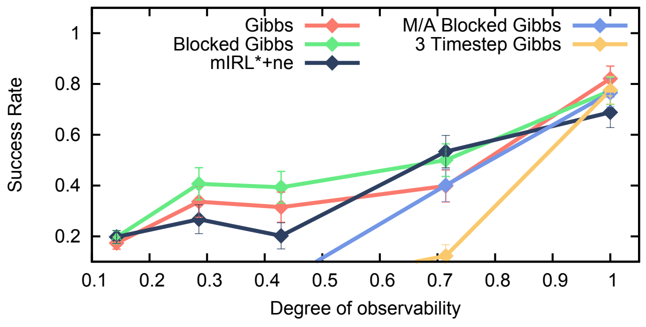
Our multi-robot experiment in this section is similar to that described in section 7.1 however the state space and feature count are expanded in size to emphasize the differences in *Hidden Data EM* and mIRL*+ne solution methods. Particularly, we expect *Hidden Data EM* to be less affected by the feature count due use of the Langrangian dual's gradient.

Each patroller is modeled as acting according to the output of its own MDP with interactions resolved by an interaction game Nash equilibrium. The states of this MDP are the cell decomposition of the patrolled area (x, y) and an additional discretized orientation ψ. Each patroller may take one of 4 actions: Move Forward, Stop, Turn Left, and Turn Right. The transition function models the probability of any action succeeding at 92.5% with the remaining probability mass distributed to the other actions.

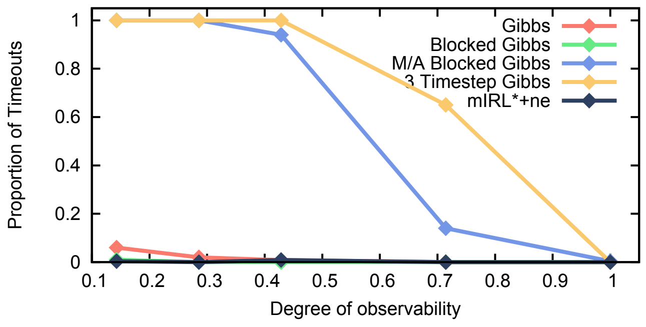
L is modeled as in section 7.1, with a 4 dimensional MDP it uses to predict the future locations of the patrollers once their policies are learned. We use the most likely policy only for our predictions (found using standard value iteration using the learned reward functions). We use a map of an indoor office environment in our simulated patroller experiments. As can be seen in figure 7.11 L is located in a corner location from which it can gather observations while the patrollers move in a cyclic fashion between their pre-determined way-points. Each robot in our experiment is a simulated Turtlebot equipped with a video camera and laser scanner. Interactions between the robots occur as they attempt to pass each other, resulting in one robot stopping while the other



(a)



(b)



(c)

Figure 7.10: (a) The effect of amount of observability on the average Learned Behavior Accuracy (error bars are one standard deviation, standard error ranges from 0.010 (*Gibbs*, lowest observability) to 0.0024 (*Gibbs*, highest observability)) (b) Success rate achieved by each method, vertical bars 95% confidence intervals (c) Proportion of runs which timed out, we stop a run after 25 minutes.

moves slowly around it, and are time extended to 3 timesteps to model the slower movement of the robots during interaction. Timesteps are two seconds long.

The reward functions of both robots are a linear combination of two types of features:

1. *Has moved*, which returns 1 if the action causes the patroller to leave its current location, otherwise 0; and

2. *Turn around at state, s*, which returns 1 if the robot turns at the location given by s, otherwise 0.

3. *Catch All*, which returns 1 for all other state-action pairs not matched by the above two features.

The true reward function penalizes turning around in the middle of the hallway and rewards turning at the goal states. As a result, the optimum policies of the patrollers generates trajectories that move through the hallways, turn around at the ends, and move back. This scenario is illustrated in figure 7.11 (a) which shows a trace of the patrollers during their patrol route up until they begin an interaction. A diagram of the MDP used is shown in figure 7.11 (b), the shaded squares are the states in which the patrollers turn around in. L must move from its position in the top left corner to reach the red X without being seen by either I or J.

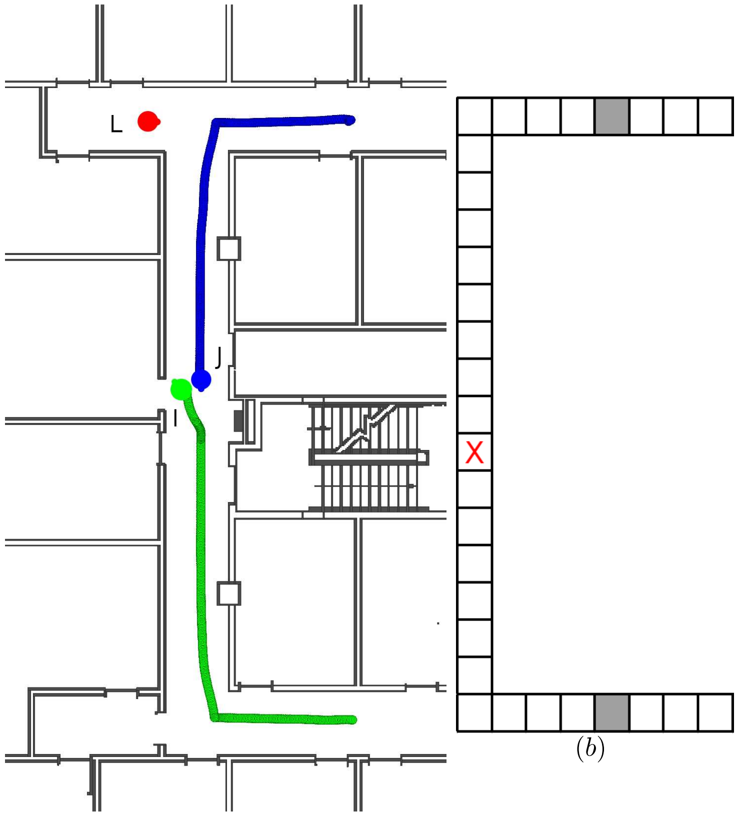


Figure 7.11: (a) A diagram showing the map for our simulated patrolling experiment (b) corresponding MDP state space for each patroller. Shaded squares are the turn around states and the red X is L's goal state

We test the methods given in section 6.3.1 in this section, we do not test the exact methods as their runtime is prohibitively expensive. As the size of $\mathbb{Z}$ is very large due to the large number of timesteps in a given trajectory, the multi-agent state space increase, and the large number of states per agent the exact methods of calculating the E step of *Hidden Data EM* are not employed here. Instead we test *Gibbs Sampling*, *Blocked Gibbs Sampling*, *Multi-Agent Gibbs Sampling*, and *3*

Timestep Gibbs Sampling as previously described. As time is a limiting factor we do not perform restarts of the EM algorithm, limit the maximum EM iterations to 7 (3 in *3 Timestep Gibbs Sampling* for performance reasons), and limit the E step feature expectation mean-of-means calculation to a maximum of 50 repeats.

Additionally, we compare to mIRL*+ne as described in section 6.1.1. This approach, while penalized due to unavailable of the gradient, has an advantage in that it does not require computing the expectation over trajectories and so is expected to take constant time with respect to $|S_{occ}|$.

7.2.2 Experiments and Results

We first examine the **Learned Behavior Accuracy** of all provided methods as a function of the percent of states that are visible to L. Figure 7.10 *(a)* shows that of all the methods tested, *Blocked Gibbs* performed the best. Observe the large standard deviation of mIRL*+ne and *Gibbs* as compared to *Blocked Gibbs*, which is much more consistent in its output. Note when all states are observable, all *Hidden Data EM*-based methods are equivalent; without missing data no E step is needed. Each data point is the result of 230 runs, resulting in a standard error that ranges from 0.010 (*Gibbs*, lowest observability) to 0.0024 (*Gibbs*, highest observability).

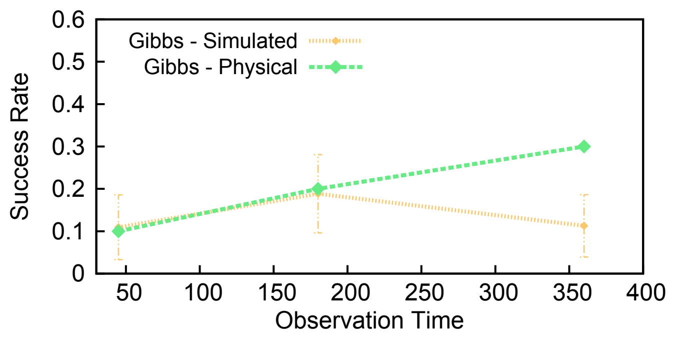
We next report the **Success Rate** achieved by all methods as a observability changes. In figure 7.10 *(b)* we see *Blocked Gibbs* is again the best overall method, matching or outperforming mIRL*+ne and all other *Hidden Data EM*-based methods. As observability decreases these methods are required to sample from a greatly increased space of possible trajectories, increasing the time to find a solution. This leaves less time to find an opportunity to successfully attack, and results in an increased number of **Time Outs**. As can be seen in figure 7.10 *(c)*, mIRL*+ne experiences the fewest timeouts of all methods followed closely by *Blocked Gibbs*. Notice the high proportion of timeouts experienced by *3 Timestep Gibbs* and *M/A Blocked Gibbs*, this explains its observed poor performance as these sampling algorithms require too much time to sample $\mathbb{Z}$ under large amounts of occlusion and cause most runs to time out without an attempt on the goal.

These experiments indicate that *Blocked Gibbs* is the best method of those tested, achieving a higher expected accuracy and a success rate that outperforms all other methods in most cases and avoiding an excessive run time in high occlusion scenarios.

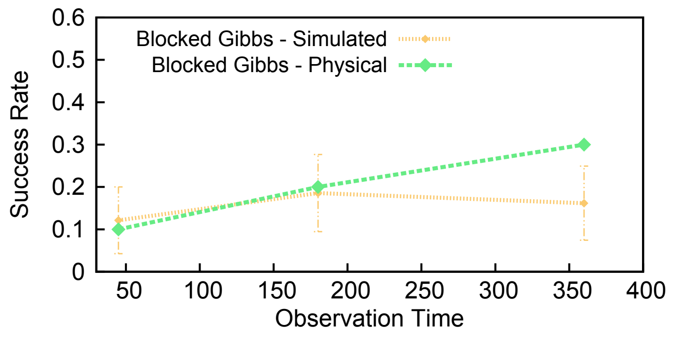
To verify the applicability of *Hidden Data EM* to real-world robots we performed a set of experiments with physical Turtlebots in conditions that matched the lowest observability in our simulations. As altering the observability is not possible in the physical setup we instead varied the observation time and performed 10 runs per data point. As can be seen in figure 7.12 all methods closely matched simulation results for 45 and 180 second observation times. However, interestingly both *Hidden Data EM* methods outperformed their simulations at 360 seconds. While the amount of data is too small to draw conclusions this may hint at an unexpected robustness of *Hidden Data EM* to noisy trajectories, more research is needed.

7.2.3 Comparison of M-Step Methods

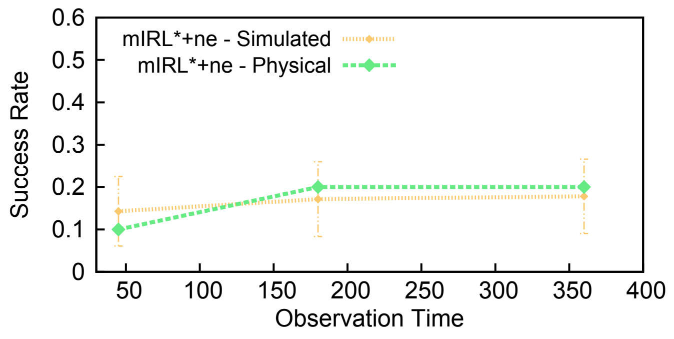
We described the stochastic gradient descent approach (*SGD*) we use in the M step of *Hidden Data EM* in section 6.3.2 as well as two approximations: using the previous empirical state distribution as μ_{t-1} which we label *SGD-Approx* and simply assigning μ_t to $\sum_{a'} \tilde{Pr}(s,a';t)$ which we label *SGD-Empirical.* Here we compare the performance of these methods on randomly generated MDPs. We generate MDPs with six states, four actions, and random transition and reward functions. We



(*a*)



(*b*)



(*c*)

Figure 7.12: Success rates based on observation times in the physical runs for (*a*) *Gibbs Sampling* (*b*) *Blocked Gibbs Sampling* and (*c*) mIRL*+ne. We compare these with those obtained from the simulations for the same degree of observability. The vertical bars are 95% confidence intervals.

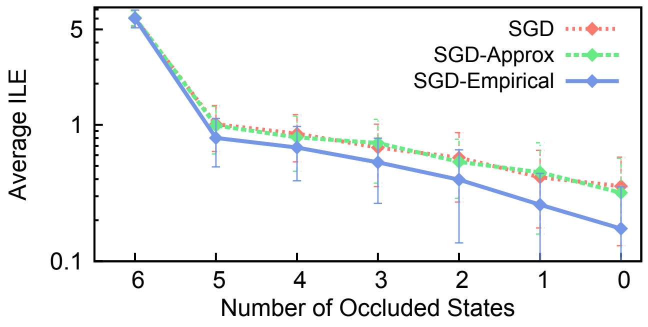


Figure 7.13: Average Inverse Learning Error (ILE) of the M-step methods as the number of occluded states decreases. Note log scale. Random MDPs with 6 states and 4 actions, trajectories of length 10 and 10 trajectories sampled. Error bars are one standard deviation, standard error is between 0.08999 (6 occluded states, *SGD-Empirical*) to 0.017 (0 occluded states, *SGD-Empirical*)

optimally solve each one using value iteration and repeatedly sample 10 trajectories of length 10 timesteps using the optimal policy. The E step of *Hidden Data EM* is solved using Gibbs sampling.

We show the ILE of each method as a function of the occlusion level in figure 7.13. Notice that *SGD* and *SGD-Approx* both achieve approximately the same ILE and both are outperformed by *SGD-Empirical*. We may explain this by noting that with this low number of samples the empirical state distributions may be ill-estimated penalizing the *SGD* approach. In this experiment the initial state distribution was set to the empirical state distribution at the first timestep; this ensures that an ill-matching initial state distribution could not be the cause of the performance difference.

As *SGD-Empirical* cannot correctly compute the gradient if state-only features are used (the state visitation frequency must be used to do so) we must utilize reward features that map to one state-action pair, therefore these results are not directly comparable to figure 4.1 despite the similar random MDP settings.Each data point is the average of averages from the output of 1000 trials grouped into blocks of size 10. Standard error is from 0.08999 (6 occluded states, *SGD-Empirical*) to 0.017 (0 occluded states, *SGD-Empirical*).

The motivation for developing the approximations used in the M-step is the increase in computation time of *SGD* as the length of the observed expert's trajectory increases due to the need to recursively calculate the state distribution for t timesteps whereas *SGD-Approx* and *SGD-Empirical* take constant time regardless of the length of the trajectory. This is relevant to our patrol penetration experiments due to their sensitivity to IRL computation time and a maximum trajectory length of 180 time steps. As can be seen in figure 7.14 the time taken for all three methods is linear in the length of the trajectories but *SGD-Approx* and *SGD-Empirical* scale much better than *SGD*. Each data point is the average of 20 trials, standard error is between 5.851 (150 trajectories, *SGD*) to 0.015 (10 trajectories, *SGD-Empirical*).

7.2.4 Discussion

Our experiments demonstrate that multi-agent *Hidden Data EM* offers improved accuracy and in many cases improved performance in our multi-robot patrolling domain. Blocked Gibbs sampling,

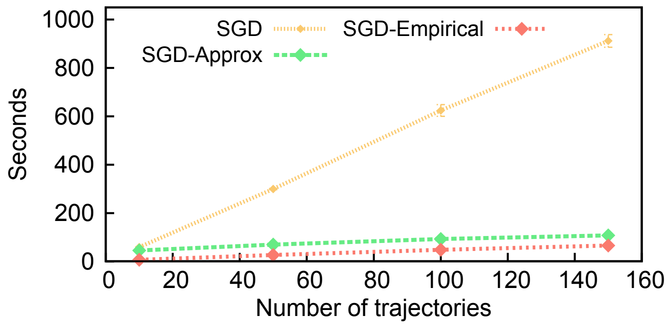


Figure 7.14: Average time to solve randomly generated MDPs (25 states, 4 actions) for each M-step method as the length of sampled trajectories increases. 10 trajectories sampled. Error bars are one standard deviation, standard error is between 5.851 (150 timesteps, *SGD*) to 0.015 (10 timesteps, *SGD-Empirical*)

with each block the size of one agent's state-action pair at a single timestep, is shown to perform the best of the options tested. When the learner is time-constrained the method of calculating the E step of *Hidden Data EM* is shown to be of critical importance.

We analyzed our stochastic gradient descent approximations and found that *SGD-Empirical* outperformed the others in both accuracy and speed. This is attributed to receiving too few trajectories to adequately estimate the state distribution at each timestep, leading to inaccurate results. We note though that when *SGD-Empirical* is used no features may be state-only; weights for these features may only be learned by considering the state distributions.

7.3 Evaluation of $\textbf{mIRL}^*_{/T}\textbf{+Int}$

We evaluate our method of estimating the transition function of an observed agent, developed in section 5, in a similar multi-robot patrolling scenario to that described in section 7.1 and so to save space we will only describe here the differences from those experiments.

As before, there are two robots, I and J, patrolling a space that an attacker L wants to move through without being detected. We now allow for the possibility that one of the patroller's wheels is damaged, unknown to L. We show a simulated trace of the damaged patroller's movement in figure 7.15 This damage will cause one patroller to move much more slowly than the other which manifests as lower transition probabilities to its intended next state. This potentially gives L more opportunities to reach its goal if the patrollers can be correctly predicted. A robust test of our new technique is if L can identify the correct patroller and specific feature that is causing it lower transition probabilities. The error model used in these experiments is to allocate all probability mass to the current state in event of a failure which accurately models a robot that is moving too slowly to make it to the next state within a given timestep.

7.3.1 Transition Features and Algorithms

L utilizes the following independent binary feature random variables as part of $\mathcal{T}_I$ and $\mathcal{T}_J$:

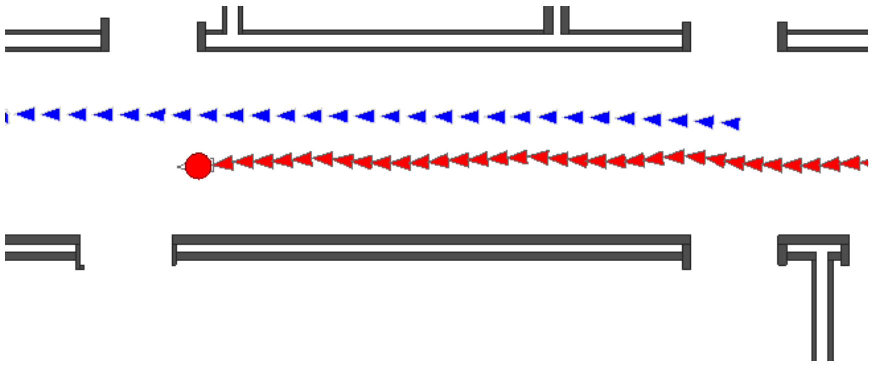


Figure 7.15: Simulated trace showing the damaged patrolling robot (red) and its slower, oscillating movement. Undamaged patroller trace (blue) shown for reference.

1. *Rotate left wheel at specified speed*, used at all states and for all actions except stop;
2. *Rotate right wheel at specified speed*, used at all states and for all actions except stop;
3. *Navigation ability* that models the robot's localization and plan following capabilities in the absence of motion errors, used at all states and for all actions except stop.
4. *Floor slip*, used for all states and actions

We use the technique developed in this chapter to augment the algorithm described in chapter 6 by first learning the transition function of the observed robots before learning the Rewards. This produces an algorithm labeled $\mathsf{mIRL}^*_{/T}$+Int and we compare this directly to mIRL^*+Int with various manual settings for ψ. We take a similar approach with the upper bound algorithm, *Known* $R_{/T}$ reveals the true reward function of all patrollers to L but utilizes our technique to calculate the transition function, and in *Known R* we manually fix ψ to various settings. Finally, we compare our techniques to a popular existing approach for learning the transition function by modeling as a dynamic Bayesian network and solving using Expectation-Maximization (DBN_{EM}). In this approach a patrolling robot's transitions are modeled as a two step state-action-state DBN whose parameters are learned from the observed portions of the trajectory using EM.

7.3.2 Most likely transition error model

One remaining concern is which transition error model to use in our domain? Following the procedure detailed in 5.4 we develop a number of possible error models and choose the maximum likelihood one as calculated using the error transitions from the given trajectory. The candidate models are:

- *Stop* - all remaining probability mass to the current state the agent is in.
- *Other Actions* - remaining probability mass divided equally among the intended states for the other actions (not including the one the agent performed)

- *Nearby States* - remaining probability mass divided equally among all states whose center point is ≤ 1 meter from the starting state's center point
- *Uniform* - remaining probability mass divided equally among all states (except the intended)

In our experiments the *Stop* model is found to be the maximum likelihood 100% of the time. We attribute this to the most common error mode of the robots being moving too slow to make it to the next state, thereby appearing to "stop" for a timestep. For easy comparison all experiments in this section use the *Stop* error model to distribute the remaining probability mass.

7.3.3 Simulation Results

We first examine the Learned Behavior accuracy of the algorithms that utilize our technique described in this chapter. As can be seen in figure 7.16 increased observation amounts result in higher accuracy for both algorithms. Each data point is calculated from at least 100 simulated runs by placing them in blocks according to learned behavior accuracy of size 15 at intervals of 0.05, then averaging the success rate over the blocks. Standard error ranges from 0.09 (0.90, Known $\mathsf{R}_{/T}$) to 0.0099 (1.0, Known $\mathsf{R}_{/T}$). Furthermore, the success rate of L correlates strongly with the learned behavior accuracy, again demonstrating the importance of accurate learning.

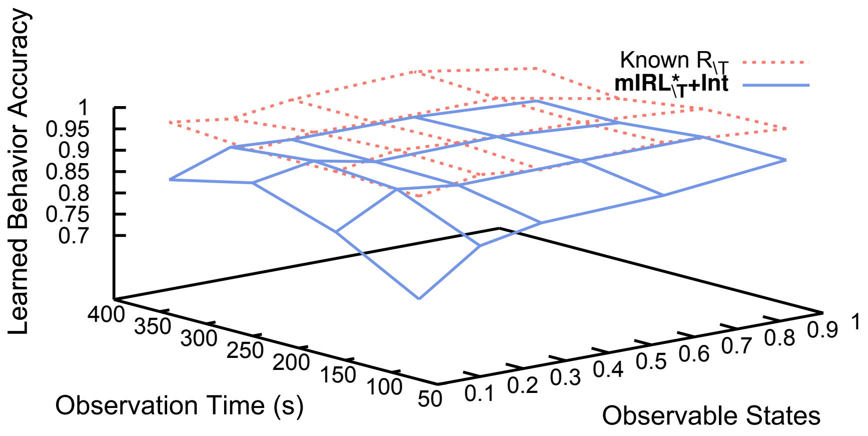
Next, we evaluate the success rate when J's left wheel is damaged. As can be seen in figure 7.17 (a), the two algorithms which used fixed values for ψ performed worse than their equivalent that used our method of learning it. Notice too the performance of the DBN_{EM} method is below random, due to this method's inability to learn the transition function of the other agents under occlusion. Indeed, runs using this method timed out (no valid attack plan found) 100% of the time when occlusion was present and 90% without it. By comparison, the time out rate of all other methods is under 4% regardless of occlusion. This strongly indicates that utilizing transition features that are shared among multiple state-action pairs facilities robust learning.

In figure 7.17 (b) we show the learned transition feature probabilities for both patrollers averaged over all state-action pairs. Notice that $mIRL^*_{/T}+Interaction$ correctly learns the left wheel of J is damaged by assigning the feature a lower probability of success, much lower than for I. Additionally the left wheel's success probability is lower than the right wheel by a statistically significant amount (student's 2-tailed t-test, $p << 0.001$).

Finally, in figure 7.17 (c) we see the learned transition feature probabilities for both patrollers when there are no damaged wheels. Comparing the probabilities found to those in 7.17 (b) we see that both wheels are assigned approximately the same probability of success for both patrollers and closely matches our upper bound control.

7.3.4 Physical Robot Results

For our physical robot experiments we compared $\mathsf{mIRL}^*_{/T}$+Int with mIRL^*+Int where the transition success rate fixed at 0.9 as well as a *Random* baseline in runs where patroller J's wheel is damaged and not. We report the success rate in table 7.1 with 10 runs per data point. Notice that the success rate improves when L is allowed to learn the transition function. Also as can be seen in this table runs with a damaged wheel improve the chances of L's success across all methods used. We illustrate our physical experiments in figure 7.18 by showing a successful attack by L (top) as well as the view from L's vantage point as it watches a patroller (bottom).



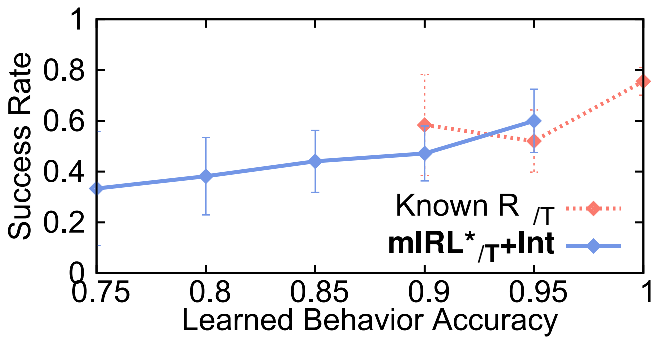
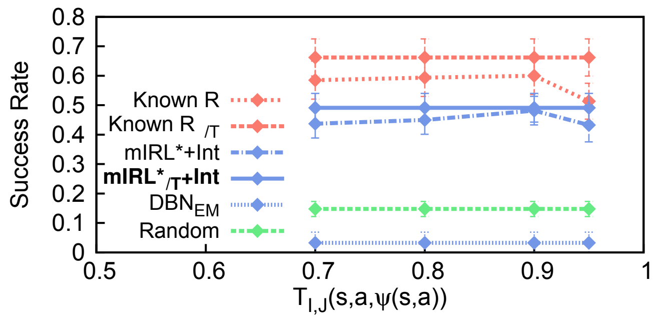


Figure 7.16: (**top**) Learned behavior accuracy of $\mathsf{mIRL}^*_{/T}$+Int and Known $R_{/T}$ for different occlusion rates and observing times. (**bottom**) Improving accuracy of learned behavior correlates almost linearly with success rate. Vertical bars denote one standard deviation, standard error ranges from 0.09 (0.90, Known $R_{/T}$) to 0.0099 (1.0, Known $R_{/T}$)

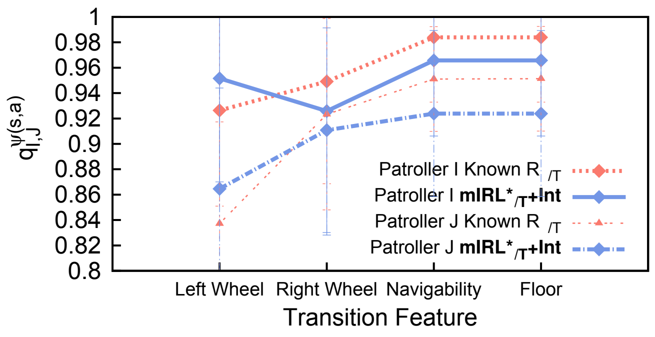
7.3.5 Bernstein Polynomial Approximation

We examine the approximation presented in section 5.5 by comparing its performance to our exact formula. Using trajectories generated from our patrolling scenario, we calculated the $Pr(\tau)$ using the exact and convex approximation techniques. The exact method utilized Sequential Least Squares Programming implemented in the SciPy software package and the approximate method was solved with a primal-dual interior point method from the CVXOPT python package. We then calculated the average difference in the probabilities found by the two methods: over 1,110 trajectories the median average difference was 0.047 with quartiles 0.026 and 0.075.

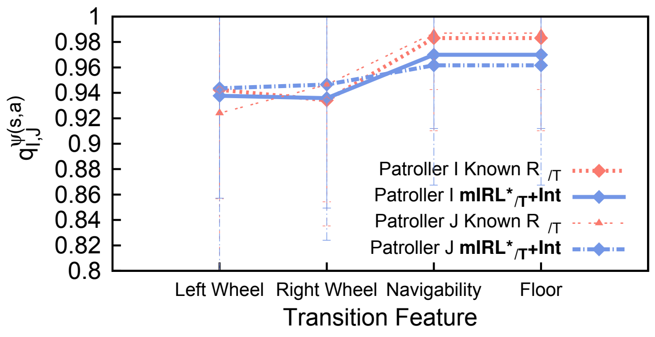
The primary advantage of using the convex approximation is scalability as the number of constraints increases. Our scenario uses only 4 features and a mapping resulting in 4 $q_I^{(s,a)}$ constraints (one per action). To show scalability, we increased the number of features and mapped them such that there is one $q_I^{(s,a)}$ constraint per state (9 features, 4 mapped to each state) and again such that each state-action pair had a unique constraint (12 features, 4 mapped to each state-action pair). This resulted in 124 and 496 constraints, respectively. In figure 7.19 we report the average



(*a*)



(*b*)



(*c*)

Figure 7.17: (*a*) Comparative success rates of L for various methods that either learn T_I and T_J or fix it arbitrarily. True transition probabilities of the patrollers are not known. (*b*) Transition feature probabilities that correctly identify that the left wheel of J is partially damaged as indicated by its comparatively low success probability. (*c*) Transition feature probabilities when both patrollers are operating properly.

	Success rate	
Method	**J's left wheel damaged**	**No damaged wheels**
mIRL$^*_{/T}$+Int	*0.60*	*0.50*
mIRL*	0.50	0.40
Random	0.40	0.20

Table 7.1: L's success rates using various methods over 10 *physical runs* each. L suffers from very high occlusion of patrollers.

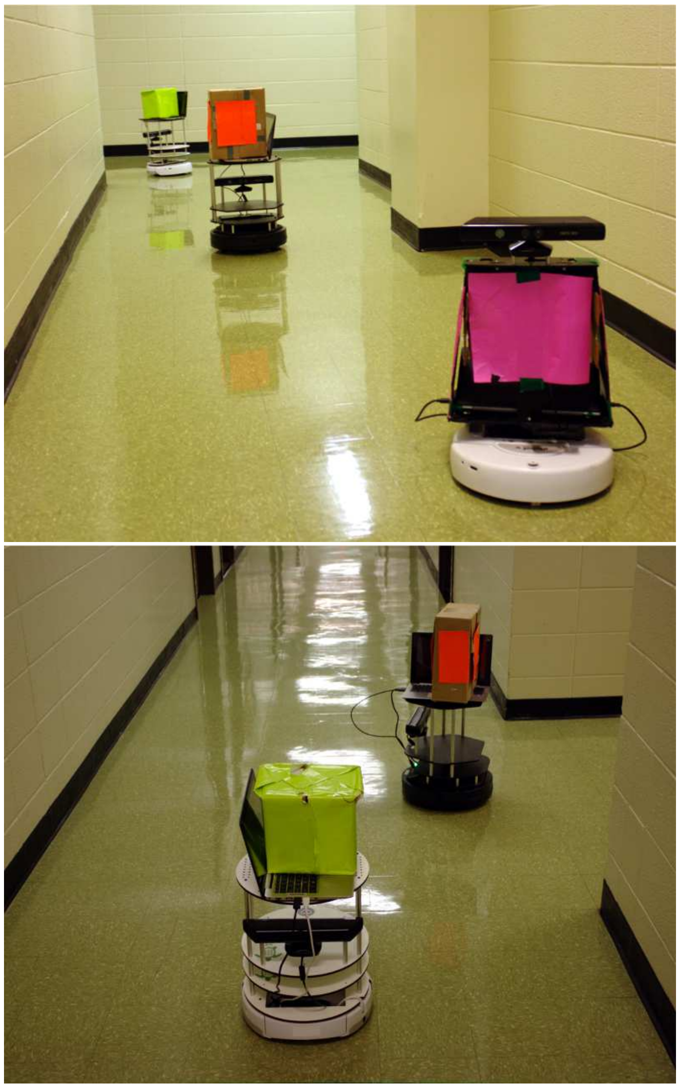


Figure 7.18: (top) Patrollers I and J moving through the hallway while L begins its attack behind them (bottom) L observing I from its vantage point

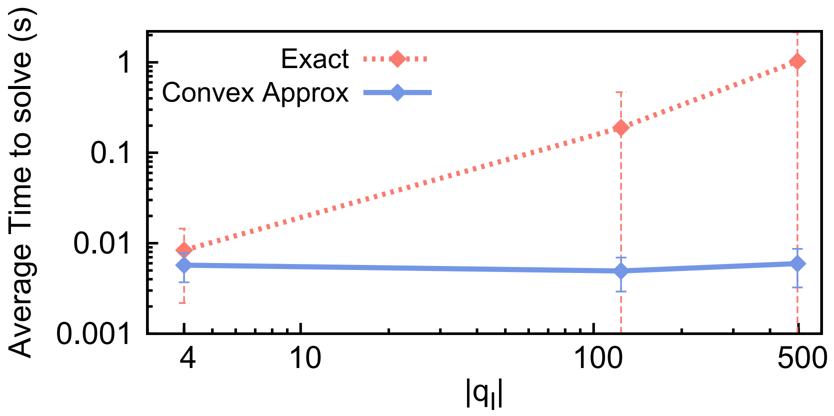


Figure 7.19: Time to solve (in seconds) as a function of the number of q_I constraints. Note log scale on both axes. Error bars are one standard deviation, standard error ranged from 0.000006 (4 constraints, approximate method) to 0.033 (496 constraints, exact method)

time our exact and convex approximation algorithms took to solve on these two larger problems. The approximation method took about 0.005 seconds to solve all problems on average while the exact method ranged from 0.0085 seconds (std. dev. 0.0062) with 4 constraints to 1.02 seconds (std. dev. 1.114) with 496 constraints. Standard error ranged from 0.000006 (4 constraints, approximate method) to 0.033 (496 constraints, exact method).

7.3.6 Discussion

We empirically validate our model in our robotic patrolling domain by showing the ability of the attacker to correctly determine which wheel is damaged of a patrolling robot. We show our model provides improved success rate over a manually specified transition function, in particular because the transition functions for the damaged and undamaged robots are different. Finally, the convex approximation developed in section 5.5 is shown to offer improved scalability over the exact method while producing similar results. Note that our approximation uses a Bernstein polynomial of degree 4, higher degrees uniformly converge towards the original function and are therefore expected to provide more accurate results with minimal effect on scalability.

7.4 Summary

Our experiments demonstrate the applicability of our methods to domains involving real-world robots. Even under extreme occlusion useful policies could still be learned that explains the behavior of the patrollers accurately enough for L to reach the goal. By modeling the interaction between robots as a Nash equilibrium that is unknown to the learner we remove a source of persistent noise from observations of the robots improving L's ability to perform IRL on them and predict their future positions.

By taking into account an expectation of the patroller's behavior in the occluded areas and compensating for the increased computation time this causes we improve the accuracy of the learned policies and in many cases the performance of L on its task.

In real-world scenarios the learner will likely not have the transition function of the expert ahead of time and must estimate $T()$ from the expert's observed behavior. When the requirements of our model are satisfied (intended next state known, transition feature mapping known) we show improvement in the learner's performance by using our model to learn $T()$ over fixing the transition function to various values ahead of time.

Chapter 8

Conclusions

In this chapter we discuss our contributions and potential future avenues for research.

8.1 Discussion

When an autonomous robot is attempting to perform Inverse Reinforcement Learning on the agents it encounters in its environment a number of challenges arise. Existing IRL methods fail to adequately address these difficulties, in particular: occlusion creating persistent missing data in the observations the robot receives, the presence of multiple interacting agents in the environment whose interactions are sources of persistent noise, and a lack of knowledge of the observed agent's transition function. The work presented in this dissertation has developed principled extensions of an IRL method, MaxEnt IRL, to address these challenges and demonstrated their effectiveness in a number of simulated and physical robotic domains.

We first show the effects of simply ignoring missing data, which is justified in some contexts, before presenting a method to replace missing data with its expectation in IRL as an application of the latent maximum entropy principle and illustrate the accuracy improvement it provides. Simple methods are shown to be effective in reducing the solving time of *Hidden Data EM* without compromising accuracy.

We present a model of spurious agent interaction based upon game theory and show that its use improves performance in a challenging robotic domain. This new model is partially applied to *Hidden Data EM* and this algorithm is adapted to significantly sized robotic problems. We show this algorithm offers an accuracy improvement over mIRL*+ne which may well be worth its increased computational costs in many contexts. In scenarios where the interaction behavior is unknown, possibly due to occlusion, we presented a method to consider all possible equilibria during the solving iterations.

Finally, we develop a new method for estimating the transition function of an observed agent based upon learning underlying transition feature probabilities. We show this method's utility in a scenario in which one of the robots it must learn from has a damaged wheel, necessitating a learning algorithm. Despite heavy occlusion and a lack of exploration of the state-action space by the observed robot our method is capable of learning useful transition functions as well as successfully identifying the specific damaged wheel.

Our work represents important first steps towards making IRL a machine learning technique usable by autonomous robots in real world scenarios in which other agents are encountered. As robots are expected to be deployed among people in the near future it is our hope that this work contributes positively to the development of these machines.

8.2 Future Work

This work is by no means an exhaustive treatment of all possible issues a robot could encounter performing IRL in the real world. For example, Inverse Reinforcement Learning assumes that the expert is rational which may not be justified when attempting to learn from humans and other animals. A rigorous treatment of this situation may model the expert as risk-sensitive or using a one-switch utility function.

Future applications include utilizing IRL to learn the policy of an agent, formulating a best response to it, and subsequently forming an ad-hoc team between the learner and expert. Inverse Reinforcement Learning generally assumes no interaction between the expert and learner during the observed demonstration preventing the learning of behavior that requires cooperation between the two. Relaxing this constraint would allow the learner to play an active role in the task being performed and influence the expert towards states the learner is interested in observing, resulting in more complete learning.

One possible future advancement of IRL*'s solution method is to treat all constraints as lower bounds on the possible expert feature expectations with an upper bound provided by the maximum possible feature count that could be received from the time steps in which the expert is occluded, then solve the resulting problem using any convex optimization with inequality constraints method [15].

An open question that remains from our work is how can multi-agent *Hidden Data EM* be extended to scenarios where the interaction behavior is unknown? We believe a good direction of inquiry is inspired by our work in section 6.1.2 where, assuming the interactions are sparse, the probability that each Nash equilibrium is the one the robots are using is proportional to the difference of feature expectations of the distribution over trajectories and feature expectations of the expert. We may imagine adding extra constraints onto equation 4.4:

$$\begin{aligned}
&\max_{\Delta} \left(- \sum_{X \in \mathbb{X}} Pr(X)\ log\ Pr(X) \right) \\
&\textbf{subject to} \quad \sum_{X \in \mathbb{X}} Pr(X) = 1 \\
&\sum_{X \in \mathbb{X}} Pr(X) \sum_{\langle s,a \rangle \in X} \phi_k(s,a) = \hat{\phi}_k^{Z|_Y} \qquad \forall k \\
&\sum_{X \in \mathbb{X}} Pr(X) \sum_{\langle s^I,a^I,s^J,a^J \rangle \in X} \Upsilon_l(s^I,a^I,s^J,a^J) = \hat{\Upsilon}_{ll}^{Z|_Y} \qquad \forall l
\end{aligned} \tag{8.1}$$

Where Υ_l is a feature that matches Nash equilibrium l when a^I and a^J indicate equilibrium l and S^I and S^J are interacting states.

Our occlusion methods require fully observable and fully hidden states. Depending upon the application it may not be feasible to satisfy this constraint. Instead a more advanced technique which allows for partial observability of states and actions is needed. This was first treated in [26] by defining Hidden Variable Inverse Optimal Control in which the learner receives an observation which stochastically depends upon the state of the expert agent at a given timestep. Generalizing this technique to allow the observation to also depend upon the expert's action requires a more advanced method and we believe the *Hidden Data EM* approach may be applicable here by changing $\mathbb{X}$ to be $(Traj \cup \omega)$, where Trajectory is one possible trajectory of the expert and ω is a sequence of observations of the expert made by the learner which only partially reveals the expert's state-action pair at each timestep.

In chapter 5 we partially relaxed the requirement that the learner know the complete transition function of the expert it is observing by requiring instead that the learner identify the expert's intended next state for every state-action pair and have full knowledge of the transition feature set and mapping. A limitation of our current approach is that each state-action pair is restricted to a single intended next state only. One avenue of future work is to relax this limitation allowing multiple intended next states. A possible approach may be to split the transition function into multiple transition functions each mapped to its own subset of features.

Another requirement is full knowledge of these intended next states. In some cases they may be determinable through observations of the expert, for instance a probabilistic model of the intended next state for each state-action pair could be developed whose parameters are learned from trajectory data. A set of transition feature probabilities could be learned from this model, and an expectation taken over the probabilities (weighted by the intended next state model) to arrive at the most likely transition probabilities.

Appendix A

Appendix

A.1 Robotic Experiment Details

Here we provide extra details related to our robotic experiments that may clarify some details for the reader or assist with repetition of this work.

A.1.1 Sorting under occlusion: details for section 4.4.1

Expert MDP

States:

The states combine every combination of the following variables, for a total of **384** states.

- *Location* - {On table, Center area, Bin 1, Bin 2}
- *Ball type* - {1, 2, 3, 4}
- *Holding ball* - {Yes, No}
- *Time for previous action* - { $<$ 0.3s, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, $>$ 1.3s}

Holding no ball in Bin 1 or Bin 2 are terminal states.

Actions:

- *Grab Firm* - Only applies in the On table states
- *Grab Soft* - Only applies in the On table states
- *Move to center position* - Only applies in the On table states
- *Move to Bin 1* - Only applies in the Center area states
- *Move to Bin 2* - Only applies in the Center area states
- *Release Firm* - Only applies in the Bin states

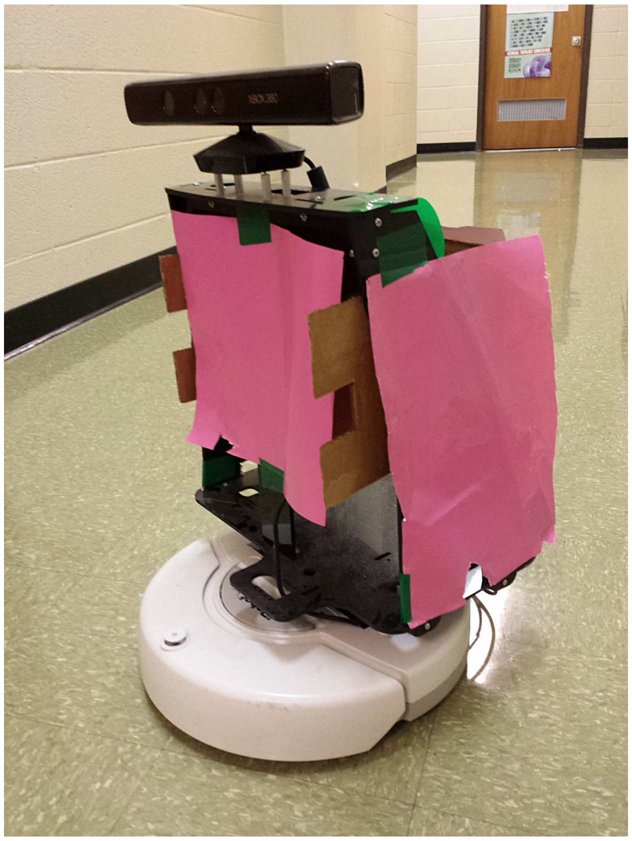


Figure A.1: Patroller J with distinct color patches for detection by a blob finder.

- *Release Soft* - Only applies in the Bin states

In observations received from the expert, the Grab and Release actions are never observed due to the ambiguity involved. Portions of the trajectories that involve a grab or place action are removed before the start of IRL (with the exception of the occlusion free control method)

Transitions:

The transition function is deterministic in all variables except for the previous action time, which is stochastic. To create this, we start by creating a transition function that is deterministic in the variables Location, Holding ball, and ball type and considers all Time for previous action variables equally probable for every state and action. Now, assuming we know the true action taken we build a distribution over action times from the set of observed trajectories for each action. To create the final transition function, we multiply this distribution by the transition function generated previously.

Reward Features:

12 features total:

- Release ball type X in Bin 1
- Release ball type X in Bin 2
- Handle ball type X softly

where X = 1, 2, 3, 4

Converting observations to trajectories

The expert's arm was tracked using a motion capture system and the sorting task was recorded with an RGB camera. We discretized the position of the wrist into one of four locations as can be seen in figure A.2, and recorded the time the transition between states took. We then manually identified the ball type handled in each trajectory.

The reward functions learned by each IRL method are shown in table A.1.

Robot MDP

States:

- *Ball Type X at sorting location*
- *Ball Type X at bin 1* *terminal*
- *Ball Type X at bin 2* *terminal*
- *Dropped ball X* *terminal*

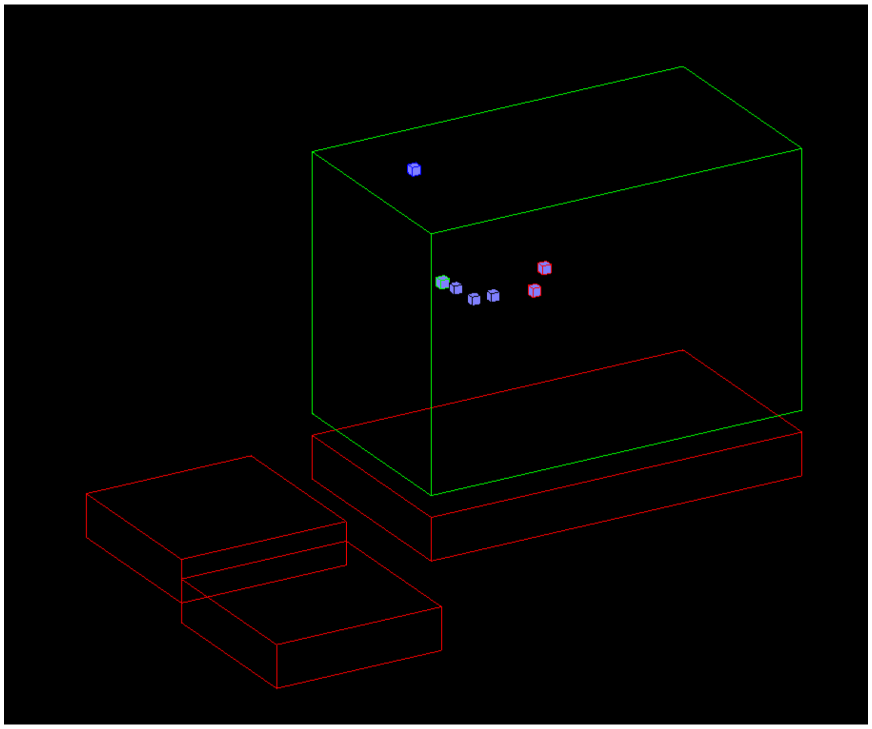


Figure A.2: Visualization of the motion capture points attached to the expert's arm with discretized regions corresponding to the four MDP locations (*On table, Center Area, Bin 1, Bin 2*) superimposed. The wrist is currently in the highlighted state *Center Area*.

Ball sorting task												
Method	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7	ϕ_8	ϕ_9	ϕ_{10}	ϕ_{11}	ϕ_{12}
MaxEnt IRL	0.2213	0.2456	0	0	0	0	0.27	0.259	0	0	0	0
IRL*	0.8331	1.697	-0.4020	-0.44	0.25	0.26	1.08	1.07	1.08	1.06	0.61	-0.78
Hidden Data EM	0.1622	0.1033	0	0	0	0	0.1879	0.1647	0	0.37	0	0.0109
Occlusion-free control	0.0002	0.0001	0	0	0	0	0.0002	0.00006	0	0.5	0	0.4991

Table A.1: Feature weights learned by all four methods from ball sorting data. ϕ_1 - ϕ_4: reward for sorting ball types 1-4 into bin 1, ϕ_5 - ϕ_8: reward for sorting ball types 1-4 into bin 2, and ϕ_9 - ϕ_{12}: reward for handling ball types 1-4 gently.

Method	Ball type hard and black	Ball type soft and red	Ball type hard and white	Ball type soft and green
MaxEnt IRL	To Bin 1 firmly	To Bin 1 firmly	To Bin 2 firmly	To Bin 2 firmly
IRL*	To Bin 1 gently	To Bin 1 gently	To Bin 2 gently	To Bin 2 firmly
Hidden Data EM	To Bin 1 firmly	To Bin 1 gently	To Bin 2 firmly	To Bin 2 gently
Occlusion-free control	To Bin 1 firmly	To Bin 1 gently	To Bin 2 firmly	To Bin 2 gently

Table A.2: Optimal policies learned by all four methods from the ball sorting data with hidden variables. While the state consists of multiple variables, we show the robot's action map for the main state variable of ball type here. Underlined actions are erroneous.

where X = 1, 2, 3, 4

Actions:

- *Sort to Bin 1 Firmly*
- *Sort to Bin 1 Softly*
- *Sort to Bin 2 Firmly*
- *Sort to Bin 2 Softly*

Transitions:

The transition function modeled successfully sorting a ball softly at 83% and firmly at 89%. A failure results in transitioning to the dropped state.

Reward Features:

13 features total

- 1 - 12 are the same as the expert's MDP
- 13 - a feature for entering the Dropped ball state, which is given weight of zero.

The optimum policies for the reward functions learned by each IRL method is shown in table A.2.

This experiment was implemented using the ROS Indigo system. We used a PhantomX Pincher arm to perform the sorting. We used the turtlebot_arm package with kinematic modifications for a pinching gripper to provide movement planning. This arm has no force sensors in its gripper so to simulate grabbing the balls hard and soft we instead designate grabbing hard as closing the gripper much more than the soft version, then made the balls approximately the same size. "Firm" balls are thus balls that can be crushed without damaging them, firm foam and heavy duty scouring pads were used for this purpose. Soft balls are simply balls made of play doh.

A.1.2 Penetrating a patrol under occlusion: details for section 7.1.4

This experiment was implemented using the ROS Fuerte system. The AMCL package provided localization, move_base provided local and global navigation, and a map of the relevant area was provided.

L utilized algorithms 1 (Subject) and 2 (ExecutePolicy) to perform its task. The patrollers solved their MDP before the experiment started and moved according to the optimum policy found using algorithm 2 (ExecutePolicy). Interaction was handled by designating one patroller to step when it sees the other patroller within a certain distance and the other patroller simply reduces its maximum speed.

The Predict() algorithm in this experiment jointly projects the provided policies from the last point each patroller was seen. For this experiment, a single projection was used to save time.

As can be seen in the ExecutePolicy algorithm 2 the robots were moved by translating the MDP states into way-points and giving them to the ROS move_base package. move_base at this time

Algorithm 1 Subject

```
Input: obstime, maxTime, startPosition, goalPosition, PatrollerMDP/R, AttackerMDP/R, MaxPrediction
  while time $\leq$ obstime do
    Percepts $\leftarrow$ ObservePatrollers()
  Trajectories $\leftarrow$ ConvertPerceptsToTrajectory(Percepts)
  PatrollerPolicies $\leftarrow$ IRL(PatrollerMDP/R, Trajectories)
  repeat
    lastSeenPosition $\leftarrow$ ObservePatrollers()
    V $\leftarrow$ ValueIteration(AttackerMDP/R, Predict(PatrollerMDP/R, PatrollerPolicies, lastSeenPosition))
    bestTime $\leftarrow$ -1
    bestValue $\leftarrow$ 0
    for t = 1 to MaxPrediction do
      if V( (StartPosition, t) ) > bestValue then
        bestTime $\leftarrow$ t
        bestValue $\leftarrow$ V( (StartPosition, t) )
        Policy $\leftarrow$ CreatePolicy(V)
  until bestTime $\geq$ 0 or time $\geq$ maxTime
  while time $\leq$ bestTime and time $\leq$ maxTime do
    sleep()
  while curPosition $\neq$ goalPosition do
    if time > maxTime then
      exit(FAILURE)
    ExecutePolicy(Policy, (curPosition, time) )
  exit(SUCCESS)
```

Algorithm 2 ExecutePolicy

```
Input: Policy, CurrentState
  action $\leftarrow$ Policy(CurrentState)
  nextState $\leftarrow$ IntendedNextState (CurrentState, action)
  nextWaypoint $\leftarrow$ centerPoint (nextState)
  nextWaypoint $\leftarrow$ nextWaypoint + RandomGaussian(0, 0.1)
  {Small random noise added to waypoint to get around move_base bugs}
  sendWayPointCommand(nextWaypoint)
```

was very buggy and would disregard repeated commands if they were similar to each other, to get around this and ensure the robot moved when a way-point was given a small amount of Gaussian noise was added to it. This movement algorithm is also used by both patrollers once they have solved their individual MDPs.

Patroller MDP

Two MDPs were created, one for each map the experiment was performed on.
Actions:

- *Move Forward*
- *Turn Left*
- *Turn Right*
- *Turn Around*
- *Stop*

Transitions:

Actions caused the agent to move to the intended state 66% of the time with the remaining mass distributed to the intended next states of the other actions evenly.

States:

Smaller Map:

76 states representing the patrolling map discretized into 19 squares, and the patroller's orientation discretized into the 4 cardinal directions

Larger Map:

96 states representing the patrolling map discretized into 24 squares, and the patroller's orientation discretized into the 4 cardinal directions

Reward Features:

Smaller Map

To ease the learning process, reward features were designed to restrict the policy space to considering plausible policies.

14 features

- Move Forward
- Turnaround X distance from the center point

True rewards: 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, .5, -1, -1, -1

The turn around features are mapped to all states that are ideally reachable from the center point of the map in X actions. To ease learning, we condensed these to just two parameters P when given to Nelder-Mead (though the full feature expectations were still utilized). These two parameters corresponded to a magnitude value and the center-point of a Gaussian function. The reward weights are then generated from these two parameters by setting $\theta_1 = P[1]$, and all other $\theta_n = P[1] * \mathcal{N}(n-1, 0.632)$. In essence, rather than individually choosing weights for each feature this technique enforced a Gaussian shape onto the weights, ensuring that just one would receive a maximum value while the others much lower.

Larger Map:

5 features

- Move Forward
- Is in a room
- Turn around in a hallway
- Move from a room to the hallway
- Move from a hallway to a room

True rewards: 1, -1, -1, 0, 0

Subject MDP

The subject MDP is similar to the patroller's with the following changes:

States:

The subject robot moves through a larger area than the patrollers (particularly in the larger map) so extra map space is added to the subject robot's MDP. These states are then extended with a time variable to model the dynamic environment the attacker must move through. 30 timesteps were considered, timesteps are of length 4 seconds for the smaller map and 4.5 seconds for the larger.

Actions:

- *Move forward*
- *Turn left*
- *Turn right*

Transitions:

Actions caused the agent to move to the intended state 90% or 95% of the time (evenly distributed throughout runs) with the remaining mass distributed to the intended next states of the other actions evenly. Regardless, the agent always moves forward one time step per action.

Reward Function:

There are two parts to the subject's reward function:

- Reward for being at the goal position
- Penalty for being observed

The reward was simply set to 10 for all states at the goal position for all timesteps. To set the penalty, the predicted positions of each patroller were queried for each timestep. For each state, if the state was within detection distance from a patroller (3 states) and in the correct direction to be detected it received a penalty of -10.

The method used to observe the patrollers varied between simulation and reality; we discuss this further in the following sections. Percepts are sub-timestep data points of the position of each patroller in continuous. These must be converted into discrete MDP states and the action performed inferred at each timestep.

Simulation

Simulation runs were provided by the Stage ROS package. This package provides simulated laser range finders to enable the AMCL localization system to be used however it was unable to provide adequate simulated RGB cameras for detecting robots visually. We therefore queried the simulator for the true position of each robot whenever needed, and filtered the results to simulate a limited range camera by only giving the subject robot a percept if it is within specified observable states.

Percepts are the tuple (x, y, ψ, t) corresponding to the position and orientation of the robot at time t. Each percept is simply classified according to which MDP state it is within, then the classified percepts are grouped together by timestep and the majority state becomes the state used. The action at each timestep is calculated by simply using the most likely action given two subsequent states.

Physical

All robots used are Turtlebot version 1.0. These robots are constructed of an iRobot Create with an added gyroscope sensor, a Microsoft Kinect, and a body frame. The patrollers used a non-standard body frame, as can be seen in figure A.1.

The primary difference from simulation is in the detection of robots. We use the CMVision ROS package to identify certain colors in the RGB video stream received from the turtlebot's camera. We take the largest identified blob of each configured color and query the range finder for the distance to the center point. By combining this range and bearing information with the robot's current belief we are able to estimate the position of the other robot. Unfortunately, our experiment has no way of observing the orientation of an observed robot. Instead we compared subsequent percepts and, assuming the robot does not move backward, assume it is facing in the direction it is moving.

A.1.3 Penetrating a patrol w/ unknown T(): details for section 7.3.4

This experiment was implemented using the ROS Indigo system. The AMCL package provided localization, move_base provided local and global navigation, and a map of the relevant area was provided.

Most details given in appendix section A.1.3 apply to this experiment. For completeness, we repeat them here. To summarize the changes, the MDPs were discretized to a finer extent to allow for more accurate transition functions to be learned, and subsequently the reward features used were revised for the new MDPs. The *Predict*() algorithm in this experiment is modified to jointly projects the provided policies numerous times and average the results from the last point each patroller was seen. This way, the learned transition function affects the prediction quality as well as the policies learned.

In between the path planning component of ROS (move_base) and the turtlebot driver was inserted a small node with acted to simulate a damaged wheel. A percentage could be given to this node for the left wheel, and the speed this wheel is turned at would be multiplied by this amount before given to the driver, with the navigation system unaware this modification was happening. This resulted in oscillating behavior as the navigation system constantly had to slow down and correct its path.

Patroller MDP

Two MDPs were created, one for each map the experiment was performed on.

Actions:

- *Move Forward*
- *Turn Left*
- *Turn Right*
- *Stop*

Note: the Turn Around action is not used

Transitions:

The transition functions were either learned using the technique in chapter 5 or fixed to specific values: actions caused the agent to move to the intended state X% of the time (X = 70, 80, 90, or 95) with the remaining mass distributed to the same state it is currently in (Stop error model).

States:

Smaller Map:

124 states representing the patrolling map discretized into 31 squares, and the patroller's orientation discretized into the 4 cardinal directions

Larger Map:

212 states representing the patrolling map discretized into 53 squares, and the patroller's orientation discretized into the 4 cardinal directions

Reward Features:

To ease the learning process, reward features were designed to restrict the policy space to considering plausible policies.

Smaller Map

6 features

- Move Forward
- Turn in the vertical portion of the hallway
- Turn in the first two states of the horizontal portion of the hallway
- Turn between the 2nd and 4th states of the horizontal portion of the hallway
- Turn between the 4th and 6th states of the horizontal portion of the hallway
- Turn between the 6th and 8th states of the horizontal portion of the hallway

True rewards: 1, -1, -1, -1, 0.5, -1

These "grouped" features each match multiple states and overlap on the horizontal sections of the hallway. These features restrict the space of optimal policies while still allowing a great many "patrolling"-like behaviors. As the run time of the solver used, Nelder-Mead simplex, scales with the number of parameters it is important that these be kept to as few as possible.

Larger Map:

5 features

- Move Forward
- Is in a room
- Turn around in a hallway
- Move from a room to the hallway
- Move from a hallway to a room

True rewards: 1, -1, .1, 0, 0;

Subject MDP

The subject MDP is similar to the patroller's with the following changes:

States:

The subject robot moves through a larger area than the patrollers (particularly in the larger map) so extra map space is added to the subject robot's MDP. These states are then extended with a time variable to model the dynamic environment the attacker must move through. 30 timesteps were considered for the smaller map and 60 for the larger, timesteps are of length 2 seconds.

Actions:

- *Move forward*
- *Turn left*
- *Turn right*

Transitions:

Actions caused the agent to move to the intended state 97.5% of the time (evenly distributed throughout runs) with the remaining mass distributed to the intended next states of the other actions evenly. Regardless, the agent always moves forward one time step per action.

Reward Function:

There are two parts to the subject's reward function:

- Reward for being at the goal position
- Penalty for being observed

The reward was simply set to 10 for all states at the goal position for all timesteps. To set the penalty, the predicted positions of each patroller were queried for each timestep. For each state, if the state was within detection distance from a patroller (6 states) and in the correct direction to be detected it received a penalty of -10.

Observations

The method used to observe the patrollers varied between simulation and reality; we discuss this further in the following sections. Percepts are sub-timestep data points of the position of each patroller in continuous. These must be converted into discrete MDP states and the action performed inferred at each timestep.

Simulation

Simulation runs were provided by the Stage ROS package. This package provides simulated laser range finders to enable the AMCL localization system to be used however it was unable to provide adequate simulated RGB cameras for detecting robots visually. We therefore queried the simulator for the true position of each robot whenever needed, and filtered the results to simulate a limited range camera by only giving the subject robot a percept if it is within specified observable states.

Percepts are the tuple (x, y, ψ, t) corresponding to the position and orientation of the robot at time t. Each percept is simply classified according to which MDP state it is within, then the classified percepts are grouped together by timestep and the majority state becomes the state used. The action at each timestep is calculated by simply using the most likely action given two subsequent states.

Physical

Robots used are Turtlebot version 2.0 (L and I) and 1.0 (J). Version 2.0 is constructed of a Yujin robotics Kobuki base, a Microsoft Kinect, and a body frame. Version 1.0 robots are constructed of an iRobot Create with an added gyroscope sensor, a Microsoft Kinect, and a body frame. Patroller J used a non-standard body frame, as can be seen in figure A.1.

The primary difference from simulation is in the detection of robots. We use the CMVision ROS package to identify certain colors in the RGB video stream received from the turtlebot's camera. We take the largest identified blob of each configured color and query the range finder for the distance to the center point. By combining this range and bearing information with the robot's current belief we are able to estimate the position of the other robot. Unfortunately, our experiment has no way of observing the orientation of an observed robot. Instead we compared subsequent percepts and, assuming the robot does not move backward, assume it is facing in the direction it is moving.

A.1.4 Penetrating a patrol w/ Hidden Data EM: details for section 7.2.2

This experiment was implemented using the ROS Hydro system.

Most details given in appendix section A.1.3 apply to this experiment. For completeness, we repeat them here. To summarize, the reward features of our smaller environment were expanded to better illustrate the difference between the *Hidden data EM* methods and mIRL*+ne, as the larger feature count allowed lengthened the number of iterations needed for Nelder-Mead simplex and expanded the space of possible optimum policies.

Patroller MDP

Actions:

- *Move Forward*
- *Turn Left*
- *Turn Right*
- *Stop*

The Turn Around action is not used

Transitions:

The transition functions were fixed to specific values: actions caused the agent to move to the intended state 92.5% of the time with the remaining mass distributed to the same state it is currently in (Stop error model).

States:

124 states representing the patrolling map discretized into 31 squares, and the patroller's orientation discretized into the 4 cardinal directions

Reward Features:

16 features

- Move Forward
- Turn back at Manhattan distance X from the center state
- Feature which matches any state-action pair that does not match any other feature (catch-all)

True rewards: 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0.5, -1, -1, -1

These simple features allow a much large policy space to be considered than in our previous experiments. They specifically match turning towards the center point, so actions that turn away from the center point instead match the catch-all feature. This helps prevent policies that specify "turn around in place forever" behaviors during the solving process as one feature temporarily gains more value than others.

Subject MDP

The subject MDP is similar to the patroller's with the following changes:

States:

The subject robot moves through a larger area than the patrollers (particularly in the larger map) so extra map space is added to the subject robot's MDP. These states are then extended with a time variable to model the dynamic environment the attacker must move through. 30 timesteps were considered, timesteps are of length 2 seconds.

Actions:

- *Move forward*
- *Turn left*
- *Turn right*

Transitions:

Actions caused the agent to move to the intended state 97.5% of the time (evenly distributed throughout runs) with the remaining mass distributed to the intended next states of the other actions evenly. Regardless, the agent always moves forward one time step per action.

Reward Function:

There are two parts to the subject's reward function:

- Reward for being at the goal position
- Penalty for being observed

The reward was simply set to 10 for all states at the goal position for all timesteps. To set the penalty, the predicted positions of each patroller were queried for each timestep. For each state, if the state was within detection distance from a patroller (6 states) and in the correct direction to be detected it received a penalty of -10.

Percepts are sub-timestep data points of the position of each patroller in continuous. These must be converted into discrete MDP states and the action performed inferred at each timestep.

Simulation

Simulation runs were provided by the Stage ROS package. This package provides simulated laser range finders to enable the AMCL localization system to be used however it was unable to provide adequate simulated RGB cameras for detecting robots visually. We therefore queried the simulator for the true position of each robot whenever needed, and filtered the results to simulate a limited range camera by only giving the subject robot a percept if it is within specified observable states.

Percepts are the tuple (x, y, ψ, t) corresponding to the position and orientation of the robot at time t. Each percept is simply classified according to which MDP state it is within, then the classified percepts are grouped together by timestep and the majority state becomes the state used. The action at each timestep is calculated by simply using the most likely action given two subsequent states.

Physical

Robots used are Turtlebot version 2.0 (L and I) and 1.0 (J). Version 2.0 is constructed of a Yujin robotics Kobuki base, a Microsoft Kinect, and a body frame. Version 1.0 robots are constructed of an iRobot Create with an added gyroscope sensor, a Microsoft Kinect, and a body frame. Patroller J used a non-standard body frame, as can be seen in figure A.1.

The primary difference from simulation is in the detection of robots. We use the CMVision ROS package to identify certain colors in the RGB video stream received from the turtlebot's camera. We take the largest identified blob of each configured color and query the range finder for the distance to the center point. By combining this range and bearing information with the robot's current belief we are able to estimate the position of the other robot. Unfortunately, our experiment has no way of observing the orientation of an observed robot. Instead we compared subsequent percepts and, assuming the robot does not move backward, assume it is facing in the direction it is moving.

A.2 Proofs

A.2.1 Non-Convexity of IRL*

The non-convexity of IRL* arises because of the use of a probability distribution defined using all states but constrained over only a subset of them. If the probability distribution only used the subset of non-occluded states then the resulting program would be convex but may admit no feasible solution.

Because of the under-constrained feature expectations, it is possible for more than one distribution over policies to match these limited constraints while having the same entropy: the maximum achievable. Call the feature weight vectors that produce these two distributions X and Y, X may not equal Y or else the distributions would be the same and X and Y must have the same magnitude or else the distributions generated by them would have different entropy allowing for one to be chosen over the other.

Theorem: *If X and Y each produce distributions with equal, maximum entropy that match constraints then all points between X and Y must also result in distributions with the same entropy whose feature expectations match constraints or else the program is non-convex.*

Proof: The points between X and Y must have less magnitude than either one due to this being a chord line between them. Now, assume all these points do in fact result in distributions with the same entropy that match the reduced feature expectations, then we conclude that the features in the occluded space have no effect on the distribution and therefore can be removed, creating a convex program. Otherwise, all points between X and Y will necessarily produce distributions that have more entropy due to the points on the line between X and Y having less magnitude than either point. The distributions created using these interior points may not match the feature expectations; otherwise, due to their increased entropy, they would be chosen instead for X or Y. We therefore conclude that the points on the line between X and Y are not in the feasible set, therefore the feasible set is non-convex and the program is non-convex.

References

[1] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[2] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1, 2007.

[3] Pieter Abbeel and AY Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, page 1, 2004.

[4] Navid Aghasadeghi and Timothy Bretl. Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1561–1566. IEEE, 2011.

[5] Noa Agmon, Sarit Kraus, and Gal a. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *ICRA*, pages 2339–2345, 2008.

[6] Aris Alissandrakis, Chrystopher L Nehaniv, and Kerstin Dautenhahn. Correspondence mapping induced state and action metrics for robotic imitation. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics*, 37(2):299–307, 2007.

[7] Narayanaswamy Balakrishnan. *Continuous multivariate distributions.* Wiley Online Library, 2001.

[8] Yonathan Bard. Estimation of state probabilities using the maximum entropy principle. *IBM Journal of Research and Development*, 24(5):563–569, 1980.

[9] Kenneth Bogert and Prashant Doshi. Multi-robot inverse reinforcement learning under occlusion with interactions. In *AAMAS*, pages 173–180, 2014.

[10] Kenneth Bogert and Prashant Doshi. Toward estimating others' transition models under occlusion for multi-robot IRL. In *IJCAI*, pages 1867–1873. AAAI Press, 2015.

[11] Kenneth Bogert, Jonathan Feng-Shun Lin, Prashant Doshi, and Dana Kulic. Expectation-maximization for inverse reinforcement learning with hidden data. In *AAMAS*, pages 1034–1042, 2016.

[12] Joseph-Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, and Claudia A Sagastizábal. *Numerical optimization: theoretical and practical aspects.* Springer Science & Business Media, 2006.

[13] A Boularias, Jens Kober, and J Peters. Relative entropy inverse reinforcement learning. In *AISTATS*, pages 182–189, 2011.

[14] Abdeslam Boularias, O Krömer, and J Peters. Structured apprenticeship learning. *Machine Learning and Knowledge Discovery in Databases*, pages 227–242, 2012.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[16] R. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[17] Sonia Chernova and Manuela Veloso. Teaching multi-robot coordination using demonstration of communication and state sharing (short paper). In *AAMAS 2008*, pages 1183–1186, 2008.

[18] J Choi and Kee-eung Kim. Inverse reinforcement learning in partially observable environments. *Machine Learning Research*, 12:691–730, 2011.

[19] Jaedeug Choi and Kee-eung Kim. Inverse Reinforcement Learning in Partially Observable Environments. In *IJCAI*, pages 1028–1033, 2009.

[20] Claudia Goldman and Shlomo Zilberstein. Communication-based decomposition mechanisms for decentralized mdps. *Journal of Artificial Intelligence Research*, 32:169–202, 2008.

[21] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

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

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

[24] Beomjoon Kim and Joelle Pineau. Socially adaptive path planning in human environments using inverse reinforcement learning. *International Journal of Social Robotics*, 8(1):51–66, 2016.

[25] Dongho Kim, Catherine Breslin, Pirros Tsiakoulis, Milica Gasic, Matthew Henderson, and Steve J Young. Inverse reinforcement learning for micro-turn management. In *INTERSPEECH*, pages 328–332, 2014.

[26] Kris M. Kitani, Brian D. Ziebart, J. Andrew Bagnell, and Martial Hebert. Activity forecasting. *Computer Vision ECCV*, pages 201–214, 2012.

[27] Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

[28] Edouard Klein and M Geist. Inverse Reinforcement Learning through Structured Classification. *Advances in Neural Information Processing Systems*, pages 1–9, 2012.

[29] Jeffrey C Lagarias, James A Reeds, Margaret H Wright, and Paul E Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on optimization*, 9(1):112–147, 1998.

[30] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[31] Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 31–46. Springer, 2009.

[32] Ken IM McKinnon. Convergence of the nelder–mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1998.

[33] Geoffrey McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.

[34] T Michael and I Jordan. Reinforcement learning algorithm for partially observable markov decision problems. *Proceedings of the Advances in Neural Information Processing Systems*, pages 345–352, 1995.

[35] Mathew Monfort and Brian D Ziebart. Intent Prediction and Trajectory Forecasting via Predictive Inverse Linear-Quadratic Regulation. In *AAAI*, 2015.

[36] Katharina Muelling, Abdeslam Boularias, Betty Mohler, Bernhard Schölkopf, and Jan Peters. Learning strategies in table tennis using inverse reinforcement learning. *Biological cybernetics*, 108(5):603–619, 2014.

[37] Sriraam Natarajan, Gautam Kunapuli, Kshitij Judah, Prasad Tadepalli, Kristian Kersting, and Jude Shavlik. Multi-agent inverse reinforcement learning. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pages 395–400. IEEE, 2010.

[38] Ronald C Neath et al. On convergence properties of the monte carlo em algorithm. In *Advances in Modern Statistical Theory and Applications: A Festschrift in Honor of Morris L. Eaton*, pages 43–62. Institute of Mathematical Statistics, 2013.

[39] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[40] Gergely Neu and Csaba Szepesvari. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *UAI*, pages 295–302, 2007.

[41] Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670, 2000.

[42] Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. Inverse reinforcement learning with locally consistent reward functions. In *Advances in Neural Information Processing Systems*, pages 1747–1755, 2015.

[43] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[44] RW Obermayer and Frederick A Muckler. *On the inverse optimal control problem in manual control systems*, volume 208. NASA, 1965.

[45] Takayuki Osogami and Rudy Raymond. Map matching with inverse reinforcement learning. In *IJCAI*, pages 2547–2553. AAAI Press, 2013.

[46] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[47] Deepak Ramachandran. Bayesian inverse reinforcement learning. In *IJCAI*, pages 2586–2591, 2007.

[48] Nathan D. Ratliff, J. Andrew Bagnell, and Martin a. Zinkevich. Maximum Margin Planning. In *ICML*, pages 729–736, 2006.

[49] Christian Robert and George Casella. *Monte Carlo Statistical Methods.* Springer-Verlag New York, 2 edition, 2004.

[50] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 2 edition, 2003.

[51] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[52] Masamichi Shimosaka, Kentaro Nishi, Junichi Sato, and Hirokatsu Kataoka. Predicting driving behavior using inverse reinforcement learning with multiple reward functions towards environmental diversity. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 567–572. IEEE, 2015.

[53] MTJ Spaan and FS Melo. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *AAMAS*, pages 525–532, 2008.

[54] Jacob Steinhardt and Serra Street. Adaptivity and Optimism : An Improved Exponentiated Gradient Algorithm. *Journal of Machine Learning Research*, 32(2012), 2014.

[55] Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, pages 1449–1456, 2007.

[56] Sebastian Thrun and John J Leonard. Simultaneous localization and mapping. In *Handbook of Robotics*, pages 871–889. Springer, 2008.

[57] Aris Valtazanos and S Ramamoorthy. Bayesian interaction shaping: learning to influence strategic interactions in mixed robotic domains. In *AAMAS*, pages 6–10, 2013.

[58] Adam Vogel, Deepak Ramachandran, Rakesh Gupta, and Antoine Raux. Improving hybrid vehicle fuel efficiency using inverse reinforcement learning. In *AAAI*, pages 384–390. AAAI Press, 2012.

[59] Shaojung Wang, Dale Schuurmans, and Yunxin Zhao. The Latent Maximum Entropy Principle. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(2):8:1–8:42, 2012.

[60] Brian D Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy.* PhD thesis, 2010.

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

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