

2016-04-25

Reinforcement Learning from Demonstration

Halit Bener Suay
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/etd-dissertations>

Repository Citation

Suay, H. B. (2016). *Reinforcement Learning from Demonstration*. Retrieved from <https://digitalcommons.wpi.edu/etd-dissertations/173>

This dissertation is brought to you for free and open access by [Digital WPI](#). It has been accepted for inclusion in Doctoral Dissertations (All Dissertations, All Years) by an authorized administrator of Digital WPI. For more information, please contact wpi-etd@wpi.edu.

Reinforcement Learning from Demonstration

by

Halit Bener Suay

A Dissertation

Submitted to the Faculty

of the

Worcester Polytechnic Institute

In Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

in

Robotics Engineering

April 2016

Approved as to style and content by:



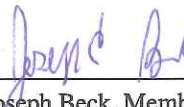
4/22/16

Sonia Chernova, Advisor



4/20/2016

Dmitry Berenson, Member

 4/20/16

Joseph Beck, Member



4/22/16

Matthew E. Taylor, Member

© Copyright by Halit Bener Suay 2016
All rights reserved

Abstract

Off-the-shelf Reinforcement Learning (RL) algorithms suffer from slow learning performance, partly because they are expected to learn a task from scratch merely through an agent's own experience. In this thesis, we show that learning from scratch is a limiting factor for the learning performance, and that when prior knowledge is available RL agents can learn a task faster. We evaluate relevant previous work and our own algorithms in various experiments.

Our first contribution is the first implementation and evaluation of an existing interactive RL algorithm in a real-world domain with a humanoid robot. Interactive RL was evaluated in a simulated domain which motivated us for evaluating its practicality on a robot. Our evaluation shows that guidance reduces learning time, and that its positive effects increase with state space size.

A natural follow up question after our first evaluation was, how do some other previous works compare to interactive RL. Our second contribution is an analysis of a user study, where naïve human teachers demonstrated a real-world object catching with a humanoid robot. We present the first comparison of several previous works in a common real-world domain with a user study.

One conclusion of the user study was the high potential of RL despite poor usability due to slow learning rate. As an effort to improve the learning efficiency of RL learners, our third contribution is a novel human-agent knowledge transfer algorithm. Using demonstrations from three teachers with varying expertise in a simulated domain, we show that regardless of the skill level, human demonstrations can improve the asymptotic performance of an RL agent.

As an alternative approach for encoding human knowledge in RL, we investigated the use of reward shaping. Our final contributions are Static Inverse Reinforcement Learning Shaping and Dynamic Inverse Reinforcement Learning Shaping algorithms that use human demonstrations for recovering a shaping reward function. Our experiments in simulated domains show that our approach outperforms the state-of-the-art in cumulative reward, learning rate and asymptotic performance.

Overall we show that human demonstrators with varying skills can help RL agents to learn tasks more efficiently.

Acknowledgments

Research presented in different chapters of this thesis was supported by Worcester Polytechnic Institute, NSF Award 1149876, ONR Award N00014-14-1-0795. This research has taken place in part at the Intelligent Robot Learning (IRL) Lab, Washington State University and research presented in Chapter 6 was supported in part by grants AFRL FA8750-14-1-0069, AFRL FA8750-14-1-0070, NSF IIS-1149917, NSF IIS-1319412, USDA 2014-67021-22174, and a Google Research Award.

Contents

1	Introduction	1
1.1	Contributions and Overview	3
2	Background	5
2.1	Learning from Demonstration	5
2.2	Reinforcement Learning	7
2.3	Inverse Reinforcement Learning	10
2.4	Transfer Learning	13
2.5	Reward Shaping	15
2.6	Reinforcement Learning from Demonstration	16
3	Effect of Human Guidance and State Space Size on Interactive Reinforcement Learning	18
3.1	Overview	19
3.2	Description of Interactive RL Algorithm	20
3.3	Experimental Setup	23
3.3.1	Domain	23
3.3.2	Experimental Conditions	25
3.4	Results	28
3.5	Discussion	29
4	A Comparison of Three Learning from Demonstration Algorithms	33
4.1	Overview	33
4.2	Description of Selected LfD Algorithms	35
4.3	Evaluation	37
4.3.1	User Study	38
4.3.2	Experimental Setup	39
4.4	Qualitative Results	41
4.5	Quantitative Results	46
4.6	Discussion	53
5	Human-Agent Transfer: HAT	57
5.1	Overview	58
5.2	Description of HAT Algorithm	59

5.3	Evaluation	61
5.3.1	Keepaway	62
5.3.2	Experimental Setup	63
5.4	Results	65
5.5	Discussion	73
6	Learning from Demonstration for Shaping through Inverse Reinforcement Learning	75
6.1	Overview	75
6.2	Methodology	78
6.3	Experimental Validation	82
6.3.1	Maze	83
6.3.2	Mario AI	88
6.4	Discussion	93
7	Discussion and Conclusion	96

List of Tables

3.1	Guidance messages restrict action selection to the set of available actions as shown. The first letter <i>L</i> and <i>R</i> denotes either Left or Right hand depending on the hand with which the action is executed.	25
3.2	Experiment statistics. All results are shown as the average of three trials for each experiment. Number of states discovered (#States), total number of actions performed (#Actions), sum of the positive and negative rewards given (Cumul. Rew.), total number of guidance given (Total #G.) and training time.	30
5.1	This table shows the jumpstart, final reward and total reward metrics for Figure 5.3. Values in bold have statistically significant differences in comparison to the No Prior method ($p < 0.05$).	67
5.2	This table shows the jumpstart, final reward and total reward metrics for Figure 5.4, where all HAT methods use Probabilistic Policy Reuse with 20 episodes of demonstrated play. Values in bold have statistically significant differences in comparison to the No Prior method.	68
5.3	This table shows the jumpstart, final reward and total reward metrics for Figure 5.5, where all HAT methods use Probabilistic Policy Reuse with 20 demonstrated episodes. Values in bold have statistically significant differences in comparison to the No Prior method (not shown).	70
5.4	This table shows the jumpstart, final reward and total reward metrics for Figure 5.6, where all HAT methods use Probabilistic Policy Reuse. All demonstrations are 20 episodes, recorded by Subject B. Values in bold have statistically significant differences in comparison to the No Prior method (not shown).	73
6.1	Average initial (first episode) performance, asymptotic performance, and cumulative reward with standard deviations for all agents in the Maze domain. Maximum values are shown in bold. Higher values are better. SAR: SARSA, HMN: average of the reward human demonstrator received in 10 demonstrations.	85

6.2	Statistical analysis for the Maze domain results show p-values for two-sample <i>t</i> -tests. SAR stands for SARSA. Columns show the statistical significance of the difference between the initial (first episode) performance, asymptotic (last episode) performance, and cumulative reward (i.e. total reward throughout the trial). * $p \leq 0.05$, ** $p \leq 0.01$, *** $p \leq 0.001$, **** $p \leq 0.0001$	86
6.3	Performance metrics for the agents in Mario AI: initial performance, asymptotic performance, cumulative reward (average of 10 trials). Maximum values are shown in bold. Higher values are better.	88
6.4	Statistical analysis for the Mario AI domain results show p-values for two-sample <i>t</i> -tests. QLR stands for Q-Learning. Columns show the statistical significance of the difference between the initial performance, asymptotic (final) performance, and cumulative reward (i.e. total reward throughout the trial).	89

List of Figures

3.1	Screen-shots of our interface (<i>Top row</i>) and Sophie’s Kitchen (<i>Bottom row</i>). (a) shows positive reward given with a left click and drag upwards; (b) shows guidance given with a right click on the object that the teacher wants the agent to interact with.	22
3.2	Experiment Setup. (a) Web camera captures the table and the robot for the user interface. (b) The table is divided in three zones. Orange, cyan blue and purple show z_1 , z_2 , and z_3 respectively. The robot has an object at the tip of its right hand and is about to drop it in the right cup.	23
3.3	Images of different objects taken by robot’s camera. Light blue spots show SURF, red lines show the outline, and yellow rectangles show the bounding box or rectangle of interest of the object found.	26
3.4	User interaction and training time with (a,b) and without (c,d) guidance input. Figures (a,c) and (b,d) show the results obtained in small and large state space respectively. The y-axis and legend are common for all graphs.	29
4.1	(a) Experimental setup, (b) View of the table from the Nao’s on-board camera. Sweep zone: bottom-left portion of the figure. Pick-up zone: bottom-center portion of the figure. Any other portion of the image is Wait zone.	39
4.2	Percentage of correctly executed actions in all three of the methods. The red bars in each box show median values. The whiskers cover 3 interquartile range (IQR) and the red dots show the data points that lie between 3 and 5 IQR.	45
4.3	Example Int-RL Policies.	47
4.4	Int-RL User Statistics.	47
4.5	Examples of the networks demonstrated by the subjects. P-UP: Pick-Up, FBIPZ: Found a bug in Pick-Up Zone. Numbers in the end of the behaviors show their unique id numbers.	49
4.6	BNets User Statistics.	49
4.7	Example CBA Policies.	52
4.8	CBA User Statistics.	52

5.1	This diagram shows the distances and angles used to construct the 13 state variables used for learning with 3 keepers and 2 takers. Relevant objects are the 3 keepers (K) and the two takers (T), both ordered by distance from the ball, and the center of the field.	62
5.2	This figure shows a screenshot of the visualizer used for the human to demonstrate a policy in 3 vs. 2 Keepaway. The human controls the keeper with the ball (shown as a hollow white circle) by telling the agent when, and to whom, to pass. When no input is received, the keeper with the ball executes the <code>Hold</code> action, attempting to maintain possession of the ball.	64
5.3	This graph summarizes performance of Sarsa learning in Keepaway using four different algorithms. One demonstration of 20 episodes was used for all three HAT learners. Error bars show the standard error in the performance.	66
5.4	This graph summarizes performance of no prior learning and Probabilistic Policy Reuse learning using demonstrations from three different teachers. Each teacher performed demonstrations for 20 episodes. Error bars show the standard error in performance across 10 trials. . .	69
5.5	This graph summarizes performance of Probabilistic Policy Reuse learning using five different demonstration sets. Error bars show the standard error in performance across 10 trials.	71
5.6	This graph summarizes performance of Probabilistic Policy Reuse learning using three sets of demonstrations from Subject B recorded under different simulator conditions: normal, fast and with limited actions. Each demonstration set consists of 20 episodes. Error bars show the standard error in performance across 10 trials.	73
6.1	Maze domain. The state-space consists of two continuous variables x, y that define the horizontal and vertical location of the agent in the maze with reference to the lower-left corner. The agent starts from S and its goal is to navigate around the gray walls and reach the goal destination G.	83
6.2	Suboptimal demonstrations in the Maze domain.	83
6.3	Optimal path in the Maze domain.	84
6.4	Maze domain experiment results.	85
6.5	Mario domain experiment results.	88
6.6	Asymptotic performance of the DIS agent after 20,000 episodes in Mario AI. Here the performance is a function of the length of demonstrations (the number of state-action pairs recorded per demonstration). Each condition has 10 demonstrations. P-values obtained with two-sample t -tests between marked conditions for 10 trials. Values for the bar plot are the average of 10 trials.	92

6.7 Cumulative reward the DIS agent received during 20,000 episodes in Mario AI. Here the cumulative reward is a function of the length of demonstrations (the number of state-action pairs recorded per demonstration). Each condition has 10 demonstrations. P-values obtained with two-sample *t*-tests between marked conditions for 10 trials. 92

Chapter 1

Introduction

Reinforcement Learning (RL), the problem of mapping situations to actions in order to maximize a reward signal [1], provides a convenient sequential decision making framework for conducting research on task learning. One of the major limitations of RL solutions is the high number of repetitions of observation and action selections a typical RL agent has to perform before the agent can reach a desired level of task performance. Slow learning performance and the difficulty of defining a reward function are major obstacles preventing practical use of RL solutions.

One of the main causes of the slow learning rates of RL algorithms, is the fact that the agents are assumed to have no prior knowledge of the task they are expected to solve. In this thesis, we show that learning from scratch is a limiting factor for the learning performance of standard RL agents and that when prior knowledge is available, if the agent is provided means of leveraging this prior knowledge it can learn a task faster.

In RL problems the reward function can be thought of as a compact definition of the task. This reward function is often intuitive to define in most benchmarking domains. However, in the context of complex domains and real world problems, it may not be practical or feasible to define a task in advance as this may require a detailed knowledge of the state-space. This particular knowledge may be unavailable and hard-coding a

reward function may not be an option, or simply the definition of a task may change based on external requirements of users. In this thesis, we address this problem by implementing Inverse Reinforcement Learning for leveraging human demonstrations and computing a complementary reward function for an RL agent. We show that this secondary human reward function can be used for reward shaping for an RL agent to be able to fuse an environment reward signal with the learned reward signal.

The general theme of this thesis is *Reinforcement Learning from Demonstration*: a combination of RL and Learning from Demonstration (RLfD). In a nutshell, we define RLfD as the class of agents that can leverage prior knowledge, specifically the prior knowledge obtained from an entity other than the agent itself, in an RL setting. In this thesis, we show that this class of agents have superior learning capabilities compared to off-the-shelf RL learners.

Augmenting the abilities of RL agents with prior knowledge is a relatively new topic of research within the field of RL. Design of RLfD agents is an underconstrained process. Some of the challenges are:

- **Lack of infrastructure:** Widely used RL algorithms are not equipped with means for collecting priors about the task or possible solutions for the task.
- **Source of prior:** There are many possible sources for gathering prior knowledge with varying consistency and quality; a previously trained RL agent, an online database for directing queries, a previously generated policy for a task and a human teacher can all be valid sources of prior knowledge.
- **Use of prior:** The structure in which the prior is conveyed to an RL agent is crucial and it is not immediately clear how an agent can take advantage of this new input.

1.1 Contributions and Overview

In this thesis, we address all three of the challenges listed above. Our focus on obtaining the priors from humans and evaluating the use of human knowledge for RL agents in several ways. The overview of this thesis is as follows:

- In Chapter 2 we provide background material on Learning from Demonstration, the Reinforcement Learning problem, the Inverse Reinforcement Learning problem and Reinforcement Learning from Demonstration, specifically by the means of Reward Shaping and Transfer Learning.
- In Chapter 3 we present a study of a previously introduced RL algorithm that allows a human demonstrator to provide online guidance and rewards to the agent. Our study is the first evaluation of this method in real-world robotic systems [2]. We show that guidance reduces the number of explored state and the learning time, and that its positive effects increase with state space size. In the presence of guidance the teacher reduces the number of interactions earlier in the training process and the teacher provides more positive rewards.
- In Chapter 4 we present our findings on a user study where naïve teachers help a robot learn a task with three previously introduced learning from demonstration algorithms, one of which is the algorithm we evaluated with an expert teacher in the previous chapter. Here we evaluate the effect of human-agent interactions and present the results where we quantitatively and qualitatively compare the algorithms on a common real-world task [3].
- In Chapter 5 we show that it is possible to collect human demonstrations from teachers and use the teacher as an offline resource for modeling policies and for action selection advice. We contribute a novel RL algorithm Human-Agent Transfer (HAT) and evaluate the algorithm in a simulated soccer domain with three different teachers of varying skills [4]. We show that regardless of the skill level, human demonstrations can improve the asymptotic performance of an RL

agent.

- In Chapter 6 we introduce two novel RL algorithms, Static Inverse Reinforcement Learning Shaping (SIS) and Dynamic Inverse Reinforcement Learning Shaping (DIS) where human demonstrations serve as an offline resource for learning a secondary reward function [5] for reward shaping [6]. We present two sets of experiments in a two dimensional Maze domain, and the 27 dimensional Mario AI domain. We compare the performance of our algorithms to previously introduced reinforcement learning from demonstration algorithms. Our experiments show that our approach outperforms the state-of-the-art in cumulative reward, learning rate and asymptotic performance.

Overall the contributions of this thesis show that in several simulated domains RLfD agents can surpass the abilities of RL agents. An RL agent can benefit from human priors in order to learn a task, improve its learning rate of a task, and it can do so by combining priors with its own experience.

Chapter 2

Background

This section provides background information on topics discussed throughout this thesis. We organize the sub-sections to narrate *why* we talk about the topic, *what* has been done in that field of research, and finally *how* previous work relates to topics we present. We use a general-to-specific pattern with the hope to convey our deductive reasoning throughout this work.

2.1 Learning from Demonstration

In their book, Chernova and Thomaz define Robot Learning from Demonstrations as the field that explores techniques for learning a task policy from examples provided by a human teacher [7]. Algorithms for robot LfD seek to enable human users to expand the capabilities of robots through teaching instead of explicit programming. Specifically, LfD methods enable new robot behaviors to be learned based on demonstrations of the desired actions performed by the teacher. Many LfD algorithms are inspired by naturally occurring learning paradigms in humans and other animals, and the broad research goal is to develop an intuitive method of programming that is accessible to untrained, naïve, users.

There are many ways to define a LfD algorithm, which often creates confusion

when one tries to understand where an approach stands within the field. Conveniently enough, one recent survey of the field presents a categorization of existing algorithms in order to help comparative assessments. In this survey Argall et al. [8] categorize then existing algorithms by:

- **Demonstration Technique:** teleoperation [9, 10, 11], shadowing [12, 13], sensors on the teacher [14, 15, 16] or external observation [17, 18]; the categorization is dependent on who performs the task during demonstration (the human or robot) and what information about the demonstrator’s actions is available.
- **Policy Derivation Method:** mapping functions (e.g., classification [9, 19, 20] and regression [10] techniques), system model (e.g., reinforcement learning [21, 22]) and planning [23] approaches.

Previous work has proposed dozens of variants of learning from demonstration algorithms within each category mentioned above. However following three algorithms are of specific interest to us, as we will see them being used later in Chapter 4: Interactive Reinforcement Learning [24], Behavior Networks [12, 25], and Confidence-Based Autonomy [9].

We introduce in detail and evaluate these three algorithms later in Chapter 4 in a comparative user study on a common real world task. We specifically chose these three algorithms because they are good representatives of the three different categories as presented by Argall et al. for (high level) task learning [8]. What we have learned from the implementation, and the naïve users’ feedback on these different approaches influenced the algorithms we present in this thesis. Throughout our literature search and the user study we found that a complementary approach that would bring the positive aspects of multiple algorithms together can potentially improve teaching experience, learning efficiency, and the task performance.

2.2 Reinforcement Learning

Throughout this work we will frequently refer to concepts related to Reinforcement Learning, and that is why here we introduce a formal definition of Reinforcement Learning problems. We also talk about how previous work has used Learning from Demonstration within Reinforcement Learning framework, mainly as a way to convey prior knowledge to an agent.

Reinforcement learning (RL) is a common approach to agent learning from experience. We define reinforcement learning using the standard notation of Markov decision processes (MDPs) [26, 27]. At every time step the agent observes its state $s \in S$ as a vector of k state variables such that $s = \langle x_1, x_2, \dots, x_k \rangle$. The agent selects an action a from the set of available actions A at every time step, transitions into another state s' and subsequently receives a reward r based on a reward function $R(s')$. An MDP's reward function $R : S \times A \mapsto \mathbb{R}$ and (stochastic) transition function $T : S \times A \mapsto S$ fully describe the system's dynamics. The agent will attempt to maximize the long-term reward determined by the (initially unknown) reward and transition functions.

The agent chooses which action to take in a state via a policy, $\pi : S \mapsto A$. Policy $\pi(s)$ is modified by the agent over time to improve performance, which is defined as the expected total reward. Instead of learning π directly, many RL algorithms instead approximate the action-value function, $Q : S \times A \mapsto \mathbb{R}$, which maps state-action pairs to the expected real-valued return of an action chosen at a state. Value functions $V(s)$ quantify how valuable a given state is, and action-value functions $Q(s, a)$ describe how valuable is to take an action, at a given state. Depending on the RL approach, an agent may use either V or Q to assess the potential return of a state, or an action.

In Chapter 5, we introduce (RL) agents that learn using Sarsa [28, 29], a well known but relatively simple temporal difference RL algorithm. Sarsa agents learn to estimate a state-action value function $Q(s, a)$. While some RL algorithms are more sample efficient than Sarsa (i.e. agents can reach to same task performance with less

experience), in Chapter 5 we will focus on Sarsa for the sake of clarity, as it is more likely that a wider audience would be familiar with Sarsa. We record demonstrations as temporal sequences of t state-action pairs $\{(s_0, a_0), \dots, (s_t, a_t)\}$. However, these sequences typically only cover a small subset of all possible states in a domain. Typically an RL agent’s goal is to generalize from the demonstrations and learn a policy $\pi : S \mapsto A$, covering *all* states that imitates the demonstrated behavior. Most similar to our approach in Chapter 5 is the work of Smart and Kaelbling, which shows that human demonstration can be used to bootstrap reinforcement learning in domains with sparse rewards by initializing the $Q(s, a)$ (i.e. action-value function) using the observed states, actions and rewards [30]. In contrast to this approach, our work uses demonstration data to learn generalized rules, which are then used to bias the reinforcement learning process.

For RL agents, policies are as important as value functions since π defines which action to take next. To clarify the connection between a policy and a value function, consider an agent that employs a *greedy policy* which dictates the agent to take *the most valuable action* at a given state s . An agent could use a policy $\operatorname{argmax}_a Q(s, a)$ which would return the action that maximizes $Q(s, a)$ for s . This is just a simple example, and when an agent uses $V(s)$ instead of $Q(s, a)$ learning a more sophisticated policy $\pi(s)$ may be necessary instead of simply using a greedy policy [27]. Previous work proposed different algorithms for using demonstration data to learn π . Deisenroth et al. present a survey on model-free and model-based policy search algorithms with a focus on applications in robotics [31]. Fernandez et al. present Probabilistic Policy Reuse algorithm where an agent can exploit a previously learned policy in a new task. This work is at the intersection of policy learning with transfer learning. For the algorithm we present in Chapter 6, we suggest to use a human reward function recovered from demonstrations via Inverse Reinforcement Learning. We introduce Inverse Reinforcement Learning and Transfer Learning in the following sections of this chapter.

There is one big problem with LfD algorithms that RL can help. Typically, the final performance of LfD algorithms are inherently limited by the quality of the information provided by the human teacher. Most algorithms assume the dataset to contain high quality demonstrations performed by an expert. In reality, teacher demonstrations may be ambiguous, unsuccessful, or suboptimal in certain areas of the state space. A naïvely learned policy will likely perform poorly in such areas [21]. To enable the agent to improve beyond the performance of the teacher, learning from demonstration may be combined with learning from experience, for which RL provides a theoretically sound framework. In a recent work we investigated the use of **reward shaping** with LfD [32]. In RL, for reward shaping an agent sums one or more extra reward function(s) with the environment reward. In environments where the pre-coded reward function is sparse (e.g. $r := 0$ throughout the task and $r := 1$ when the agent reaches the goal), a shaping reward can provide data for a faster value approximation, which in turn can improve learning speed. In our approach, we allowed a demonstrator to make suboptimal demonstrations that lead to successful states. By using a shaping reward, their algorithm uses the demonstrations to speed up learning while dropping the requirement for the demonstrations to be given by an expert.

Just as RL algorithms can improve an LfD technique’s final performance, LfD algorithms possess a number of key strengths that can be beneficial for solving RL problems. Most significantly, demonstration leverages the vast task knowledge of the human teacher to speed up learning either by eliminating exploration entirely [33, 34], or by focusing learning on the most relevant areas of the state space [30]. Demonstration also provides an intuitive programming interface for humans, opening possibilities for policy development to naïve teachers.

In robotics, RL approaches have enjoyed multiple past successes (e.g., inverted Helicopter control [35], and agent locomotion [36]). However they frequently take substantial amounts of data to learn a reasonable control policy. In many domains,

collecting such data may be slow, expensive, or infeasible, motivating the need for ways of making RL algorithms more sample-efficient.

2.3 Inverse Reinforcement Learning

Here we formally introduce the Inverse Reinforcement Learning (IRL) problem, and some commonly known solution techniques. IRL problem is also known as Inverse Optimal Control (IOC), or Inverse Optimal Planning (IOP) problem. We will use the name IRL in this thesis, and refer to IRL later in Chapter 6. Here we give details that motivate the use of IRL in our work.

We saw in the previous section that RL problems consist of learning a value function, or a policy given a reward function. In contrast, IRL research deals with the problem of recovering a reward function (or a cost function) given an optimal policy and a state transition model (i.e. environment model). However, there are IRL algorithms that do *not* require the environment model to recover a reward function and we will refer to these as model-free IRL algorithms.

Even though the history of IOC goes back to Kalman’s question “When is a linear control system optimal?” [37], the number of publications show that in robotics and agents research the topic became more popular in early 2000’s. In an early paper, Ng and Russell base the motivation of IRL approaches to natural phenomena, stating that a reward function for humans’ and animals’ behavior control is often unknown [38]. For instance, for a bee that flies off to collect nectar, it is hard to know how the bee weights the relative benefit of flight distance against flight time, and risk from wind and predators in advance. Another similar example is human economic behavior. Therefore, it makes sense to try learn these relative weights based on observation of world states and executed actions. In his lecture notes Abbeel adds to this motivation a presupposition that a reward function also provides the most compact and transferable definition of a given task [39].

The general formal definition of an IRL problem is as follows: For a reward function that consists of linear combinations of a set of reward features $R(s) = \mathbf{w}^\top \phi(s)$, $\phi(s)$ is the vector of reward features that we compute based on the state vector, and \mathbf{w} is a vector feature weights. An IRL algorithm aims to find a set of optimal feature weights \mathbf{w}^* such that for all policies: $\mathbf{w}^{*\top} \mu(\pi^*) \geq \mathbf{w}^{*\top} \mu(\pi)$ (note that $\mu(\pi)$ symbolizes the expected cumulative sum of reward feature values (i.e. feature expectations) given a policy π . Also, π^* is the optimal policy, whereas π is a suboptimal policy).

Some of the challenges for finding solutions to IRL problems are: the ambiguity of the solution (that is, more than one reward function may satisfy the constraints), the assumption that the expert demonstrations are optimal, and the assumption that we can enumerate all policies in order to compare one against another when finding an optimal set of reward features.

Abbeel divides existing IRL solution approaches into three main categories [39]:

- **Maximum Margin approach:** Ratliff et al. introduce a margin function $m(\pi^*, \pi)$ that returns an evaluation of disagreement between a given policy and the optimal policy, and aims to find a set of reward feature weights so that for all policies: $\mathbf{w}^\top \mu(\pi^*) \geq \mathbf{w}^\top \mu(\pi) + m(\pi^*, \pi)$ [40, 41, 42].
- **Feature Expectation Matching:** Builds on the observation that for a policy π to be guaranteed to perform as well as the optimal policy π^* , it is sufficient that the feature expectations match within a certain error threshold ϵ : $|\mathbf{w}^{*\top} \mu(\pi) - \mathbf{w}^{*\top} \mu(\pi^*)| \leq \epsilon$ [22, 43, 44, 45].
- **Reward Function as a Parameter:** Assumes that an expert behaves according to a probabilistic policy $\pi(a|s, R, \alpha)$, that is the expert's policy is a function of the reward function. This formulation allows the use of Bayesian inference and evaluation of likelihood of a trajectory for a given R , and α . Previous work then used various methods to find a reward function by sampling from this distribution and optimizing the likelihood of the policy [46, 47, 48].

Example applications of the IRL algorithms implemented in the work we cite above are simulated highway driving, aerial imagery based navigation, parking lot navigation, destination prediction for urban navigation, human path planning, human goal inference, and quadruped robot locomotion.

All of the solutions we cite above are model based, where a transition model of the world – either deterministic, or probabilistic – is assumed to be known. Although outnumbered by model-based approaches, there are model-free IRL solutions that do *not* require an environment model. Klein et al. [49] approach the IRL problem as a supervised learning problem, where they suggest a two-layer approach. First layer consists of training a classifier to choose a *score function* $q(s, a)$ that evaluates the association of a state with an action, and the second layer learns a regressor to predict a reward $\hat{r} = q(s, a) - \gamma q(s', a')$, i.e. the difference between the past state-action score and the current state-action score.

Boularias et al. also introduce Relative Entropy Inverse Reinforcement Learning (RE-IRL), a model-free IRL algorithm [5]. RE-IRL aims to minimize the KL - divergence between two probability distributions: a conditional distribution $P(\tau|\mathbf{w})$, and a user-defined distribution $Q(\tau)$, where $\tau = \langle (s_0, a_0), \dots, (s_H, a_H) \rangle$ is a demonstrated trajectory of length H , and \mathbf{w} is the vector of reward feature weights. The distribution Q can be defined so that samples from a particular part of the trajectory space are weighted with more importance than others. If we want to recover the most informative reward feature weights then we can choose $Q(\tau)$ to be a uniform distribution across the trajectory space. Boularias et al. [5] define the RE-IRL problem as:

$$\begin{aligned} \max_{\mathbf{w}} g(\mathbf{w}) &= \mathbf{w}^\top \mathbf{f}^{\pi_E} - \ln Z(\mathbf{w}) - \lambda \|\mathbf{w}\|_1 \\ Z(\mathbf{w}) &= \sum_{\tau} Q(\tau) \exp(\mathbf{w}^\top \mathbf{f}_i^\tau) \\ P(\tau|\mathbf{w}) &= \frac{1}{Z(\mathbf{w})} Q(\tau) \exp(\mathbf{w}^\top \mathbf{f}_i^\tau) \end{aligned}$$

For finding the vector of reward feature weights that maximize the objective function $g(\mathbf{w})$, the algorithm empirically estimates the subgradient of the objective function with importance sampling of trajectories and executes gradient ascent.

Boularias et al. evaluate the algorithm on a simulated robot for ball-in-a-cup task, based on human expert demonstrations where a human tries to catch a ball attached to a string hanging off the bottom of a cup [5]. Muelling et al. used RE-IRL for inferring strategies for table tennis based on human gameplay tracking data [50]. They show that the estimated reward function was able to capture strategic information enough to distinguish the expert among players with different skill levels and styles.

In Chapter 6 we use RE-IRL algorithm as an off-the-shelf solution for recovering a reward function from demonstration data. We do not intend to improve RE-IRL nor do we plan to go in the direction of IRL research. The requirement of the algorithms we present is to employ a simple to implement, yet usable, model-free IRL algorithm that can recover a function that would help a learner assess how desirable is a change in the world state. Using relative weights of reward features, we execute gradient ascent in order to reach a goal state on a path that prioritizes the relative importance of the features along the path (e.g. approach in x direction can be more desirable than approach in y direction).

2.4 Transfer Learning

Transfer Learning (TL) allows an agent to begin learning with an informative prior instead of relying on random exploration [51]. The intuition behind TL is that generalization may occur not only within tasks, but also *across tasks*.

In Chapter 5 we introduce Human-Agent Transfer HAT algorithm. For HAT, we use *Rule Transfer* [52], a particularly appropriate transfer method that is agnostic to the knowledge representation of the source learner. The ability to transfer knowledge between agents that have different state representations and/or actions is a critical ability

when considering transfer of knowledge between a human and an agent.

The following steps summarize Rule Transfer:

- 1a: Learn a policy** ($\pi : S \mapsto A$) **in the source task.** Any type of reinforcement learning algorithm may be used.
- 1b: Generate samples from the learned policy** After training has finished, or during the final training episodes, the agent records some number of interactions with the environment in the form of (S, A) pairs while following the learned policy.
- 2: Learn a decision list** ($D_s : S \mapsto A$) **that summarizes the source policy.** After the data is collected, a propositional rule learner is used to summarize the collected data to approximate the learned policy by mapping states to actions.¹ This decision list is used as a type of inter-lingua, allowing the following step to be independent of the type of policy learned (step 1a).
- 3: Use D_t to bootstrap learning of an improved policy in the target task.** For instance, previous work [52] provided three ways of leveraging this knowledge; we discuss two of these methods later in Chapter 5.

Transfer learning methods for reinforcement learning can transfer a variety of information between agents. However, many transfer methods restrict what type of learning algorithm is used by both agents (for instance, some methods require temporal difference learning [53] or a particular function approximator [54] to be used in both agents). However, when transferring from a human, it is impossible to copy a human’s “value function” — both because the human would likely be incapable of providing a complete and consistent value function, and because the human would quickly grow wary of evaluating a large number of state-action pairs.

¹Additionally, if the agents in the source and target task use different state representations or have different available actions, the decision list can be translated via inter-task mappings [52, 53] (as step 2b). For our work, this translation is not necessary, as the source and target agents operate in the same task.

In a recent work by Knox and Stone [55], their TAMER [56] system learns to predict and maximize a reward that is interactively provided by a human. The learned human reward is combined in various ways with Sarsa(λ), providing significant improvements. The primary difference between HAT and this method is that we focus on leveraging human demonstration, rather than estimating and integrating a human reinforcement signal.

The idea of transfer between a human and an agent is somewhat similar to *implicit imitation* [57], in that one agent teaches another how to act in a task, but HAT does not require the agents to have the same (or very similar) representations.

Allowing for such shifts in representation gives additional flexibility to an agent designer; past experience may be transferred rather than discarded if a new representation is desired. Representation transfer is similar in spirit to HAT in that both the teacher and the learner function in the same task, but very different techniques are used since the human’s “value function” cannot be directly examined.

High-level advice and suggestions have also been used to bias agent learning. Such advice can provide a powerful learning tool that speeds up learning by biasing the behavior of an agent and reducing the policy search space. However, existing methods typically require either a significant user sophistication (e.g., the human must use a specific programming language to provide advice [58]) or significant effort is needed to design a human interface (e.g., the learning agent must have natural language processing abilities [59]). Allowing a teacher to demonstrate behaviors is preferable in domains where demonstrating a policy is a more natural interaction than providing such high-level advice.

2.5 Reward Shaping

Reward shaping, derived from behavioral psychology [60], is a way of incorporating prior knowledge in the learning process. A shaping reward F provides an agent with

additional information after each state transition. With shaping rewards, an agent is rewarded for taking appropriate steps toward the goal. This extra reward supplements the environment reward which is typically defined as a sparse function for practical reasons.

The extra reward signal F , defined by agent designers, is added to the environment reward R , defined by problem designers. The agent learns a task using the shaped reward R' :

$$R'(s, a, s') = R(s, a, s') + F(s, a, s') \quad (2.1)$$

F function usually encodes heuristic knowledge, and is intended to complement the typically less informative signal R . As we mentioned earlier, the agent’s goal is defined by the reward function, and by manipulating the reward signal we may risk changing the task. Ng et al. [61] proved that potential-based shaping is a sound way to provide a shaping reward without changing the reinforcement learning problem.

2.6 Reinforcement Learning from Demonstration

RL from Demonstration (RLfD) is a learning setting for RL agents where both demonstrations and an environment reward signal are available. Algorithms use different methods to convert demonstrations into a representation that the agent will be able to use as heuristic information and the environment reward is treated as the ground truth for learning. Early work on Reinforcement Learning from Demonstration (RLfD) was presented by Schaal [62], in which the term described a set of approaches that leveraged initial training for learning a policy and a value function from demonstrations, followed by a “trial by trial learning” with RL.

Most recently, Brys et al. introduced a new algorithm we refer to as Similarity Based Shaping (SBS) [32]. With the SBS algorithm Brys et al. took a different approach to investigate the use of prior knowledge as a potential-based shaping reward.

The SBS algorithm uses human demonstrations as a potential function $\Phi^D(s, a)$ for potential-based shaping. With this approach each demonstrated state-action pair is assumed to be desirable for the agent, and using a similarity metric, the SBS algorithm computes a shaping reward F . During learning, the agent receives a shaped reward signal $R' = R + F$ which is the combination of the environment reward (i.e. ground truth) and the shaping reward. The SBS algorithm uses non-normalized multi-variate Gaussian distributions to compute the similarity between demonstrated states and observed states throughout learning.

Chapter 3

Effect of Human Guidance and State Space Size on Interactive Reinforcement Learning

In this chapter, we study how Reinforcement Learning can be made more efficient for real-world robotic systems by enabling human teachers to provide feedback at runtime. Our feedback differs from the related work in the sense that the teacher can provide not only reward for the preceding action but also guidance for the subsequent action. Our work builds upon a series of studies by Thomaz et al. that show that in a reinforcement learning scenario users tend to not only reward past actions but also provide anticipatory rewards to guide the learner in future actions [63]. Based on this analysis, Thomaz and Breazeal present *Interactive Reinforcement Learning* (Interactive RL), an algorithm that enables a human trainer to 1) provide positive and negative rewards in real time in response to robot actions, and 2) provide anticipatory guidance input that restricts action selection choice and guides the learner towards performing the desired behavior [24]. The authors show that incorporating user guidance into the policy learning process significantly improves the learning rate of a software agent by reducing the number of states it explores.

We report the results of a case study in which Interactive Reinforcement Learning was applied to teaching an Aldebaran Nao robot. To our knowledge, this work is the first to apply this type of learning algorithm to a real-world robotic system. To evaluate the approach, we closely reproduce the interactive reward interface proposed in [24] and perform four experiments comparing Interactive RL with and without anticipatory guidance for two different state space sizes. We discuss modifications made to apply this learning approach to a real-world system. Our results show that guidance significantly reduces the learning rate, and that its positive effects increase with state space size. We conclude with a discussion of the benefits and challenges of using Interactive Reinforcement Learning in real-world robotic applications.

3.1 Overview

The ability to acquire and customize robot behavior through learning is vital for the successful deployment of robots in a broad range of consumer and service applications. In order for robots to become effective assistants in diverse human environments we must develop the capability to quickly and efficiently customize robot behavior and learn new tasks. Numerous works have addressed the challenges of learning in complex real-world environments [64, 65]. One popular technique is Reinforcement Learning (RL) [66], which enables a robot to learn from rewards it obtains while attempting to perform its task. Although RL algorithms have had great success in offline learning and software applications, the large amount of data and high exploration times that such algorithms require make them intractable for many real-world robotic domains.

A number of recent work has shown that in the presence of a human, robotic systems can be designed to take advantage of human input [67, 9]. In the case of Reinforcement Learning, one example is the work of Kaplan et al. where they present a technique in which user-generated reward and punishment signals are used to train the robot new tasks in a form of clicker training [68]. Another related work was presented

by Smart and Kaelbling, where human demonstrations of robot actions can be used to make learning of otherwise intractable policies possible [11]. Also, Knox and Stone introduce the TAMER framework that allows a human trainer to interactively shape an agents policy via reinforcement signals [69].

The motivation behind our work is similar to these past examples.

3.2 Description of Interactive RL Algorithm

The Interactive Reinforcement Learning algorithm with human guidance, as described in [24], modifies traditional Q-learning by integrating human input into the reward and action selection mechanisms. In Interactive RL, all *reward input* to the robot comes from the human teacher instead of the environment; the teacher is given a short period of time following each action to provide positive or negative feedback. Additionally, during a short delay period between actions, the user can provide *guidance input*. Guidance is given with respect to an object or location (e.g., table), with the effect of restricting the choice of immediately available actions to those related to the target object (e.g., look at table, put down object on table). This technique can be thought of as a means of directing the robot’s attention to the target object or area. In the following sections, we first present the details of the Interactive RL algorithm and then describe the interaction interface used by the human teacher.

Algorithm 1 presents the pseudo-code for Interactive RL. The algorithm extends traditional Q-learning with the addition of human reward input (lines 18-21) and human guidance (lines 6-8). When learning begins, the agent has no knowledge of the task and the Q-table contains only the robot’s current state. The table expands (line 17) as the robot performs actions, receives rewards and encounters new states, and the Q-values are updated (lines 23-24). In our implementation, we set the learning rate $\alpha = 0.3$ and the discount factor $\gamma = 0.75$. We initialize the Q-values for every new state that the agent discovers to 0.5.

Our initial implementation of Interactive RL followed the exact description provided in [24]. Following preliminary testing, we implemented a change to the action selection mechanism. Specifically, in the original implementation, in the absence of teacher guidance the agent uses the $Q[s, a]$ values to weight available actions and probabilistically select the next action to perform. During preliminary experiments, we found that this approach resulted in very long learning times for the robot (over 80 minutes for the small domain described in Section 3.3.2). In followup experiments, we found that ε -greedy action selection (lines 1-2, 12-15) results in significantly improved learning times. In this selection method, the robot performs the optimal action with probability $1 - \varepsilon$ and a random action with probability ε . Over time, the value of ε decays (line 25) until eventually the robot always performs the optimal action (i.e., policy exploitation).

The ε -greedy action selection method is used to select the next action both when a guidance signal is present and when it is absent. If none of the actions specified by the guidance message are valid (not physically possible), the robot ignores the guidance message and uses ε -greedy selection to pick among all available actions. For example, in the domain described in Section 3.3.1, the *PickUp* action is not valid before the robot determines whether there is an object in front of it using the *TakePicture* action.

A key contribution of the original Interactive RL paper is the interactive reward interface that enables an online user to train a virtual robot to bake a cake in a domain called Sophie’s Kitchen¹. The human trainer uses the left mouse button to provide reward signals $r = [-1, 1]$ for the agent’s actions, and the right mouse button to provide guidance for future actions. In this work, we recreate the interactive reward interface for a real robot by providing the human trainer with a fixed view of the environment via a web-cam (Figure 3.2a). As with the algorithm, we began by implementing the interface described in [24] in order to evaluate its use in real-world systems. Following

¹The original Sophie’s Kitchen web application is available at <http://www.cc.gatech.edu/~athomaz/sophie>

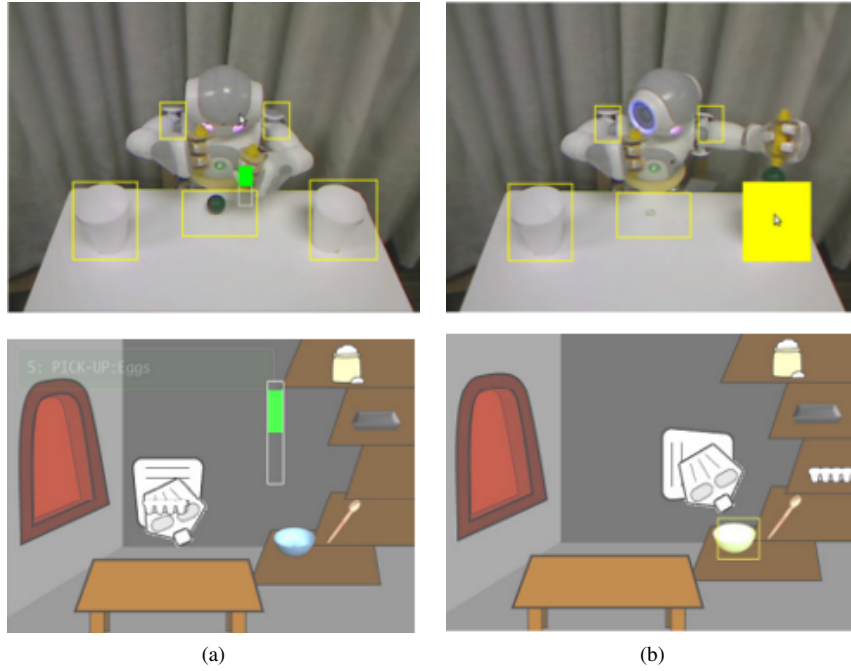


Figure 3.1: Screen-shots of our interface (*Top row*) and Sophie’s Kitchen (*Bottom row*). (a) shows positive reward given with a left click and drag upwards; (b) shows guidance given with a right click on the object that the teacher wants the agent to interact with. preliminary evaluations we found the need to make one modification to the interface in order to improve the ease of interaction for our domain.

In our evaluation, the reward interface functions as in the original web application, in which left clicking anywhere in the image brings up a reward bar (Figure 3.1a) that the user can fill with a positive or negative reward value representing the desirability of the current state. The guidance interface differs slightly; instead of allowing the user to click anywhere, our interface highlights five rectangular regions as potential guidance targets. The teacher provides guidance by right-clicking within the highlighted boxes (Figure 3.1b). Table 3.1 lists the five regions and their effect on action selection.

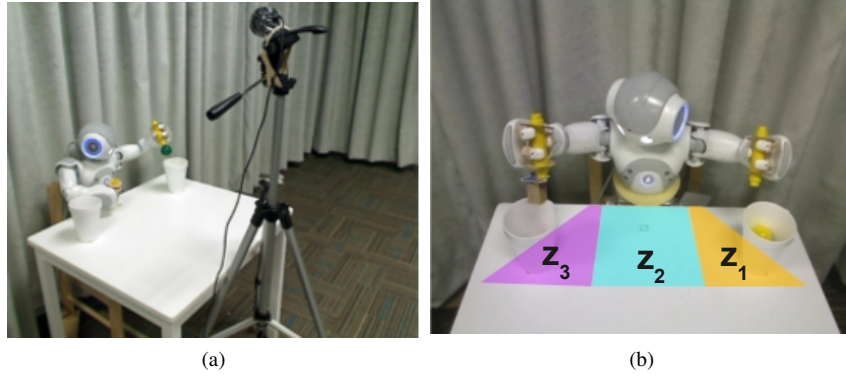


Figure 3.2: Experiment Setup. (a) Web camera captures the table and the robot for the user interface. (b) The table is divided in three zones. Orange, cyan blue and purple show z_1 , z_2 , and z_3 respectively. The robot has an object at the tip of its right hand and is about to drop it in the right cup.

3.3 Experimental Setup

To investigate the ways in which human guidance impacts the policy learning rate in real-world systems we developed an object sorting domain using an Aldebaran Nao humanoid robot. The domain setup consists of a stationary work table where the Nao is seated on a chair. Magnetic objects are placed in front of the robot one at a time. The robot must learn to use its camera at the appropriate time to identify the characteristics of the object and then to pick up and place the object in one of two cups located on the table. The user observes the robot’s actions via a web-cam (see Figure 3.2a) and provides feedback to the robot through the graphical user interface. The described setup enables us to evaluate the impact that human guidance input has on robot learning performance and the usability of the user interface.

3.3.1 Domain

The experimental domain $D = (Z, S, A, T)$ is defined by a finite set of zones Z , a finite number of world states S , a finite set of possible robot actions A and the transition function $T : S \times A \rightarrow S$ that determines transitions between states by way of actions.

Zones Z represent the regions where the robot’s arms or an object can be located. In our domain, $Z = \{z_1, z_2, z_3\}$ corresponds to the left, front and right side of the robot respectively (Fig. 3.2b).

The state of the world $S = \{S_r \cup S_o\}$ consists of the union of the state of the robot S_r and state of the object being classified S_o . The robot state is defined by $S_r = (z_{lh}, z_{rh}, hand_o)$, where $z_{lh}, z_{rh} \in Z$ correspond to the current zone of the left and right hands, respectively, and $hand_o \in \{left, right, none\}$ represents whether an object is located in either of the robot’s hands, or not. For example, in Fig. 3.2b the robot’s state is represented by $S_r = (z_1, z_3, right)$.

The object state is defined by $S_o = (I, o_z, o_p)$, where I is an object descriptor that specifies the characteristics of the object using a set of features described in Section 3.3.2. $o_z \in Z$ is the current zone of the object, and $o_p \in P$ is the placement of the object such that $P = \{z_1, z_2, z_3, right_cup, left_cup\}$. For example, in Fig. 3.2b the object state is represented by $S_o = (pattern, z_3, z_3)$.

The robot has eleven possible actions, $TakePicture, xPickUp, xPutDown, xDrop, xLeft$, and $xRight$ where $x \in \{L, R\}$ determines the arm with which the action is performed. The $TakePicture$ action makes the robot take a snapshot of the table in front of it and extract features from the image to determine the characteristics of the object currently placed there (if any) (see Fig. 3.3). $xRight$ and $xLeft$ actions move the hands between zones (e.g. $LLeft$ moves left hand left, that is, to Zone 1 and $RLeft$ moves right hand left, that is, Zone 2. Note that the robot’s left hand can operate only in $z_{1,2}$, and the right hand can operate only in $z_{2,3}$, which enables the robot to pick up an object with either hand but allows only one hand to reach each cup and $Drop$ the object into it. If the robot initially picks up the object with the wrong hand, it can switch hands by performing a $xPutDown$ action followed by a $xPickUp$.

Table 3.1 shows how different guidance messages restrict the set of available robot actions. Note that the degree to which guidance reduces the set of actions has a signif-

Guidance	ActionSet
Zone 1	<i>LLeft, LDrop</i>
Zone 2	<i>LPutDown, LPickUp, RPutDown, RPickUp, TakePicture</i>
Zone 3	<i>RRight, RDrop</i>
Left Shoulder	<i>LPickUp, LRight, LLeft, LPutDown, LDrop</i>
Right Shoulder	<i>RPickUp, RRight, RLeft, RPutDown, RDrop</i>

Table 3.1: Guidance messages restrict action selection to the set of available actions as shown. The first letter *L* and *R* denotes either Left or Right hand depending on the hand with which the action is executed.

icant effect on the learning rate. If guidance does not significantly reduce the choice of available actions, then the effect of this input method is reduced since the robot will have to choose randomly among them. In contrast, if the guidance message limits the robot to a single action, then all choice is taken away from the robot; this approach would be equivalent to the human demonstration techniques applied in *learning from demonstration* approaches [70, 11]. The effect on learning would be to eliminate exploration, resulting in a policy that is only as good as the decisions demonstrated by the human, and is poorly defined in areas of the state space not encountered during training. Further studies are needed to determine the precise impact of these choices. In our implementation we chose to design guidance as an effective teaching method that always restricts the list of available actions but never reduces action selection to a single choice.

3.3.2 Experimental Conditions

We performed four sets of experiments to evaluate the Interactive RL algorithm and study the effects that 1) teacher guidance, and 2) size of the state space have on learning performance. The same object sorting task was used for all experimental conditions. In the no-guidance condition, the user was restricted to only the reward input, and the

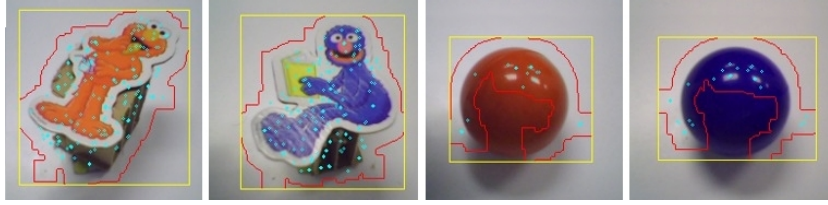


Figure 3.3: Images of different objects taken by robot’s camera. Light blue spots show SURF, red lines show the outline, and yellow rectangles show the bounding box or rectangle of interest of the object found.

guidance condition allowed for both types of input. To vary the state space size we changed the number of features used to describe the object being sorted. Specifically we used the following two experimental conditions:

- **Small deterministic state space:** In the small state space condition, the object descriptor I of $S_o = (I, o_z, o_p)$ consists of a single variable with two possible values, *plain* or *pattern*, representing the color class of object. This mapping is determined based on the number of Speeded Up Robust Features (SURF) [71] identified in the image of the object and identifies the object as either a solid colored ball or a character magnet, respectively. Specifically, objects for which over 50 SURF features were identified were characterized as *pattern*, otherwise the object was *plain*. In our tests we found that the number of SURF features deterministically separate these two object types. Combining this binary object representation with other state information, such as object location and robot state, results in 360 possible states.
- **Large, non-deterministic state space:** In the large state space condition, the object descriptor I consists of four elements, where measured values are rounded to the closest defined category: the number of SURF features (50, 100, or 150), smoothness (0.05 or 0.1), entropy (5.0 or 10.0) and area of the bounding box of the object (15,000, 20,000 or 25,000). We calculate smoothness and entropy as described in [72]. After image processing, the value for each descriptive fea-

ture is thresholded into the categories listed above. This representation was purposefully designed to reduce the size of the state space by providing a small set of possible values for each variable, while at the same time making sure that none of the descriptive elements alone is sufficient for distinguishing between the plain and patterned object types. For example, although they are the same type of object, the plain green and plain yellow magnets have different smoothness, perceived area and entropy states. To the robot these objects therefore appear distinct and it must learn that they belong to the same group. Furthermore, this representation is non-deterministic and variations in object placement will result in slightly different state representations, allowing us to evaluate how Interactive RL performs under these conditions. In total, the large state space representation results in 6480 states.

During the experiments all processing was performed on a PC connected to the robot over the network. For image processing we used OpenCV [73], examples of the processed images are shown in Figure 3.3. All experiments were performed by author of this thesis. The teacher used the following consistent guidance and reward shaping protocol. For the first 20 actions of the experiments, all actions were given reward and guidance (in case of experiments with guidance). After 20 actions, reward and guidance were given only after the incorrect action that the robot performed. After each incorrect action, the following 5 actions were rewarded and guided. If the action was correct, no reward or guidance was given until the next incorrect action. In the small state space condition we used $\varepsilon = 0.1$ and $\delta_\varepsilon = 0.002$ (ε reaches 0 after 50 time steps), and in the large state space condition $\varepsilon = 0.1$ and $\delta_\varepsilon = 0.001$ ($\varepsilon = 0.1$ reaches 0 after 100 time steps). We terminate learning once the robot is able to correctly sort three objects into each cup without guidance or reward. Objects were presented to the robot in random order. A video of the experiment and the interface can be found

online².

3.4 Results

We ran three trials for each of the four experimental conditions, the averaged results are presented in Figure 3.4. Graphs plot the level of teacher involvement over time in terms of the percentage of actions for which the teacher provided reward and/or guidance. A summary of experimental statistics is given in Table 3.2.

A comparison of plots Fig. 3.4a to 3.4c and Fig. 3.4b to 3.4d, shows that the use of guidance remarkably reduces the learning time (small state space: 28%, large state space: 44%) as well as the number of states explored (small state space: 51.5%, large state space: 38%). In effect, the robot learns more quickly with guidance because it selects the correct action more frequently and therefore avoids exploring irrelevant areas of the state space. As a result, we observe that in the presence of guidance the teacher reduces the number of interactions earlier in the training process compared to the no-guidance condition.

We also observe differences in teacher behavior between the guidance and no-guidance conditions. The cumulative reward for the trials is positive in the presence of guidance and negative in its absence. This is a direct consequence of the fact that the robot selects more inappropriate actions when learning without guidance and the teacher spends more time giving praise to the robot than punishment. This is an interesting result given that a number of studies show that given the option between reward and punishment most human teachers prefer to use positive reward and avoid punishment when training an agent [63].

In summary, the comparison study shows that teacher guidance significantly reduces learning time by reducing the number of states explored. This result supports the findings of [63], in which average reductions of 30% in case of expert teachers and 49%

²<http://www.youtube.com/WPIrail>

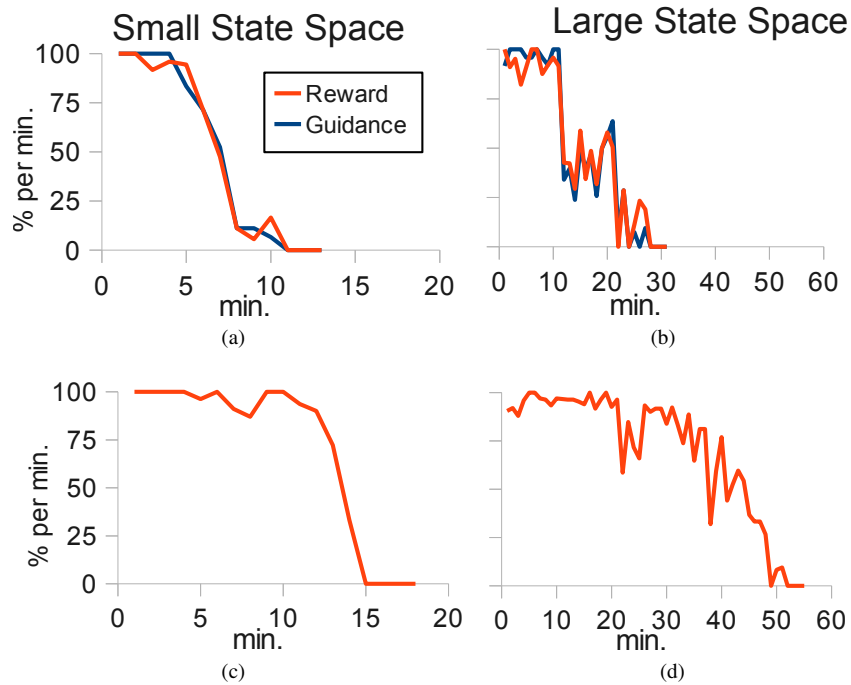


Figure 3.4: User interaction and training time with (a,b) and without (c,d) guidance input. Figures (a,c) and (b,d) show the results obtained in small and large state space respectively. The y-axis and legend are common for all graphs.

in case of non-expert teachers were observed in the number of trials. Furthermore, we find that the impact of teacher guidance increases as the size of the state space grows. We hypothesize that similar benefits would be observed with respect to growth in the size of the action space, although further studies are required to verify this effect.

3.5 Discussion

The Interactive Reinforcement Learning approach has several key strengths: 1) its simple action selection mechanism can be integrated into a broad range of existing RL approaches, 2) its application results in a flexible learning system that can learn independently but is able to take advantage of human guidance when it is available, and 3) guidance is combined with independent exploration in a way that enables the robot

	Fig.3.4a	Fig.3.4b	Fig.3.4c	Fig.3.4d
#States	11.33	42.67	22	68.67
#Actions	82	201	134.67	366.67
Cumul. Rew.	16.67	58	-44.67	-74.67
Total #G.	44	113.67	0	0
Training Time [min.]	14	31	18	55

Table 3.2: Experiment statistics. All results are shown as the average of three trials for each experiment. Number of states discovered (#States), total number of actions performed (#Actions), sum of the positive and negative rewards given (Cumul. Rew.), total number of guidance given (Total #G.) and training time.

to not only imitate the demonstrated behavior but also to surpass the performance of the teacher if necessary.

However, there are a number of challenges to deploying this approach in real-world applications. The current implementation of the teaching interface assumes a fixed-camera setup that enables the human user to select relevant areas of the state space for guidance. While this interface may translate into some service and factory applications, it is not suitable for mobile robotic systems or unconstrained environments. Further work is required to evaluate how other input methods, such as portable tablets, speech, and gestures, can be used to interact with the robot in this setting.

The running time of the algorithm is another area for potential improvement. Currently, the robot requires approximately half an hour to learn a relatively simple task. Much of this learning time is dependent on the time required to execute each action. For example, an average of 201 actions was required to learn the large task, which is a relatively small number compared to the total number of states in the domain, but each action had an average time of 9.8 seconds. Other critical factors include the exploration rate, state representation, and the degree to which guidance reduces the choice of available actions. In order to enable this approach to scale to more complex domains, we must explore the effects that these factors have on learning time, as well as study how Interactive RL can be applied to RL algorithms that utilize continuous state space representations which model complex world state more effectively [74, 75]. An inter-

esting future work is a comparison of the Interactive RL algorithm with the TAMER framework [56] and the SABL algorithm [76]. The TAMER framework learns a human reward function however does not contain any guidance input. The SABL algorithm is able to interpret the lack of feedback which is different from both the Interactive RL and the TAMER algorithms. A fair comparison of these three RL algorithms that learn tasks from humans can be informative for discovering future directions for developing more capable and efficient algorithms.

We compared two conditions, the large state space, in which the generalization across multiple sensory inputs had to be learned, and the small state space, in which a single sensory input automatically generalized across the two object types. Both conditions were designed through manual selection of the appropriate state features in a preliminary testing stage. In the case of the small state space, the chosen representation enabled us to teach the task more effectively because instructing the robot through the use of just one to two patterned magnets enabled it to automatically generalize the behavior to all patterned shapes. In the case of the large domain, each patterned item was described by a unique set of features, and more examples were required for the algorithm to learn the appropriate generalization. The large state space more closely represents a real-world scenario; robots designed for operation in real-world environments in the presence of non-expert users are likely to have an even larger number of sensory inputs, possibly with many irrelevant features, which will lead to longer learning times. This motivates the need for semi-autonomous feature selection methods that reduce the size of the state space and enable the robot to learn more effectively.

Finally, when designing interactive learning algorithms we must consider their usability for non-expert users. This chapter focused on empirical evaluation of the best-case scenario in which the robot was taught by a roboticist after numerous practice trials.

Algorithm 1 Interactive Q-Learning with human generated reward and guidance

```
1:  $\varepsilon$  = initial value
2:  $\delta_\varepsilon = \varepsilon /$  number of desired time steps to stop exploration
3: Initialize Q-table with starting state
4: while learning do
5:   while waiting for guidance do
6:     if receive human guidance then
7:        $g = \text{guidanceTarget}$ 
8:     end if
9:   end while
10:  if received guidance then
11:     $A =$  set of legal actions that only contain  $g$ 
12:  else
13:     $A =$  set of legal actions for the current state
14:  end if
15:   $rand =$  random number  $[0,1]$ 
16:  if  $rand > \varepsilon$  then
17:     $a = \arg \max_a Q[s, a]$ 
18:  else
19:     $a =$  randomly selected action in  $A$ 
20:  end if
21:  execute  $a$  and transition to  $s'$ 
22:  if  $s'$  not in Q then
23:    add  $s'$  to Q-table
24:  end if
25:  while waiting for reward do
26:    if receive human reward then
27:       $r =$  human given reward
28:    else
29:       $r = 0$ 
30:    end if
31:  end while
32:  update values:
      
$$Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma(\max_{a'} Q[s', a']) - Q[s, a])$$

33:  if  $\varepsilon > 0$  then
34:     $\varepsilon \leftarrow \varepsilon - \delta_\varepsilon$ 
35:  end if
36: end while
```

Chapter 4

A Comparison of Three Learning from Demonstration Algorithms

In the previous chapter we studied how Reinforcement Learning can be made more efficient for real-world robotic systems, however our study was evaluated using an expert teacher’s demonstrators. In this chapter, we present findings from a user study in which we asked non-expert users to use and evaluate three different robot Learning from Demonstration algorithms. The three algorithms we chose – Behavior Networks, Interactive Reinforcement Learning, and Confidence Based Autonomy – utilize distinctly different policy learning and demonstration approaches. This enables us to examine a broad spectrum of the field.

4.1 Overview

Research on robot Learning from Demonstration has seen significant growth in recent years, but the field has had only limited evaluation of existing algorithms with respect to algorithm usability by naïve users. Despite the existence of a diverse body of work, research in Learning from Demonstration lacks a comparative user study that focuses

on the performance and preferences of naïve users on a common domain. The lack of comparison leaves many critical questions unanswered. Without the comparison of existing methods, future researchers have no guidance with respect to which algorithms and interaction techniques are most effective, and what underlying algorithmic assumptions accurately reflect the real-world use case. These problems are further heightened by the fact that many LfD techniques have only been evaluated through use of expert roboticists, typically the authors themselves [77, 9, 78, 11, 10]. As a result, the correctness of the underlying assumptions with respect to the type and quality of demonstrations, and the ease of use of interaction methods have not been evaluated. For example, human teachers often make noisy demonstrations to robots [9]. Since as computer scientists and roboticists we know the inhibiting effects such noise can have on learning algorithms, we may take great care in providing reasonably clear training data. This certainly cannot be expected of a naïve teacher. If naïve teachers expect certain types of interactions and responses from the robot, then we as engineers and scientists must address these issues in our own work. It is only then that the success of LfD can progress in the population of non-expert users.

Previous studies typically focus on evaluating policy performance [79, 80], and often compare learning time against non-interactive methods, such as Reinforcement Learning [27, 11, 81]. However, we are not aware of any user studies that have examined the usability of LfD methods on a common test domain with naïve test subjects. We note that this fact is not due to the lack of academic rigor in LfD research, but instead highlights some of the challenges of this research problem. The use of different robotic platforms, interfaces, and demonstration techniques leads to different representations and characteristics of demonstration data that makes comparisons difficult across applications. The use of standard data sets, which is common in other research fields, is not possible due to the human interaction component and the diversity of interaction techniques. As a result, comparison requires full reimplementations of algo-

gorithms a highly time consuming and challenging effort due to the absence of existing open source solutions.

4.2 Description of Selected LfD Algorithms

To further investigate the issues raised above, we present a user study of three established LfD algorithms, each representing a different variant of policy learning:

- **Interactive Reinforcement Learning [24]:** A technique inspired by reinforcement learning, where the reward signal is provided by the human user at runtime instead of a pre-coded environment reward. Interactive Reinforcement Learning is designed in light of successive user studies and works based on positive or negative feedback for the preceding action and guidance for the following action of an agent. The teacher interacts with the robot by providing reward and guidance input – in our case through an on-screen interface. With reward feedback, the teacher can tell the agent whether the preceding action was the right or the wrong thing to do. With guidance, the teacher has control over narrowing down the action choices of the agent (i.e., limits the action space when choosing an action); however, the user cannot give a specific action command for the agent to execute.
- **Behavior Networks [12, 25]:** A planning-based policy learning technique. Behavior Networks is a planning algorithm which is meant to be used in conjunction with teleoperation. In our implementation the teacher interacts with the robot through kinesthetic teaching [16], a type of experienced demonstration [78] in which the user physically guides the robot through the task. This algorithm divides a task into behaviors with pre and post conditions where each behavior can be any low level control or high level action. A behavior gets activated depending on the preconditions defined for that specific behavior. The definition of each

behavior depends on several factors such as the task, environmental conditions, implementation preferences, hardware limitations etc. Later in Chapter 4 we will see how this flexibility (or loose definitions) will reflect on the test subjects' experience.

- **Confidence-Based Autonomy [9]:** An approximation technique based on a classification algorithm. Confidence Based Autonomy is an active learning method where a learner asks for help from a human teacher when the classification confidence of an action is lower than a certain threshold. For each state of the world, the teacher interacts with the robot either by selecting (labeling) the next action or correcting a past action choice through an on-screen interface. It is different from previous algorithms in that: i) the agent initiates the interaction with the teacher, and ii) the algorithm requires to list all available action choices to the user which, allows for robot teleoperation.

When selecting the algorithms above, we were motivated by several factors: the difference in task representation (e.g. state-action mapping versus sequential action planning), the difference in the learning process (e.g. optional feedback versus active learning) and general acceptance in the LfD community (each of the chosen algorithms has over 100 citations in the literature). We believe that it is reasonable and fair to compare these methods on a common ground; the methods selected have elements that can possibly be attractive for different reasons. They can be thought as complementary elements or contradictory elements depending on the perspective.

The central goals of our evaluation are to examine three well-established LfD algorithms and to 1) identify and analyze mismatches between core algorithmic assumptions and naïve user behavior, 2) examine user preferences for three distinct robot teaching interaction styles, and 3) compare algorithm usability and performance in a common domain. To reach our goals, we applied all three algorithms to a single task and ran a user study with 31 participants. Each participant trained a robot using each

of the three algorithms. In this work, we report the qualitative and quantitative results we obtained. We examine not only the test subjects’ performance with these methods, but also take a closer look at the preferences of the user and types of demonstrations they provide to each algorithm.

We present our quantitative findings about: a) the correlation between the number of user-agent interactions and the performance of the agent and b) the correlation between agent’s final performance and its perceived accuracy by the participant. Comparatively, we found the strongest correlation in CBA data. We also discuss the possible reasons of our qualitative results. Additionally, we identify common trends and misconceptions that arise when non-experts are asked to use these algorithms, with the aim of informing future Learning from Demonstration approaches. Our results show that users achieved better performance in teaching the task using the CBA algorithm, whereas the Interactive Reinforcement Learning algorithm modeled user behavior most accurately.

Next, we explain our experimental setup and task domain. We then present our qualitative and quantitative findings in Sections 4.4 and 4.5. Finally, in Section 4.6 we discuss the implications of our findings.

4.3 Evaluation

To evaluate the three algorithms, we recruited participants to teach a robotic agent accomplish a simple task using all three methods. We recruited 31 participants from the WPI area, 21 male and 10 female, with ages ranging from 18 to 35. Six of the participants self-identified as an expert roboticist, while 14 were novice and 11 had no robotics background.

Since the training time can be quite lengthy in certain domains [80], we aimed to find a domain that was simple enough to see a progress in learning in a relatively short amount of time, yet dynamic and random enough to *not* be trivial. With this in mind,

we designed the DBug domain for the Aldebaran Nao humanoid robot, which required the humanoid robot to catch and remove small hexbug robotic toys from a table.

4.3.1 User Study

For this user study we do not have a baseline control group. Instead of changing a specific variable and observing how user preference changes [82], we are interested in obtaining broader results. For example, instead of changing only the frequency of asking for help using a single learning technique, we give our test subjects completely different techniques, which are not variations of each other. Each participant was allotted one hour to perform the study. The study followed the following process:

- Introduction
- Training Session 1 (10 minutes)
- Survey Questions for Algorithm 1
- Training Session 2 (10 minutes)
- Survey Questions for Algorithm 2
- Training Session 3 (10 minutes)
- Survey Questions for Algorithm 3
- General Survey

Upon arrival, the participant was given a broad overview of the task that was being trained as well as the capabilities of the Nao. It was noted that the Nao would be able to track the bugs with its own vision system and each of the three preprogrammed actions were also explained.

Following the introduction, the participant used each algorithm in turn to train the agent. Before beginning a trial, the participant was given verbal and written instructions about how the algorithm worked and allowed to ask any questions. Following each training session the participant was asked to fill out a brief questionnaire relating

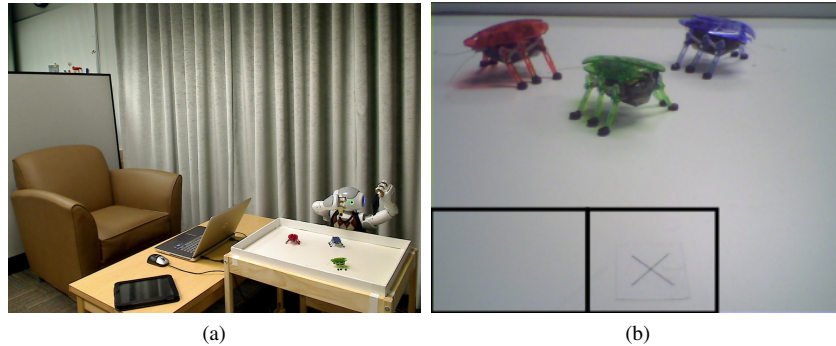


Figure 4.1: (a) Experimental setup, (b) View of the table from the Nao's on-board camera. Sweep zone: bottom-left portion of the figure. Pick-up zone: bottom-center portion of the figure. Any other portion of the image is Wait zone.

to the method that they had just used. The order of the algorithms was varied with participants, subdivided into three subgroups, based on the order in which they used the algorithms:

- Int-RL/BNets/CBA (10 participants),
- BNets/CBA/Int-RL (10 participants), and
- CBA/Int-RL/BNets (11 participants).

The training session for each algorithm was limited to 10 minutes. Participants were allowed to end training early, although none chose to do so. Once all three methods and questionnaires had been completed, a final general questionnaire was given which asked questions related to user preferences among the three methods.

4.3.2 Experimental Setup

The experimental setup, depicted in Fig. 4.1a, consisted of the Nao's tabletop setup, a laptop used by the participant to interact with the robot, and a webcam used for recording the session.

The tabletop setup consisted of the Nao seated at a small table (25" \times 19"). A red, blue, and green hexbug toy were placed on the table and allowed to move around.

For each test, bugs were randomly put on the table. The test subject had the option of turning the hexbugs on and off, or moving them around at will. A short wall was added to the perimeter of the table to prevent the bugs from walking over the edge. By default hexbugs move forward until either of their antenna touches to an obstacle (i.e. our short wall at the perimeter of the table, another hexbug, the test subject's hand, or our humanoid robot's manipulator). When the toy bumps into an obstacle, it moves backwards while rotating for a short period of time. In this manner, the bugs continuously move around the table unless turned off by the user.

When learning, the Nao's camera was directed straight at the center of the table. The field of view covered the majority of the table with small portions on the left and right being out of sight. The test subjects were not told how much of the workspace was exactly covered by the camera. Using its camera, the Nao was able to continuously track the x and y position of all bugs in its field of view (as seen in Fig. 4.1b). We used OpenCV [73] for all image processing.

In order to assist the Nao in picking up the bug toys, a magnet was placed into its right hand. A pickup zone was marked with an \times on the table in front of the Nao where it could reach down and attempt to attach a bug to the magnet (shown in Fig. 4.1b). Once a bug was attached to the magnet, the Nao raised its arm and asked the user to remove it manually. Additionally, a small foam brush was placed into the robot's left hand to help sweep bugs toward the pickup location. Both robot actions followed a precoded motion trajectory.

Participants used a laptop to start and stop the learning process and perform most of the interactions with the robot. A tablet computer was also provided, presenting written and pictorial instructions about how to use each training method. The task described was kept consistent across all three teaching methods.

The DBug Domain The domain representation, $D = (S, A)$, was defined by a finite set of states S and actions A . Each state $s = (x, y) \in S$ was defined by the x and

y coordinates of the hexbug in picture’s coordinate system, considering the top left corner to be $(0, 0)$. Since there were more than one hexbug on the table most of the time, bug with the smallest Euclidean distance to the pickup location was used as the current state. If no bugs were detected by the robot, the state was taken as $(0, 0)$.

Using a 320 by 240 pixel image from the Nao, a total of 76,800 uniquely identifiable states were possible in the domain. The domain space was considered to be large enough such that the solution was not trivial yet small enough that a user could make reasonable progress teaching the agent within a short period of time. In the case of Int-RL, we subsampled the state space to 100 states. This composed of a 10x10 grid (a rectangle of 32 by 24 pixels per (x, y) state) overlaid over the image. This was to increase the probability of visiting the same state more than once during the training so that the agent can make progress in learning during the relatively short period of time of the study.

The action space A is defined by three actions: picking up a bug at the center pickup location using the magnet in its right hand, sweeping inwards towards the center pickup location using the foam brush in its left hand, and waiting for 1 second (i.e., doing nothing). All actions were preprogrammed. The pickup action took ~ 9 seconds and the sweep action took ~ 5 seconds from start to finish. The wait action was an important action to include as it was useful in teaching the robot when it was not appropriate to either sweep or pickup (e.g., when no bugs were present around the pickup location).

Sampled across the complete state space, approximately 12.5% of the states fall into the sweep zone, 10.4% fall into the pick-up zone, and 77.1% fall into the wait zone (zones are shown in Fig. 4.1b).

4.4 Qualitative Results

In this section, we present the results of the qualitative analysis of the user study and our general observations. We take look at the algorithmic assumptions compared tech-

niques make and we comment on the usability of the algorithms from a practical perspective.

Algorithmic Assumptions We begin by examining the assumptions that the algorithms make and how naïve user behavior differed from those assumptions.

The Interactive Reinforcement algorithm makes the following assumptions:

1. A positive feedback will follow most correct actions,
2. A negative feedback will follow most incorrect actions.
3. Guidance will be given in order to narrow down the choice of the action it precedes.

This approach was very straightforward for the participants and we feel that all of the assumptions listed above held when users taught the task. We believe that the design process of the Int-RL algorithm has a big influence on that finding. Int-RL was originally designed alongside user studies with an online agent [24, 83], and, with this study, we can confirm that with a physical robot, the users interacted in a similar way compared to the online agent.

The Behavior Networks algorithm makes the following assumptions:

1. The teacher will make a sequence of demonstrations for each task, beginning each demonstration with START, performing one or more actions through kinesthetic teaching, and ending the demonstration with DONE.
2. If more than one demonstration sequence is performed for a given task (i.e. multiple START to DONE sequences), the demonstrations will result in complementary dependencies in the model, with no contradictory actions dependent on the same set of preconditions.

In our implementation of BNets, for the agent to understand that there was a hexbug to pick-up, we used a “found a bug in pick-up zone” behavior. Likewise, for understanding that there was a bug in sweep zone we used a “found a bug in sweep zone”

behavior. The precondition for these behaviors to be activated was to detect a hexbug in the pick-up or sweep zone. Ideally these behaviors should have been the precondition for a second behavior, either “pick-up the bug” or “sweep”. As stated in assumptions (1) and (2), the algorithm expects the teacher to demonstrate the task in the correct sequence. For instance, after a demonstration of “found a bug” (either by putting a hexbug in front of the camera manually, or by waiting for a hexbug to arrive randomly), a demonstration of “pick-up the bug” (by lowering the right arm of the robot and lifting it back up) would create a correct behavior network for picking up a hexbug upon detection.

We observed that the dynamic nature of the environment created a particular challenge for non-expert teachers using BNets. The pre-conditions for “found a bug in pick-up zone” or “found a bug in sweep zone” behaviors were often met unintentionally if the user did not control the location of the hexbugs (e.g., by turning them off and displacing them manually). This violated the first assumption. It was hard for most users to realize which behavior was activated when, because there was no visualization for the mental model of the agent.

Users were given the option of deleting the whole network and re-teaching the task, or keeping the current knowledge and adding new demonstrations on top of the existing network. Although users tried to erase and re-teach the task multiple times, none of them actually succeeded to teach the sweep action in the correct states. On the other hand an expert roboticist who knew about the necessary order of preconditions, could teach the task with only two demonstrations (one for pick-up and one of sweep) by controlling the location of hexbugs manually. Almost every study participant had an unintended behavior dependency. An example of an unintended dependency is executing a pickup action after every sweep, even if there is no bug in the pickup zone. Similar unintended dependencies were taught by every user, in many occasions repeatedly. The user intention conformed to assumption (2). In general the users tried to

demonstrate behaviors for the ultimate goal of sweeping and picking-up hexbugs as opposed to series of some random and conflicting series actions.

Confidence Based Autonomy with GMMs, makes the following assumptions:

1. The user will provide training data with multiple demonstrations for each action, effectively sampling the state space.
2. The user performs a correction for an incorrectly selected action during the time interval between the beginning of the incorrect action, and the beginning of the next action.

In our domain, each action had one area of the state space where the agent should have learned to execute it. With an ideal set of demonstrations the agent would have 3 - 4 Gaussian models in total (e.g., one model per action centered in that specific zone). We observed that the demonstrations given by the participants were very scattered and often resulted in multiple Gaussians for each action. Instead of being grouped around one center per action, Gaussian models were scattered in the state space. This decreased the efficiency of decision making and accuracy of the model.

Samples were often given either too sparse to be in the same Gaussian model, or different models were overlaid due to different action labels given in very similar states. Furthermore, three users violated assumption (1) by not providing enough training data for the classifier to confidently make any predictions (minimum of 3 demonstrations of each action).

Several users also violated assumption (2) by not providing corrections within the required timeframe. The delay between the transition time of the real world (e.g., $time = t$) and the state that the users wanted to give correction for (e.g., $time = k < t$) led to a mismatch between states and labels, corrupting some of the training data.

Usability For Int-RL, in terms of usability, we observed that 10 minutes was not sufficient to enable inexperienced users to train the task well (task performance is further

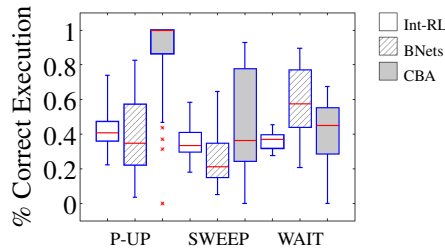


Figure 4.2: Percentage of correctly executed actions in all three of the methods. The red bars in each box show median values. The whiskers cover 3 interquartile range (IQR) and the red dots show the data points that lie between 3 and 5 IQR.

evaluated in Section 4.5). However, we note that the authors of this chapter, who are familiar with the inner workings of the algorithms, were able to teach the task using all three methods within this time frame. This highlights the difference between experts and inexperienced users. Many users could not see an improvement in learning until the end of the experiment, which led some to stop rewarding or to vary their reward strategy to try and achieve a better response.

In BNets, we received positive comments about the ease of kinesthetic teaching; however, users often tried to teach the task by making new, yet incorrect, demonstrations to the agent. Even when they erased the learned network and restarted teaching, instead of adding behaviors one by one (such as teaching picking-up the hexbug in one demonstration, then teaching sweeping the hexbug in a second demonstration) they moved both arms at the same time thus creating simultaneous actions. Since the environment was dynamic, it caused learned networks to be overly complicated.

Many users were able to achieve relatively high performance using the CBA algorithm. Due to the dynamic nature of the Dbug domain, the environment did not pause when the robot requested help, as in previous applications of CBA [9]. The robot was in a constant cycle of “perception–decision–execution” and the decision process (with added small wait commands to give people a time to do corrections) was perceived as lag by our participants. This is a practical issue. The study of inexperienced users across dynamic domains and ones in which the state does not change while the robot asks for help on teaching could be an interesting future work.

4.5 Quantitative Results

To evaluate user performance at teaching the task, we examine the final policy of each algorithm at the end of the 10 minute training session and evaluate its performance in selecting the correct action for each state. For each method, we check if the agent executes the correct action at a given zone (shown in Fig. 4.1b). The correct action label for each state was pre-determined by the region in which that state was located.

Figure 4.2 compares the percentage of correctly taught state-action pairs for each algorithm. On average, our subjects taught the pick-up action (the left three plots) and the sweep action (the middle three plots) with highest accuracy using CBA. The second most accurate result was achieved with Int-RL for these two actions. However subjects taught the wait action with higher accuracy using BNets. It is worth noting that the wait action was implicitly taught in BNets, that is, if the teacher does not demonstrate any action by moving the arms of the robot (for instance when there is no bug in pick-up zone), then the agent learns to not-to-act, i.e. to wait. Based on the lower accuracy rate of the other two actions, we think that this implicit definition is the reason why people had more success with BNets for this specific action.

In the rest of this section we evaluate algorithm-specific performance statistics. We are specifically interested in two data patterns: a) the correlation between the number of user-agent interactions and the accuracy of the final policy, and b) the correlation between the accuracy of agent’s final policy and the user’s satisfaction with the agent’s performance (i.e. perceived accuracy). We use the Pearson correlation coefficient to report our findings. For each algorithm we also give a visual example of one successful and one unsuccessful final policy (or plan) in order to highlight the difference.

Interactive Reinforcement Learning Results For our implementation of Interactive Reinforcement Learning, we used Q-learning with ϵ -greedy action selection. We used the following parameters for Q-learning: 0.5 for initial Q-values, with a learning rate

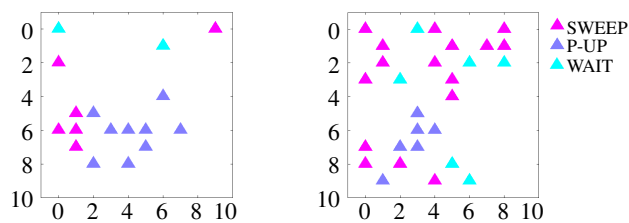


Figure 4.3:
Example Int-
RL Policies.

(a) An Example of Successful Policy (b) An Example of Unsuccessful Policy

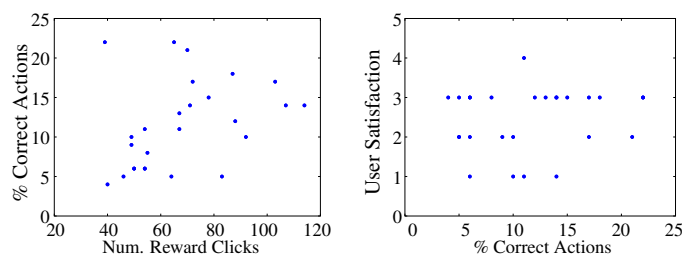


Figure 4.4:
Int-RL
User
Statistics.

(a) Effect of user interaction on agent's performance (b) User satisfaction vs. performance

$\alpha = 0.3$ and discount factor $\gamma = 0.75$. Fig. 4.3 shows a successful and unsuccessful sample Int-RL policy. Each triangle shows a sub-sampled state that was not “tied” (where, more than one action had the same Q-value). Although every (x, y) pair symbolizes a subsampled state, to increase the readability of the plots, we left “tied” states empty. Ties can be caused by that state being unexplored or more than one action being equally rewarded by the teacher, i.e. noisy demonstrations. For example in Fig. 4.3 (a), approximately 20% of the states were rewarded enough to assign a higher Q-value to one action than the others. For a given state, the action that has the highest Q-value (shown in Fig. 4.3) is executed during exploitation of the policy. If the state is equal to one of the empty states, then the action is selected randomly.

Fig. 4.4 (a) shows how the frequency of demonstrations is correlated with the agent’s final accuracy. The scatter plot presents the number of inputs given by the user (positive or negative rewards) and the resulting policy accuracy. The correlation coefficient is found as $R = 0.35$ for this plot. This shows a very weak positive correlation

between the two variables. That is, the increase in the number of rewards given is approximately 35% responsible of the increase in final accuracy of the agent. Fig. 4.4 (b) shows the scatter plot for agent’s final policy accuracy versus the users’ self-reported satisfaction with the agent’s performance in response to the question “How well did the robot learn the task?” (1-Not at all, 4-Very well). The correlation coefficient for this plot is $R = 0.16011$, which shows almost no correlation between the participants’ perceived accuracy and the real performance of the agent. We believe that one reason for this finding can be the overall poor performance of the final policies. When we look at the y-axis, we can see that the maximum accuracy shows that approximately 22% of the state space was labeled correctly.

Behavior Networks Results Two of the networks taught by our participants are shown in Fig. 4.5. Each behavior is shown in an ellipse. Start and restart are pseudo-behaviors that signal the beginning and the end of the network. Dotted lines indicate that the behaviors above must happen before those below. Dashed lines are enabling preconditions, which indicate behaviors that enable the activation of others. Solid lines show permanent preconditions, that is, the behavior above should remain active while behaviors below are also active.

Fig. 4.5(a) shows a simple and correct Behavior Network. When executed, this network would execute the pick-up action in presence of a bug in the pick-up zone, which is the correct behavior. Fig. 4.5(b) shows a network that waits until it finds a hexbug in pick-up zone for 4 cycles (FBIPZ_23,24,25,26), then picks up the hexbug (P-UP_7), it also requires to find another hexbug in order to restart executing the network (FBIPZ_27 is a precondition of restart_1). This network is inefficient for the target task, and although it still executes the pick-up action after finding several hexbugs, the unnecessary interdependencies between behaviors reduce the efficiency of this policy.

The Behavior Networks algorithm differs from the other two techniques in that the user demonstrations do not accumulate over time. The BNets algorithm generates a

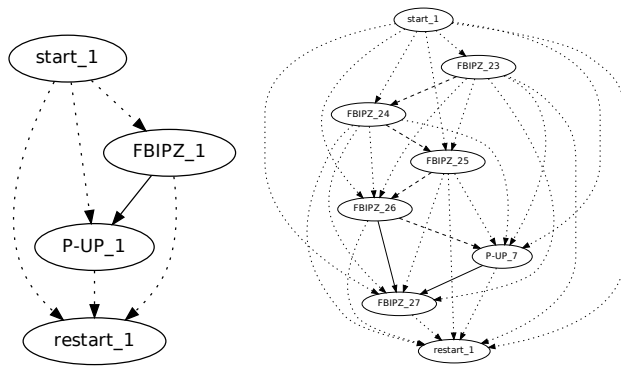
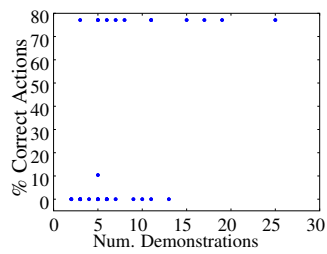
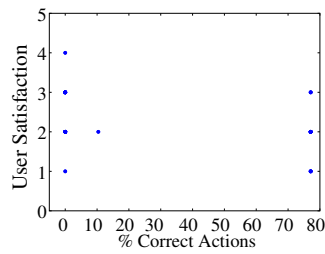


Figure 4.5: Examples of the networks demonstrated by the subjects. P-UP: Pick-Up, FBIPZ: Found a bug in Pick-Up Zone. Numbers in the end of the behaviors show their unique id numbers.

(a) A simple and correct Behavior Network. (b) A relatively more complicated Behavior Network produced by a participant.



(a) Effect of user interaction on agent's performance



(b) User satisfaction vs. performance

Figure 4.6: B Nets User Statistics.

policy from the ordered set of demonstrations performed between the user-indicated START and END signals. As a result, the entire policy can be taught in a very short period of time by an expert (under a minute for the authors).

A non-expert teacher who has little experience with the algorithm can demonstrate the task multiple times and the agent may still perform poorly. Indeed, we observed this outcome with several of our participants, who explored the behavior of the algorithm through trial and error to gain understanding of kinesthetic teaching, the robot’s perception, and the conditioning behavior of the algorithm (pre- vs. post-condition). In summary, a large number of demonstrations is not required to teach a successful policy, however, we expected our participants to get better at demonstrating the task over time.

Fig. 4.6(a) shows the scatter plot of the total number of demonstrations versus policy performance. We note that there are only three levels for the agent’s accuracy. This behavior results from the pre and post-conditions precoded into the algorithm. For example, once the BNets agent learns to pick-up a hexbug when it is located in the pick-up zone, the correct pickup behavior is automatically learned for every state in the pickup zone. This is due to the fact that the pre-condition of the pickup action generalizes across a region of states (the entire pickup area) and not a single (x, y) location.

Keeping these facts in mind, the three levels in Fig. 4.6(a) are respectively at 0%, 10.4% and 77.1%, referring to accurate performance of *none of the actions*, *pick-up*, and *wait*. No participant was able to teach the *sweep* action successfully in their final policy. Importantly, we note that although the policy performance is 77.1% for correctly teaching the wait action, this is caused by the fact that in most states, our domain required to wait.

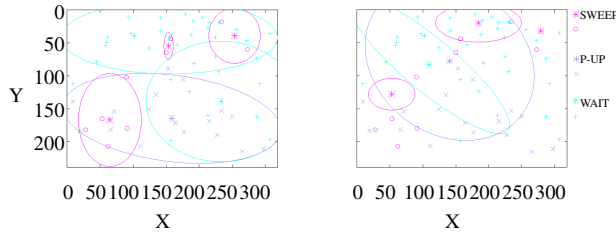
The correlation coefficient found for this plot is $R = 0.48$. This shows some (although not so strong) correlation between the participants’ experience in teaching (i.e. number of demonstrations) and the accuracy of the final policy.

Finally, Fig. 4.6(b) shows the scatter plot of the agent’s accuracy versus the participants’ self-reported satisfaction in response to the question “How well did the robot learn the task?”. Although weak, the correlation coefficient $R = -0.35$ shows a negative correlation between these two variables. We believe that the correlation coefficient is negative because of the fact that users evaluated performance based on the success of picking up as many hexbugs as possible. Hence, *waiting* at appropriate times was not considered as successful as *picking up* or *sweeping*. Our measure of accuracy gives equal weight to all state-action pairs across the state space.

Confidence Based Autonomy Results In order to assess the user statistics with CBA, we look at the final GMM classifier that was constructed for each user.

A sample of a successful and unsuccessful CBA policy is given in Fig. 4.7. To help the reader compare both policies, we highlight several characteristics of the learned classifier. First, when we look at the Gaussian models for the *wait* action in Fig. 4.7(a), we see two distinct models, covering most of the correct zones, whereas the Gaussian for the same action in Fig. 4.7(b) covers only a small part of the correct zone, and is mostly overlaid with other Gaussians. Second, in Fig. 4.7(a), the Gaussian for *pick-up* action is centered very close to its zone, whereas in Fig. 4.7(b), it covers most of the wait zone. Third, the biggest Gaussian model for *sweep* action is centered close to its zone in Fig. 4.7(a) whereas in Fig. 4.7(b) Gaussians are in the wait zone. Finally, two more Gaussian models for *pick-up* action can be seen in Fig. 4.7(a), it is possible that they are the results of the violation of our assumption (2) mentioned in the previous section; when the user trains the agent with more training samples, the risk of obtaining more Gaussian models than necessary increases. Furthermore, incorrectly timed corrections introduce noise and lead to poor model accuracy.

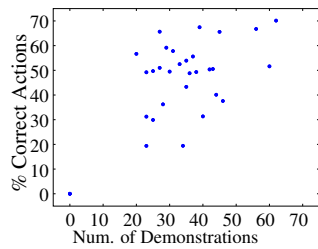
As in previous methods, we calculate policy performance as the percentage of correctly labeled states. The scatter plot in Fig. 4.8(a) shows how number of demonstrations is correlated with policy performance. The correlation coefficient found for



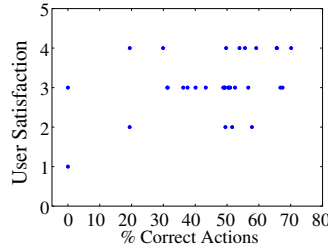
(a) An Example of Successful Policy

(b) An Example of Unsuccessful Policy

Figure 4.7:
Example CBA Policies.



(a) Effect of user interaction on agent's performance



(b) User satisfaction vs. performance

Figure 4.8:
CBA User Statistics.

this plot is, $R = 0.64$, which implies a somewhat strong correlation between these two variables. Note that it would be inaccurate to compare the strength of this correlation to our previous number of demonstrations versus performance plots; although the policy performance is calculated using the equivalent data, the values representing the number of demonstrations have inherently different meaning due to the differences in the algorithms. As a result, we can *not* conclude that in CBA user demonstrations were more effective to improve the accuracy of the agent than in previous methods. However, we can conclude that when using CBA, user demonstrations seem to be accountable for 64% of the improvement in policy performance.

Fig. 4.8(b) shows the scatter plot of participant self-reported performance evaluation versus final policy performance. The correlation coefficient found for this plot is $R = 0.39$, which implies a weak correlation between our participants' votes and the actual performance of their agents. The question we asked was the same across all methods, hence the variable on x-axis is comparable to previous methods. Likewise,

the agent accuracy was calculated in the same way across all methods. As a result, we can compare the correlation coefficient we obtain for this plot to our previous findings. We found no correlation in Fig. 4.4(b), a negative correlation in Fig. 4.6(b), and a positive correlation in Fig. 4.8(b). Based on these results, we conclude that participants were able to assess performance of their agents most accurately when using the CBA algorithm.

4.6 Discussion

Moving away from performance statistics, in this section we discuss several design factors for these algorithms.

Participant Feedback The general survey included space where participants were able to write free-form feedback about the algorithms in the study. All 31 participants included feedback on their form; the following three topics were most frequently repeated:

- the need for additional feedback from the robot regarding what it was sensing,
- the need for better understanding of what the robot “is thinking”, or what it understands about the task,
- the need of communicating with the robot more naturally, specifically through speech.

These three points, especially the importance of feedback, are surely not unprecedented findings. However, their importance is often underestimated in existing LfD methods, and our aim is to draw attention to these factors for future research.

Factors Affecting Agent Performance Our user study has four main components for each method: the teacher, the interface, the algorithm and the task domain. The teacher

and the task domain were the common components across all three experiments performed with each participant, whereas the interface and algorithms were held constant across participants. Quantifying the effect of each component on different aspects of the agent's performance requires a narrower, more focused user study. However based on our observations we can make general comments on how human and algorithmic factors contribute to each algorithm's performance.

Teaching the policy using the Int-RL algorithm requires more time than when using other methods. This is an algorithmic factor due to the fact that the agent selects its actions probabilistically and converges to a stable policy only after repeated exposure to feedback. The occurrence of users abandoning rewards before the agent converges to a stable policy (possibly due to lack of feedback) is a human factor. People quitting early because learning is too slow might be considered to be a combination of both.

Teaching a BNets agent often requires trial and error. This is an algorithmic factor; the necessity of defining pre/post-conditions for each behavior leaves a knowledge gap between the person who defines them and the person who makes demonstrations. On the other hand, realizing the relationship between given demonstrations and resulting behaviors of the robot is a human factor. Again, people giving up on trial and error process after a few demonstrations (because robot is now behaving any differently) might be considered to be a mixed effect of both factors on agent's final performance.

Teaching a CBA agent requires demonstrations and corrections. There is requirement minimum number of demonstrations for the agent to begin classifying the training data. This is an algorithmic factor which may change based on the classifier implemented. Modeling the timing of corrections, on the other hand, is a human factor. Depending on the situation, for instance if the teacher is distracted and reacts slowly, corrections may have a negative impact on agent's final performance.

Required Task Knowledge by the Programmer For the Int-RL and CBA algorithms, all that must be defined by the programmer is the state and action sets of the

robot. For the Behavior Networks algorithm, the programmer needs to know not only the state and actions, but also the state values that will result from the execution of different actions in order to form the pre/post-conditions of behaviors. We found that coding the Behavior Networks required a significantly higher degree of knowledge of the domain, as well as a precise measurement of the sensor values that would be observed. Without this pre-coded information, the robot's state would not match the post-conditions of any behaviors and no behaviors would be activated and learned.

To sum up, in this chapter we present the findings we obtained from a user study implemented in a real-world domain, in which we asked non-expert users to use and evaluate three different robot learning from demonstration algorithms. The three algorithms selected – Behavior Networks, Interactive Reinforcement Learning, and Confidence Based Autonomy – utilize distinctly different policy learning and demonstration approaches, enabling the study of a broad spectrum of the field.

We analyzed the user study data with respect to robot policy performance and user feedback. Our results show that users achieved better performance in teaching the task using the CBA algorithm. Users also positively rated the usability of the CBA algorithm, with many users showing preference for the ability to give direct commands to the robot. Many participants also positively reviewed the kinesthetic teaching interface used in the BNets algorithm. However, users were often confused by the opaqueness of the teaching experience and performance of the result.

In case of the Interactive Reinforcement Learning algorithm, despite our use of a sub-sampled state-space to improve algorithm performance, the experiment time was not sufficient to allow participants to train the complete task. However, we found that the Int-RL algorithm most accurately modeled user behavior, whereas both CBA and BNets made algorithmic assumptions that did not hold when the algorithms were applied to naïve users.

An interesting next step is to extract the strengths of the three individual algorithms

and to design a new integrated method for learning from demonstration. The results of our study indicate that presenting a model of the agent's internal policy to the teacher during training may improve transparency and usability of future algorithms.

Chapter 5

Human-Agent Transfer: HAT

In the previous chapter we presented the results of a user study where non-expert teachers demonstrated a real-world task to a humanoid robot during run-time. One of the findings of the user study was that the participants were not able to teach the task to the robot using Interactive Reinforcement Learning within the allocated time, even though an expert demonstrator was able to do so. In this chapter, we introduce *Human-Agent Transfer* (HAT), where teachers can make off-line demonstrations without a strict time limitation. HAT combines transfer learning, learning from demonstration and reinforcement learning to achieve rapid learning and high performance in complex domains. Through experiments in a simulated robot soccer domain, we show that human demonstrations transferred into a baseline policy for an agent and refined using reinforcement learning significantly improve both learning time and policy performance. Our evaluation compares three algorithmic approaches to incorporating demonstration rule summaries into transfer learning, and studies the impact of demonstration quality and quantity, as well as the effect of combining demonstrations from multiple teachers. Our results show that all three transfer methods lead to statistically significant improvement in performance over learning without demonstration. The best performance was achieved by combining the best demonstrations from two teachers.

5.1 Overview

Agent technologies for virtual agents and physical robots are rapidly expanding in industrial and research fields, enabling greater automation, increased levels of efficiency, and new applications. However, existing systems are designed to provide niche solutions to very specific problems and each system may require significant effort to develop. The ability to acquire new behaviors through learning is fundamentally important for the development of general-purpose agent platforms that can be used for a variety of tasks.

Existing approaches to agent learning generally fall into two categories: independent learning through exploration and learning from labeled training data. Agents often learn independently from exploration via *Reinforcement learning* (RL) [84]. While such techniques have had great success in offline learning and software applications, the large amount of data and high exploration times they require make them intractable for most real-world domains.

On the other end of the spectrum are *learning from demonstration* (LfD) algorithms [70]. These approaches leverage the vast experience and task knowledge of a person to enable fast learning, which is critical in real-world applications. However, human teachers provide particularly noisy and suboptimal data due to differences in embodiment (e.g., degrees of freedom, action speed) and limitations of human ability. As a result, final policy performance achieved by these methods is limited by the quality of the dataset and the performance of the teacher.

In this chapter, we propose a novel approach: use RL *transfer learning* methods [51] to combine LfD and RL and achieve both fast learning and high performance in complex domains. In transfer learning, knowledge from a *source task* is used in a *target task* to speed up learning. Equivalently, knowledge from a source agent is used to speed up learning in a target agent. For instance, knowledge has been successfully transferred between agents that balance different length poles [85], that solve a series

of mazes [86, 87], or that play different soccer tasks [53, 88, 54]. The key insight of transfer learning is that previous knowledge can be effectively reused, even if the source task and target task are not identical. This results in substantially improved learning times because the agent no longer relies on an uninformed (arbitrary) prior.

5.2 Description of HAT Algorithm

In this work, we show that we can effectively transfer knowledge from a human to an agent, even when they have different perceptions of state. Our method, *Human-Agent Transfer* (HAT): 1) allows a human teacher to perform a series of demonstrations in a task, 2) uses an existing transfer learning algorithm, *Rule Transfer* [52], to learn rule-based summaries of the demonstration, and 3) integrates the rule summaries into RL, biasing learning while also allowing improvement over the transferred policy.

We perform empirical evaluation of HAT in a simulated robot soccer domain. We compare three algorithms for incorporating rule summaries into reinforcement learning, and compare learning performance for multiple demonstration source, quantity, and quality conditions. Our findings show statistically significant improvement in performance for all variants of HAT over learning with no prior. Additionally, we find that exposure even to suboptimal demonstration training data results in significant improvements over random exploration, and combining demonstrations from multiple teachers leads to the best performance.

Our approach for combining LfD and RL (HAT) consists of three steps, motivated by those used in Rule Transfer:

1. **Demonstration:** The agent performs the task under the teleoperated control by a human teacher, or by executing an existing suboptimal controller. During execution, the agent records all state-action transitions. Multiple task executions may be performed.

2. **Policy Summarization:** HAT uses the state-action transition data recorded during the Demonstration phase to derive rules summarizing the policy. These rules are used to bootstrap autonomous learning.
3. **Independent Learning:** The agent learns independently in the task via reinforcement learning, using the policy summary to bias its learning. In this step, the agent must balance exploiting the transferred rules with attempting to learn a policy that outperforms the transferred rules.

In contrast to transfer learning, HAT assumes that either 1) the demonstrations are executed on the same agent, in the same task, as will be learned in the Independent Learning phase, or that 2) any differences between the agent or task in the demonstration phase are small enough that they can be ignored in the independent learning phase. Instead of transferring between different tasks, HAT focuses on transferring between different agents with different internal representations. For instance, it is not possible to directly use a human’s “value function” inside an agent because 1) the human’s knowledge is not directly accessible and 2) the human has a different state abstraction than the agent.

We next present three different ways that HAT can use a decision list to improve independent learning.

Value Bonus The intuition behind the *Value Bonus* method [52] is similar to that of shaping in that the summarized policy is used to add a reward bonus to certain human-favored actions. When the agent reaches a state and calculates $Q(s, a)$, the Q-value of the action suggested by the summarized policy is given a constant bonus (B). For the first C episodes, the learner is forced to execute the action suggested by the rule set. This is effectively changing the initialization of the Q-value function, or, equivalently [89], providing a shaping reward to the state-action pairs that are selected by the rules.

We use $B = 10$ and $C = 100$ to be consistent with past work [52]; the Q-value for

the action chosen by the summarized policy will be given a bonus of +10 and agents must execute the action chosen by the summarized policy for the first 100 episodes.

Extra Action The *Extra Action* method [52] augments the agent so that it can select a *pseudo-action*. When the agent selected this pseudo-action, it executed the action suggested by the decision list. The agent may either execute the action suggested by the transferred rules, or it can execute one of the “base” MDP actions. Through exploration, the RL agent can decide when it should 1) follow the transferred rules by executing the pseudo-action or 2) execute a base MDP action (e.g., the transferred rules are sub-optimal). Were the agent to always execute the pseudo-action, the agent would never learn but would simply mimic the demonstrated policy.

As with the Value Bonus algorithm, the agent initially executes the action suggested by the decision list, allowing it to estimate the value of the decision list policy. We again set this period to be 100 episodes ($C = 100$).

Probabilistic Policy Reuse The third method used is *Probabilistic Policy Reuse*, based on the π -reuse Exploration Strategy [90, 86]. In Probabilistic Policy Reuse, the agent will reuse a policy with probability ψ , explore with probability ϵ , and exploit the current value function with probability $1 - \psi - \epsilon$. By decaying ψ over time, the agent can initially leverage the decision list, but then learn to improve on it if possible. Note that Probabilistic Policy Reuse is similar to the recent TAMER+RL method #7 [55], where the agent tries to execute the action suggested by the learned human shaping reward, rather than follow a transferred policy.

5.3 Evaluation

In this section, we first introduce Keepaway [91], a simulated robot soccer domain and then we explain the experimental methodology we used to evaluate HAT.

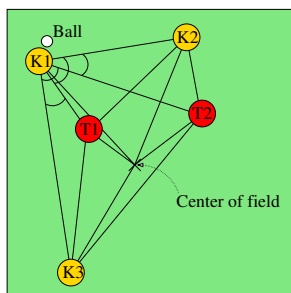


Figure 5.1: This diagram shows the distances and angles used to construct the 13 state variables used for learning with 3 keepers and 2 takers. Relevant objects are the 3 keepers (K) and the two takers (T), both ordered by distance from the ball, and the center of the field.

5.3.1 Keepaway

Keepaway is a domain with a continuous state space and significant amounts of noise in the agent’s actions and sensors. One team, the *keepers*, attempts to maintain possession of the ball within a $20\text{m} \times 20\text{m}$ region while another team, the *takers*, attempts to steal the ball or force it out of bounds. The simulator places the players at their initial positions at the start of each episode and ends an episode when the ball leaves the play region or is taken away from the keepers.

The keeper with the ball has the option to either pass the ball to one of its two teammates or to hold the ball. In *3 vs. 2 Keepaway* (3 keepers and 2 takers), the state is defined by 13 hand-selected state variables (see Figure 5.1) as defined in [91]. The reward to the learning algorithm is the number of time steps the ball remains in play after an action is taken. The keepers learn in a constrained policy space: they have the freedom to decide which action to take only when in possession of the ball. Keepers not in possession of the ball are required to execute the `Receive` macro-action in which the player who can reach the ball the fastest goes to the ball and the remaining players follow a handcoded strategy to try to get open for a pass.

For policy learning, the *Keepaway* problem is mapped onto the discrete-time, episodic RL framework. As a way of incorporating domain knowledge, the learners choose not from the simulator’s primitive actions but from a set of higher-level macro-actions implemented as part of the player [91]. These macro-actions can last more than one time step and the keepers have opportunities to make decisions only when an on-going

macro-action terminates. Keepers can choose to `Hold` (maintain possession), `PASS1` (pass to the closest teammate), and `PASS2` (pass to the further teammate). Agents then make decisions at discrete time steps (when macro-actions are initiated and terminated).

To learn Keepaway with Sarsa, each keeper is controlled by a separate agent. Many kinds of function approximation have been successfully used to approximate an action-value function in Keepaway, but a Gaussian Radial Basis Function Approximation (RBF) has been one of the most successful [92]. All weights in the RBF function approximator are initially set to zero; every initial state-action value is zero and the action-value function is uniform. Experiments in this chapter use the public versions 11.1.0 of the RoboCup Soccer Server [93], and 0.6 of UT-Austin’s Keepaway players [92].

5.3.2 Experimental Setup

In the Demonstration phase of HAT, Keepaway players in the simulator are controlled by the teacher using the keyboard. This allows a human to watch the visualization and instruct the keeper with the ball to execute the `Hold`, `PASS1`, or `PASS2` actions. During demonstration, we record all (s, a) pairs selected by the teacher. It is worth noting that the human has a very different representation of the state than the learning agent. Rather than observing a 13 dimensional state vector like the RL agent, the human uses a visualizer (Figure 5.2). It is therefore critical that whatever method used to glean information about the human’s policy does not require the agent and the human to have identical representations of state.

To be consistent with past work [92], our Sarsa learners use $\alpha = 0.05$, $\epsilon = 0.10$, and RBF function approximation. After conducting initial experiments with five values of ψ , we found that $\psi = 0.999$ was at least as good as other possible settings. In the Policy Summarization Phase, we use a simple propositional rule learner to generate



Figure 5.2: This figure shows a screenshot of the visualizer used for the human to demonstrate a policy in 3 vs. 2 Keepaway. The human controls the keeper with the ball (shown as a hollow white circle) by telling the agent when, and to whom, to pass. When no input is received, the keeper with the ball executes the `Hold` action, attempting to maintain possession of the ball.

a decision list summarizing the policy (that is, it learns to generalize which action is selected in every state). For these experiments, we use JRip, as implemented in Weka [94].

Finally, when measuring speedup in RL tasks, there are many possible metrics. In this chapter, we measure the success of HAT along three related dimensions. The initial performance of an agent in a target task may be improved by transfer. Such a *jumpstart* (relative to the initial performance of an agent learning without the benefit of any prior information), suggests that transferred information is immediately useful to the agent. In Keepaway, the jumpstart is measured as the average episode reward (corresponding to the average episode length in seconds), averaged over 1,000 episodes without learning. The jumpstart is a particularly important metric when learning is slow and/or expensive.

The *final reward* acquired by the algorithm at the end of the learning process (at 30 simulator hours) indicates the best performance achieved by the learner. This value is computed by taking the average of the final 1,000 episodes to account for the high degree of noise in the Keepaway domain.

The *total reward* accumulated by an agent (i.e., the area under the learning curve) may also be improved. This metric measures the ability of the agent to continue to learn after transfer, but is heavily dependent on the length of the experiment. In Keepaway, the total reward is the sum of the average episode durations at every integral hour of

training:

$$\sum_{t:0 \rightarrow n} (\text{average episode reward at training hour } t)$$

where the experiment lasts n hours and each average reward is computed by using a sliding window over the past 1,000 episodes.¹

5.4 Results

In this section, we present results showing that HAT can effectively use human demonstration to bootstrap RL in Keepaway agents.

To begin, we recorded a demonstration from a teacher (*Subject A*) which lasted for 20 episodes (less than 3 minutes). Next, we used JRip to summarize the policy with a decision list. The following rules were learned, where $state_k$ represents the k^{th} state variable, as defined in the keepaway task [92]:

$$\begin{aligned} & \text{if } (state_{11} \geq 74.84 \text{ and } state_3 \leq 5.99 \text{ and} \\ & \quad state_{11} \leq 76.26) \qquad \qquad \qquad \rightarrow \text{Action} = 1 \\ & \text{elseif } (state_{11} \geq 53.97 \text{ and } state_4 \leq 5.91 \text{ and} \\ & \quad state_0 \geq 8.45 \text{ and } state_8 \leq 7.06) \qquad \rightarrow \text{Action} = 1 \\ & \text{elseif } (state_3 \leq 4.84 \text{ and } state_0 \geq 7.33 \text{ and} \\ & \quad state_{12} \geq 43.66 \text{ and } state_8 \leq 5.57) \qquad \rightarrow \text{Action} = 2 \\ & \text{else} \qquad \qquad \qquad \qquad \qquad \qquad \rightarrow \text{Action} = 0 \end{aligned}$$

While not the focus of this work, we found it interesting that the policy was able to be

¹Recall that the reward in Keepaway is +1 per time step, where a time step is a 10th of a simulator second. Thus, the reward for the first hour of training is always $60 \times 60 \times 10 = 36000$ — a metric for the total reward over time must account for the reward *per episode* and simply summing the total amount of reward accrued is not appropriate.

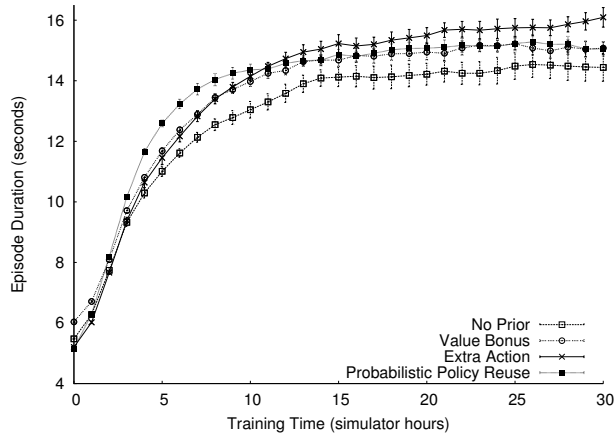


Figure 5.3: This graph summarizes performance of Sarsa learning in Keepaway using four different algorithms. One demonstration of 20 episodes was used for all three HAT learners. Error bars show the standard error in the performance.

summarized with only four rules, obtaining over 87% accuracy on when using stratified cross-validation. The data was dealt out pseudo-randomly to ensure that each fold had similar mean values.

We note that without employing independent learning, just following the learned rules, results in poor performance. The primary cause is that the rules represent a concise summary of the behavior but fail to cover many edge cases, as well as fail to adapt to states that deviate remarkably from the demonstrated behavior.

Finally, agents are trained in 3 vs. 2 Keepaway without using transfer rules (No Prior), using the Value Bonus, using the Extra Action, or using the Probabilistic Policy Reuse method. All learning algorithms were executed for 30 simulator hours (processor running time of roughly 2.5 hours) to ensure convergence.

Figure 5.3 compares the performance of the four methods, averaged over 10 independent trials. Using 20 episodes of transferred data from *Subject A* with HAT can improve the jumpstart, the final reward, and the cumulative reward. The horizontal line in the figure shows the average duration of the teacher’s demonstration episodes; all four of the RL-based learning methods improve upon and outperform the human teacher. The performance of the different algorithms is measured quantitatively in Table 5.1, where significance is tested with a Student’s t-test.

Method	Jumpstart	Final	Total Reward
<i>No Prior</i>	N/A	14.3	380
<i>Value Bonus</i>	0.57	15.1	401
<i>Extra Action</i>	-0.29	16.0	407
<i>Probabilistic Policy Reuse</i>	-0.30	15.2	411

Table 5.1: This table shows the jumpstart, final reward and total reward metrics for Figure 5.3. Values in **bold** have statistically significant differences in comparison to the No Prior method ($p < 0.05$).

While the final reward performance of the all four methods is very similar (only Extra Action has a statistically significant² improvement over No Prior), the total reward accumulated by all three algorithms is significantly higher than with No Prior learning. This result is an indication that although the same final performance is achieved in the long term because the learning algorithm is able to learn the task in all cases, high performance is achieved *faster* by using a small number of demonstrations. This difference can be best observed by selecting an arbitrary threshold of episode duration and comparing the number of simulation hours each algorithm takes to achieve this performance. In the case of a threshold of 14 seconds, we see that No Prior learning takes 13.5 hours, compared to 10.1, 8.57 and 7.9 hours for Value Bonus, Extra Action and Probabilistic Policy Reuse respectively. These results show that transferring information via HAT from the human results in significant improvements over learning without prior knowledge.

Section 5.4 will explore how performance changes with different types or amounts of demonstration, while Section 5.4 discusses how teacher ability affects learning performance. In all further experiments we use the Probabilistic Policy Reuse method as it was not dominated by either of the other two methods. Additionally, in some trials with other methods we found that the learner could start with a high jumpstart but fail to improve as much as other trials. We posit this is due to becoming stuck in a local minimum. However, because ψ explicitly decays the effect from the rules, this

²Throughout this chapter, t-tests are used to calculate significance, defined as $p < 0.05$.

phenomena was never observed when using Probabilistic Policy Reuse.

Comparison of Different Teachers Above, we used a single demonstration data set to evaluate and compare three algorithms for incorporating learned rules into reinforcement learning. In this section, we examine how demonstrations from different people impact learning performance of a single algorithm, Probabilistic Policy Reuse. Specifically, we compare three different teachers:

1. *Subject A*: This teacher has many years of research experience with the Keepaway task. (The same as Figure 5.3.)
2. *Subject B*: This teacher is new to Keepaway, but practiced for approximately 100 games before recording demonstrations.
3. *Subject C*: This teacher is an expert in LfD, but is new to Keepaway. The teacher practiced 10 games before recording demonstrations.

Each teacher recorded 20 demonstration episodes while trying to play Keepaway to the best of their ability. Figure 5.4 summarizes the results and compares performance of using these three demonstration sets against learning the Keepaway task without a prior. All reported results are averaged over 10 learning trials. Table 5.2 presents summary of the results, highlighting statistically significant changes in bold.

Method	Jumpstart	Final	Total Reward
<i>No Prior</i>	N/A	14.3	380
<i>Subject A</i>	-0.30	15.2	411
<i>Subject B</i>	3.35	15.7	423
<i>Subject C</i>	0.15	16.2	424

Table 5.2: This table shows the jumpstart, final reward and total reward metrics for Figure 5.4, where all HAT methods use Probabilistic Policy Reuse with 20 episodes of demonstrated play. Values in **bold** have statistically significant differences in comparison to the No Prior method.

All three HAT experiments outperformed learning without a bias from demonstration, with statistically significant improvements in total reward. However, as in any

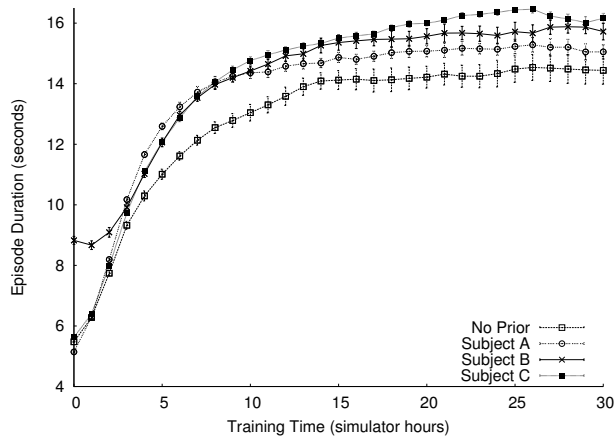


Figure 5.4: This graph summarizes performance of no prior learning and Probabilistic Policy Reuse learning using demonstrations from three different teachers. Each teacher performed demonstrations for 20 episodes. Error bars show the standard error in performance across 10 trials.

game, different Keepaway players have different strategies. While some prefer to keep the ball in one location as long as possible, others pass frequently between keepers. As a result, demonstrations from three different teachers led to different learning curves. Demonstration data from *Subjects A* and *C* resulted in a low jumpstart, while *Subject B*'s demonstration gave the learner a significant jumpstart early in the learning process. The final reward also increased for all three HAT trials, with statistically significant results in the case of *Subjects B* and *C*. These results indicate that HAT is robust to demonstrations from different people with varying degrees of task expertise.

An important factor to consider with any algorithm that learns from human input, is whether combining demonstrations from two or more different teachers helps the agent to learn faster, or whether exposure to possibly conflicting demonstrations from different teachers slows the learning process. In the following evaluation we compared five demonstration types:

1. *Subject A (20)*: Set of the original 20 demonstrations by Subject A: average duration of 10.4 seconds/episode
2. *Subject A (10)*: Set of 10 randomly selected demonstrations by Subject A: average duration 7.5 seconds/episode
3. *Subject C (20)*: Set of the original 20 demonstrations by Subject C : average

duration of 11.3 seconds/episode

4. *Subjects A + C Best (20)*: The 10 best (longest) demonstration episodes each from Subjects A and C: average duration of 17.2 and 18.0 seconds/episode, respectively
5. *Subjects A + C Worst (20)*: The 10 worst (shortest) demonstration episodes each from Subjects A and C: average duration of 4.6 seconds/episode for both

This analysis provides insight about the impact of combining demonstrations from multiple teachers (conditions 1 and 3 vs. 4 and 5) and the impact of demonstration quantity (condition 1 vs. 2) and quality (condition 4 vs. 5). Figure 5.5 presents a comparison of the five learning conditions, and Table 5.3 summarizes the results.

Method	Jumpstart	Final	Total Reward
<i>Subject A (20)</i>	-0.30	15.2	411
<i>Subject A (10)</i>	-2.23	15.8	407
<i>Subject C (20)</i>	0.15	16.2	424
<i>Subjects A + C Best</i>	2.15	15.7	431
<i>Subjects A + C Worst</i>	0.37	16.1	419

Table 5.3: This table shows the jumpstart, final reward and total reward metrics for Figure 5.5, where all HAT methods use Probabilistic Policy Reuse with 20 demonstrated episodes. Values in **bold** have statistically significant differences in comparison to the No Prior method (not shown).

With respect to learning from multiple teachers, results show that combining data from different subjects leads to performance as good as or better than learning from a single teacher. Condition *Subjects A + C Best* performs better than either *Subject A* or *Subject C* alone, and significantly outperforms all other methods in the group, in large part due to the early lead it has due to its high jumpstart. Condition *Subjects A + C Worst* shows no statistically significant change in performance between it and learning from *Subject A* or *Subject C* alone.³ This result is significant because it indicates that

³Note that because we have few subjects, our claims of significance are limited to results from demonstrations with the three subjects tested. Future work will generalize our findings by considering many more subjects.

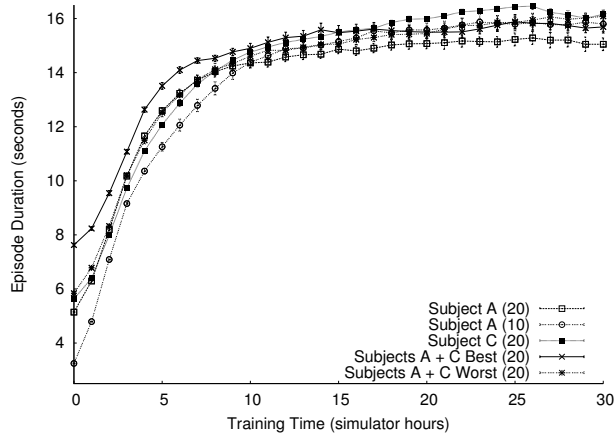


Figure 5.5: This graph summarizes performance of Probabilistic Policy Reuse learning using five different demonstration sets. Error bars show the standard error in performance across 10 trials.

while quality is important, as shown by the difference between *Subjects A + C Best* and *Worst*, any demonstration is beneficial. The fact that the worst demonstrations still lead to performance well above *No Prior* learning is an indication that exposure to any training data is better than random exploration.

In fact, quantity of demonstration may matter more than quality, as shown by the comparison of conditions 1 and 2. Reducing the number of demonstrations by half resulted in a significant decrease in jumpstart. Although performance eventually recovered to achieve a final reward comparable to that of the other methods, achieving that result took longer and there is a statistically significant difference between the total reward of the two conditions.

Most significantly, we highlight that all demonstration-based methods, regardless of data source, quantity or quality, resulted in statistically significant performance improvements over *No Prior* learning. This critical result indicates that HAT learning can benefit from variable degrees of demonstration quality. The algorithm does not require the teacher to be a task expert and easily surpasses the performance of the teacher. In the following section, we further explore the effects of suboptimal demonstrations.

Impact of Teacher Ability on Learning In the above experiments, all three teachers demonstrated the task to their best ability. In this evaluation, we alter the simulation

environment to make the teacher’s demonstrations inherently suboptimal. Specifically, we compare three types of demonstration:

1. *Subject B*: Same as above: average duration 10.5 sec./episode
2. *Subject B Fast*: Simulator speed during training was increased to approximately 5 times faster than real time: average duration 4.3 seconds/episode
3. *Subject B Limited Actions*: The teacher was limited to executing only two actions, `Hold` and `Pass1`, disallowing passes to the further keeper: average duration 5.2 seconds/episode

The two test conditions are designed to handicap the teacher and reduce the quality of demonstrations, either by affecting reaction time (*Subject B Fast*) or by providing the learning agent with demonstrations of only a subset of the state/action space (*Subject B Limited Actions*). The handicapping effects were successful, reducing the average duration of the teacher’s demonstration episodes by more than half.

Figure 5.6 presents a comparison of the three learning conditions and Table 5.4 summarizes the results. Importantly, we see again that poor teacher performance does not negatively impact the final performance of the agent. The data further supports our earlier findings that in the long-term, Probabilistic Policy Reuse can learn the task regardless of the initialization method, and there is no statistically significant difference in final reward values between conditions 1 and 2, and conditions 1 and 3. Statistically significant differences are observed, however, in the rate of learning, both with respect to jumpstart and total reward, indicating that suboptimal demonstrations slow the learning process. However, even with the added handicaps, learning from human data shows statistically significant improvements over *No Prior* learning.

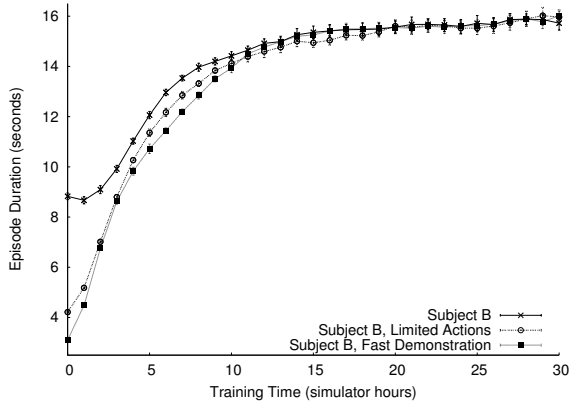


Figure 5.6: This graph summarizes performance of Probabilistic Policy Reuse learning using three sets of demonstrations from Subject B recorded under different simulator conditions: normal, fast and with limited actions. Each demonstration set consists of 20 episodes. Error bars show the standard error in performance across 10 trials.

Method	Jumpstart	Final	Total Reward
Subject B	3.35	15.7	423
Limited Actions	-1.26	16.0	404
Fast Demonstration	-2.37	16.0	401

Table 5.4: This table shows the jumpstart, final reward and total reward metrics for Figure 5.6, where all HAT methods use Probabilistic Policy Reuse. All demonstrations are 20 episodes, recorded by Subject B. Values in **bold** have statistically significant differences in comparison to the No Prior method (not shown).

5.5 Discussion

In this chapter, we introduced HAT, a novel method to combine learning from demonstration with reinforcement learning by leveraging an existing transfer learning algorithm. Using empirical evaluation in the Keepaway domain we showed that given training data from just a few minutes of human demonstration, HAT can increase the learning rate of the task by several simulation hours. We evaluated three different variants which used different methods to bias learning with the human’s demonstration. All three methods performed statistically significantly better than learning without demonstration. Probabilistic Policy Reuse consistently performed at least as well as the other methods, likely because it explicitly balances exploiting the human’s demonstration, exploring, and exploiting the learned policy. Additional evaluation using demons-

trations from different teachers, combined demonstrations from multiple teachers, and suboptimal demonstrations all showed that HAT is robust to variations in data quality and quantity. The best learning performance was achieved by combining the best demonstrations from two teachers.

One of the key strengths of this approach is its robustness. It is able to take data of good or poor quality and use it well without negative effects. This is very important when learning from humans because it can naturally handle the noisy, suboptimal data that usually occurs with human demonstration. Its ability to deal with poor teachers opens up opportunities for non-expert users.

As a future work investigating the agreement between learned rule sets by different teachers more deeply could potentially provide more insight about the demonstrators and which demonstrations can be combined to obtain either a diverse set of demonstrations or a consistent set of demonstrations. The effect of parameters on the learned set of rules is also worth investigating as a future work.

Chapter 6

Learning from Demonstration for Shaping through Inverse Reinforcement Learning

In the previous section we presented HAT, an offline, robust, transfer learning algorithm that improves the learning performance of an RL agent over learning without demonstration. The best performance was achieved by combining the best demonstrations from two teachers. Our evaluation compared algorithmic approaches to incorporating demonstration rule summaries into transfer learning. In this chapter, we introduce two novel RL algorithms that incorporate prior knowledge as reward functions learned from human demonstrations. We show that in two domains, both of our approaches result in significantly (p -values < 0.0001) more cumulative reward and have better asymptotic performance than the human demonstrator.

6.1 Overview

Model-free episodic Reinforcement Learning (RL) problems require agents to explore the state space and iteratively find a solution, which can be costly in terms of learning time. These problems are typically defined by an environment reward that rarely

changes throughout the state-space. For example, in the Cart-Pole domain [95, 96], the environment provides a reward of $r_{env} = +1$ after each step of the task until the terminal state, where the reward is $r_{env} = 0$ to emphasize that the cart has failed at balancing the pole. The information provided by the environment is *sparse* as the change in the reward signal is infrequent. The environment lets the agent know about an obstacle or the terminal state without providing much information right until then. On the one hand, it is easy to define a sparse reward function manually and the interpretation of rewards becomes very intuitive for humans. On the other hand, with this type of environment reward agents are not given enough information about how appropriate the chosen actions are throughout the task. A model-free RL agent can benefit immensely from having a *dense* reward function that provides more information about the task.

An extensively explored way to provide denser information is through *shaping* the existing reward function by adding an extra signal, thus providing the agent with a composite reward [61, 97, 32, 98, 99]. Research in reward shaping shows that potential based shaping functions provide a theoretically sound way of incorporating additional, informative reward functions in existing reinforcement learning problems without altering the problem definition [61]. Potential based shaping rewards can improve an agent's learning efficiency by reducing the number of episodes before convergence [61]. To give an example for the Cart-Pole problem, a potential based shaping reward could be the negative of the angular distance of the pole from the center at each step. There are multiple ways to define a shaping reward function. One way is to use a priori knowledge of the domain characteristics, however, doing so requires extensive knowledge of the state space and becomes infeasible as the number of state variables increases. Recent work has shown that shaping reward can also be effectively extracted from an expert's demonstration by computing the similarity between demonstrated states and observed states throughout learning [32].

Expert demonstrations have also been applied in the context of Markov decision

processes for inverse reinforcement learning (IRL), that is, for extracting a reward function given observed optimal behavior [38]. In domains where the reward function is *unknown*, IRL algorithms are used with the purpose of obtaining a reward function for optimal control.

In this chapter, we introduce Static IRL Shaping (SIS), and Dynamic IRL Shaping (DIS), two variations of a novel approach in which we combine reward shaping and inverse reinforcement learning. SIS and DIS improve upon previous work's use of an expert's demonstrations by capturing the demonstrator's general policy over the state-space for shaping, as opposed to using the observed demonstration samples directly. By using model-free IRL, we encode a demonstrator's task description as a heuristic which *complements* the environment reward function. Our approach consists of three main steps: i) collecting demonstrations, ii) recovering a potential function from demonstrations, and iii) using the recovered potential function for computing shaping rewards in reinforcement learning. Reward functions we recover let us compute a linear potential based shaping reward. It is important to note that, even though we use expert demonstrations in this chapter, there is no strict requirement for the demonstrations to be optimal. We use reward shaping as a heuristic, while the agent still receives the information provided by the environment reward. This means that even if demonstrations are suboptimal, the agent will ultimately be able to learn the task. This is due to the guarantees of policy invariance provided by potential-based reward shaping approaches, and that reward shaping that is not potential-based may alter the optimal policy of the problem. Our goal is to show that using a model-free IRL technique, we can extract additional information for summarizing the task for a standard RL agent and the agent can use this information to learn the task faster. To the best of our knowledge, the use of IRL for potential-based shaping reward functions has not been investigated in the literature.

In order to show that our approach is scalable over different number of state vari-

ables, we evaluate our approach in two domains: a Maze domain and the Mario AI domain. We compare SIS and DIS with the standard SARSA(λ) and Q(λ)-learning agents, as well as with two previously introduced RL algorithms that leverage demonstration data. Our experiment results show that a) both of our approaches result in significantly (p-values < 0.0001) more cumulative reward in both domains compared to other techniques, b) DIS results in faster learning in both domains and c) both approaches have better asymptotic performance than the human demonstrator.

6.2 Methodology

In this section, we present our approach for using a model-free IRL algorithm to obtain a shaping reward and speed up reinforcement learning. Our goal is to incorporate human task demonstration in an intuitive way and incorporate this knowledge in reinforcement learning without changing the task (i.e. the main goal). We aim to reinforce appropriate action selection as soon as an episode starts, much before the agent reaches the terminal state. Each shaping reward signal can be thought of as a *hint* for the agent to make better decisions. To the best of our knowledge, using a potential-based shaping reward function recovered with IRL has not been investigated in the literature. Our method has three main steps:

Collecting Expert Demonstrations: First, we collect observation data from a demonstrator. Each demonstration is a set of sequential observations in time. In this chapter, our focus is on episodic, simulated environments and we collect the agent’s state vector as demonstrations. The demonstrator was an expert teacher, however this is not a requirement for our setup. We assume that the demonstrations are not malicious, that is, the intent of each demonstrated state transition is to perform well on the task. RE-IRL requires trajectories to start from the same initial state and have equal lengths. To equalize the length of demonstrations we process the demonstration data

by repeating the last observation (in our first test domain) or pruning longer demonstrations (in the second test domain). Depending on the task, if demonstrating the goal state carries importance one could prefer to append a repetition of the goal state in the set of demonstrated observations, or otherwise prune the observations. In the end we obtain a set of demonstrations \mathcal{T} . The input for the next step (i.e. RE-IRL) is \mathcal{T} , where each trajectory $\tau \in \mathcal{T}$ is a set of observations of equal length, $\tau = \{s_1, \dots, s_l\}$.

Recovering a Linear Function: Once we obtain \mathcal{T} , we use RE-IRL and compute the vector of reward feature weights \mathbf{w} to use in $R_{IRL}(s)$. RE-IRL recovers a reward function $R_{IRL}(s, a)$ for the set of demonstrations \mathcal{T} as a linear combination of observed reward features $\mathbf{f} = \langle f_1, \dots, f_n \rangle$, weighted with \mathbf{w} . The output is the vector of feature weights $\mathbf{w} = \langle w_1, \dots, w_n \rangle$:

$$R_{IRL}(s) = \sum_{i=1}^n \omega_i f_i(s) \quad (6.1)$$

where, f_i is i th reward feature, ω_i is i th feature weight, s is the state vector for the current state of the agent. Each reward feature is computed from the state vector s and the mapping from state variables to reward features is typically domain dependent, defined by agent designers.

In our representation, we only use the state variables to represent the path of the agent and the policy of the demonstrator, hence the potential function is independent of the actions. RE-IRL obtains the weight vector \mathbf{w} by finding a distribution \mathcal{P} that minimizes the KL-Information loss $I(\mathcal{P}, \mathcal{Q})$ over trajectories.

KL-Information loss $I(\mathcal{P}, \mathcal{Q})$ is the information lost when the distribution \mathcal{Q} is used to approximate the distribution \mathcal{P} [100]. For this work we are interested in transferring the expert demonstration policy from humans to agents, hence we use the distribution \mathcal{Q} as the distribution of observed expert trajectories in state space. RE-IRL tries to find a vector \mathbf{w} that minimizes the KL-Information loss. With \mathbf{w} , during learning, when $\mathbf{f}_{\text{observed}} \approx \mathbf{f}_{\text{demonstration}}$ feature weights return a higher reward to reinforce the

agent’s state-action values.

For the full derivation of the solution we refer readers to [5]. An explanation of RE-IRL with a pseudo-code for the algorithm is also presented by Muelling et al. [50].

Using The Recovered Function: Last, we compute potential-based shaping rewards that the agent receives during learning. For this step we tested the use of the recovered linear function as a traditional, static potential function and also as a dynamic potential function. To implement potential-based shaping we define the above potential function $\Phi : S \mapsto \mathbb{R}$ over the state space, and define the shaping reward F as the difference between the potential of the new state s' and the old state s :

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \tag{6.2}$$

where $0 \leq \gamma \leq 1$ is the discount factor which describes the present value of future rewards and potentials [1]. The effect of shaping is that the agent’s exploration is less random and the agent is biased towards states with high potential. In this work, we use inverse reinforcement learning to obtain a linear function $\Phi(s) = R_{IRL}(s)$ from expert teacher demonstrations. We call this approach Static IRL Shaping (SIS). We name the approach *static* since the potential function does not change over time.

The formulation given in Eq.(6.2) is limited in that the shaping reward is dependent only upon the states encountered by the agent and not its actions. Wiewiora et al. [99] extend the definition of F and Φ to state-action pairs (s, a) as:

$$F(s, a, s', a') = \gamma\Phi(s', a') - \Phi(s, a) \tag{6.3}$$

This formulation allows more information to be incorporated, as it is not limited to states but uses action choices for computing the shaping reward.

Devlin et al. [97] extended Ng’s potential-based reward shaping to dynamic potential-

based shaping, allowing the shaping function to change over time:

$$F(s, t, s', t') = \gamma\Phi(s', t') - \Phi(s, t) \quad (6.4)$$

By changing the shaping function over time, this formulation allows incorporating learning in the shaping function based on agent's experience in order to improve the shaping function. Harutyunyan et al. [98] combine these two extensions into *dynamic shaping* over state-action pairs:

$$F(s, a, t, s', a', t') = \gamma\Phi(s', a', t') - \Phi(s, a, t) \quad (6.5)$$

Importantly, the above variants of shaping all preserve the guarantees such as policy invariance and consistent Nash Equilibria proven by Ng et al [61], and Devlin and Kudenko [97]. Using dynamic shaping, any expert provided reward function R^\dagger can be transformed into a potential-based shaping function by learning a secondary Q-function Φ^\dagger , which serves as the potential function for shaping. The formulation below defines the dynamic potential-based shaping reward. The secondary value function must be learned on-policy, with a technique such as SARSA [101]:

$$\Phi^\dagger(s, a) \leftarrow \Phi^\dagger(s, a) + \beta\delta^{\Phi^\dagger} \quad (6.6)$$

Where β is the learning rate, and δ^{Φ^\dagger} is defined as the temporal difference (TD) error of the state transition:

$$\delta^{\Phi^\dagger} = r^{\phi^\dagger} + \gamma\Phi^\dagger(s', a') - \Phi^\dagger(s, a) \quad (6.7)$$

As the secondary Q-function Φ^\dagger gets updated after each step the shaping function becomes:

$$F = \gamma\Phi^\dagger(s', a') - \Phi^\dagger(s, a) \quad (6.8)$$

By substitution we obtain:

$$\delta_t^{\Phi^\dagger} = r^{\phi^\dagger} + F \quad (6.9)$$

By definition, the temporal difference error (i.e. the error between the estimated Q-value and the actual return) goes to 0 at the time of convergence. Thus for $\delta^{\Phi^\dagger} = 0$:

$$r^{\phi^\dagger} + F = 0 \quad (6.10)$$

And finally at the time of convergence the potential-based shaping function becomes:

$$F(s, a) = R^\dagger(s, a) = -r^{\phi^\dagger} \quad (6.11)$$

When the secondary value function has converged (i.e. when $\delta^{\Phi^\dagger} = 0$), the main value function $Q(s, a)$ will be supplied with a potential-based reward shaping that is equivalent to the reward function R^\dagger . Even before convergence, Φ^\dagger provides useful information about R^\dagger , just like the main Q- function provides information about good policies before converging to an optimal policy. As implied by the last equation above, we learn the secondary Q-function Φ^\dagger using the negative of the expert defined reward $r^{\phi^\dagger} = -R^\dagger$. In our work, $R^\dagger(s, a, s') = R_{IRL}(s)$ is the reward function we recover using inverse reinforcement learning. We call this approach Dynamic IRL Shaping (DIS), as the potential function changes over time based on the learned, secondary Q-function which uses Equation 6.1 as its reward input.

6.3 Experimental Validation

In order to validate our approach we compare the performance of our algorithms against HAT with value-bonus [4] and SBS in two domains: a relatively simple Maze domain, and the Mario AI [102] domain. For both domains, we investigate the initial performance, the asymptotic performance, and the total cumulative reward for evaluation.

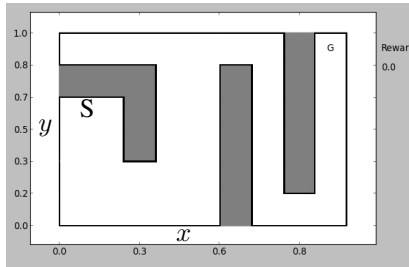


Figure 6.1: Maze domain. The state-space consists of two continuous variables x, y that define the horizontal and vertical location of the agent in the maze with reference to the lower-left corner. The agent starts from S and its goal is to navigate around the gray walls and reach the goal destination G .

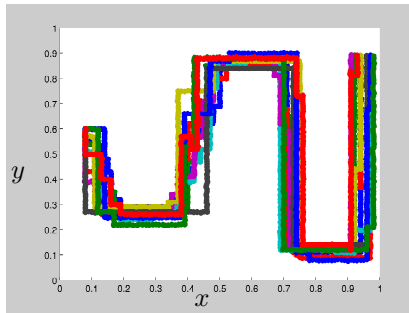


Figure 6.2: Suboptimal demonstrations in the Maze domain.

In both domains we have 5 agents: a standard RL agent, a SIS agent, a DIS agent, a HAT agent and a SBS agent. For the Maze domain we use SARSA(λ) as the standard learner and the underlying learning algorithm for all other agents. For the Mario AI domain we use $Q(\lambda)$ -learning as the standard learner and the underlying learning algorithm for all other agents (with the exception of the secondary learner in DIS, which has to be a SARSA learner).

6.3.1 Maze

The Maze domain shown in Fig.6.1 is a modified version of the Dyna Maze domain introduced by Sutton and Barto [1]. In this domain, the goal of the agent is to start from the position marked with S , navigate around the walls shown as dark gray cells

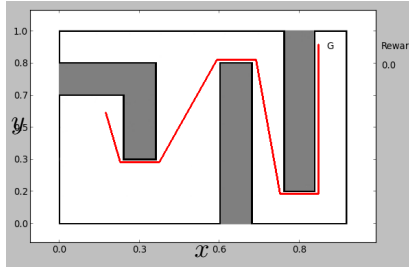


Figure 6.3: Optimal path in the Maze domain.

and reach the goal state marked with G . In our version the state-space consists of two continuous variables $0 \leq x, y < 1.0$. The agent starts at $s = (x : 0.08, y : 0.6)$, and chooses one of four available actions: left, right, up, and down. Step size for each action is 0.02. After each step the agent receives an environment reward of -1 . If the agent tries to get outside the domain boundaries in any direction, or hits a wall, it receives an environment reward of -10 . The terminal condition is $(x \geq 0.9) \ \&\& \ (y \geq 0.9)$ with an environment reward of $+10$. This is an episodic task and each episode starts from the initial state and ends at the goal state (or terminates with failure after 6000 steps).

In order to evaluate our approach we collected 10 suboptimal demonstrations in the Maze domain. The demonstrator was the first author who also designed and implemented the modifications to the original Dyna Maze domain. Fig. 6.2 shows the suboptimal demonstrations. We measured suboptimality based on two metrics: the demonstration path length and the number of actions that did *not* help the agent to navigate toward the goal. Fig. 6.3 shows the optimal, piecewise linear path. The length of the optimal path is 2.665. On average the demonstration paths are 27.4% longer than the optimal path length and on average 1.5% of the performed actions during demonstrations are suboptimal. Features we recorded during demonstrations were the horizontal and vertical location of the agent at each time-step, that is, $s_t(x_t, y_t)$ for each observation s_t . The shortest demonstration took 328 steps to reach the goal and the longest demonstration was 353 steps long ($\mu = 337.5$ and $\sigma = 10.71$ steps). None

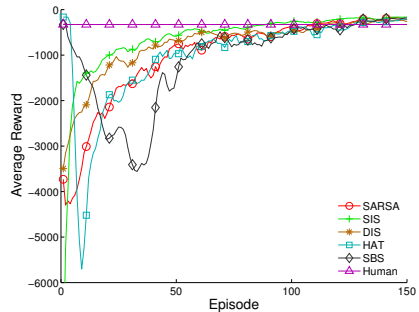


Figure 6.4: Maze domain experiment results.

Table 6.1: Average initial (first episode) performance, asymptotic performance, and cumulative reward with standard deviations for all agents in the Maze domain. Maximum values are shown in bold. Higher values are better. SAR: SARSA, HMN: average of the reward human demonstrator received in 10 demonstrations.

Agents	μ_{init}	μ_{asympt}	μ_{cumul}
SAR	-3,730.36	-222.52	-151,525.96
SIS	-7,973.16	-165.88	-93,591.96
DIS	-3,494.08	-267.88	-113,689.12
HAT	-164.68	-222.36	-150,141.28
SBS	-324.04	-185.84	-159,634.52
HMN	-327.5	-327.5	

of the demonstrations had the agent collide with the walls or the borders of the maze. For RE-IRL, we appended the last observation of the shorter demonstrations repeatedly in order to equalize the lengths of the demonstrations. We used the following parameters for our agents: number of tiles = 32, $\alpha = 0.25/\#tiles$, $\beta = 0.05/\#tiles$, $\gamma = 1.0$, $\lambda = 0.25$, with ϵ -greedy action selection where $\epsilon = 0.02$. Finally, we used a scaling of 100 for all shaping reward signals, except for the HAT agent where the scaling was 10.

Fig. 6.4 shows the agents' performance in 150 episodes, averaged over 25 trials. The vertical axis shows the agents' performance in terms of total reward received per episode and the horizontal axis shows the number of episodes. Each line shows the performance of a different agent. For comparison we also show human demonstrator's performance averaged over 10 demonstrations. Average initial (first episode), asymp-

Table 6.2: Statistical analysis for the Maze domain results show p-values for two-sample t -tests. SAR stands for SARSA. Columns show the statistical significance of the difference between the initial (first episode) performance, asymptotic (last episode) performance, and cumulative reward (i.e. total reward throughout the trial). * $p \leq 0.05$, ** $p \leq 0.01$, *** $p \leq 0.001$, **** $p \leq 0.0001$.

Agents	Initial	Asymp.	Cumulative
SAR vs SIS	3.8929e-16****	0.051603	1.4592e-44****
SAR vs DIS	0.62928	0.56365	8.3354e-33****
SAR vs HAT	1.5995e-13****	0.99673	0.51621
SAR vs SBS	1.6913e-12****	0.26082	0.022387*
DIS vs HAT	3.4262e-13****	0.56402	1.0199e-22****
DIS vs SIS	9.6501e-18****	0.17313	5.2131e-26****
DIS vs SBS	4.1881e-12****	0.28159	5.4586e-18****
SIS vs HAT	3.0772e-94****	0.060007	4.3662e-32****
SIS vs SBS	2.597e-55****	0.32015	9.6945e-25****
SBS vs HAT	0.062916	0.27499	0.015977*

otic (last episode), and cumulative reward are detailed in Table 6.1.

First, we note that all five algorithms are able to achieve a similar asymptotic performance in this domain, although they reach it at different rates. For example, if we consider the number of episodes each algorithm takes to surpass the performance of the demonstrator, then we see that SARSA(λ), SIS, DIS, HAT and SBS require 110, 90, 115, 135, and 136 episodes, respectively. In summary, SIS learns the task remarkably faster than all other methods.

The performance of the different techniques also varies largely during the learning process. The standard SARSA(λ) agent starts from $-3, 730$ and reaches its asymptotic performance after 130 episodes. The SIS approach starts with the worst performance, $-7, 973$, however is the fastest to surpass the demonstrator's performance only after 90 episodes. In comparison the DIS approach starts with an initial performance of $-3, 494$ and learns the task comparably fast.

The HAT algorithm, initializes the Q-function with the potential function at the very beginning. The first column of Table 6.1 shows that the initialization results in high performance during first episodes. However later on, as the agent probabilistically

explores the state-space its performance degrades over time. It is important to point out that the performance boost gained by amount of initial jump-start is dependent of the demonstration quality and demonstrator’s ability. Taylor et al. [4] have investigated the impact of demonstration quality and demonstrator expertise. HAT suffers from a performance loss after about 10 episodes due to the negative step reward. The agent starts with the “good” policy, however receives negative reward at every step and thus starts to explore beyond the good policy. After exploring different states only to learn that they were “bad” states, the agent recovers and improves its performance to catch-up with the learning speed of the standard SARSA(λ) agent. This finding is in parallel with Taylor et al.’s [4] observations, in the sense that human demonstrations do not hurt the agent’s final performance.

Similarly, the SBS approach implements Q-function initialization and starts with a high performance which degrades over time with exploration. After about 136 episodes the SBS agent reaches the demonstrator’s performance and soon after converges to its asymptotic performance of -186 .

Using the rewards agents collected per trial, we performed two-sample t -tests to evaluate whether the agents’ initial performance, asymptotic performance and cumulative reward were significantly different from each other (Table 6.2, sample size: 25 trials).

The difference in asymptotic performance is not significant as can be obvious from Fig. 6.4. That being said, we would like to underline that all methods have surpassed human demonstrator’s average performance. Especially SIS results in about a 50% increase in asymptotic performance after 150 episodes w.r.t the demonstrator’s performance (second column of Table 6.1).

As for the initial performance data, there wasn’t a significant difference between the DIS and the SARSA agents. However, the HAT and the SBS agents performed significantly better than the SARSA agent.

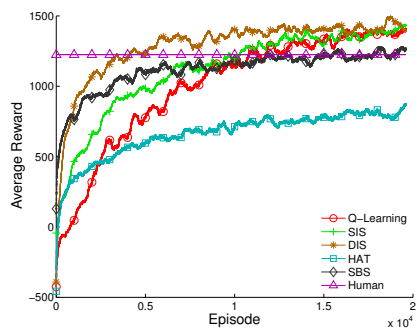


Figure 6.5: Mario domain experiment results.

Table 6.3: Performance metrics for the agents in Mario AI: initial performance, asymptotic performance, cumulative reward (average of 10 trials). Maximum values are shown in bold. Higher values are better.

Agents	μ_{init}	μ_{asympt}	μ_{cumul}
QLR	-424.6	1,410.4155	20,101,973.6
SIS	-43.6	1,431.5328	22,341,947.6
DIS	-391.2	1,398.7714	25,725,731.6
HAT	-457	867.2241	13,250,664.4
SBS	129.8	1,267.8642	22,332,848.2
HMN	1,224	1,224	

Finally, both the SIS and the DIS agents collected significantly more reward throughout 150 episodes (i.e. cumulative reward) than the SARSA, the HAT and the SBS agents.

In summary, the SIS agent shows the steepest learning curve by converging to a stable policy fastest and collects the highest cumulative reward of approximately -93,000 after 150 episodes. The DIS agent performs comparably good with a better initial performance than SIS, however collects less cumulative reward, approximately -113,000.

6.3.2 Mario AI

Super Mario Bros is a popular 2-D side-scrolling video game developed by the Nintendo Corporation. In this work, we use the publicly available Mario AI version released by Karakovskiy and Togelius [102]. There are many possible representations of

Table 6.4: Statistical analysis for the Mario AI domain results show p-values for two-sample t -tests. QLR stands for Q-Learning. Columns show the statistical significance of the difference between the initial performance, asymptotic (final) performance, and cumulative reward (i.e. total reward throughout the trial).

Agents	Initial	Asymp.	Cumulative
QLR vs SIS	0.018698*	0.99448	0.0010416**
QLR vs DIS	0.65987	0.11448	5.5167e-09*****
QLR vs HAT	0.60982	0.71677	9.0692e-12*****
QLR vs SBS	0.050229	0.059223	0.00014223***
DIS vs HAT	0.40846	0.23382	1.6551e-18*****
DIS vs SIS	0.037257*	0.094503	1.9024e-06*****
DIS vs SBS	0.067898	9.1049e-05*****	1.9184e-08*****
SIS vs HAT	0.012506*	0.70797	4.121e-15*****
SIS vs SBS	0.56634	0.040042*	0.98207
SBS vs HAT	0.039996*	0.023199*	4.0285e-21*****

the state-space in this domain and here in order to test our approach in a large state-space, we use 27 discrete state-variables. Boolean state variables *jump*, *ground*, and *shoot* define whether Mario is able to jump, is standing on a tile, or is able to shoot fireballs. We use $x-dir$ and $y-dir$ variables that can take on either one of $\{-1, 0, 1\}$ values to define the direction of the agent’s movements. The immediate surrounding of the agent is discretized in 8 grid cells. For each grid cell we have a state variable *close-enemies* defining whether there is a enemy in the matching cell. We also keep track of the grid cells that are one step farther than the immediate cells, and we have 8 boolean variables called *mid-enemies*. We keep four boolean state variables *obstacle* to identify whether the four vertical cells to the immediate right of Mario are occupied by obstacles such as pipes. Finally we have two variables *closest-enemy-x* and *closest-enemy-y* that define the Euclidean distance between the agent and the closest enemy within a 22x22 grid. This state space is inspired by Liao’s work [103]. We note that this domain is a relatively high dimensional domain compared to most standard RL benchmarking domains. The main reason of our evaluation in the Mario AI domain is to show that our model-free approach is scalable. The agent can choose one of 12 actions combining one of each options of the following three sets: {left, right, no-action}, {jump, do

not jump}, {run, do not run}. The goal of the agent is to maximize the total reward it receives throughout the level. The agent receives a reward of +10 for stepping on an enemy, +58 for eating a mushroom, +64 for eating a fire-flower, +16 for collecting a coin, +24 for finding a hidden block, +1024 for successfully finishing the level, -42 for getting hurt by an enemy (e.g. losing fire-flower), and -512 for dying. The maximum number of steps per episode is limited by the game’s timer and the default timeout is 200 seconds. In the context of potential-based reward shaping, recent work by Harutyunyan et al. [104] implements and evaluates their reward shaping framework [98] in Mario AI domain as well.

We used the 10 demonstrations’ first 953 observations for RE-IRL, and used the same demonstrations in full length for all other methods. For RE-IRL, the 10th longest demonstration was 953 observations long and we pruned the first nine demonstrations at this length to keep the demonstration length consistent. We used the following parameters for our agents: number of tiles = 32, $\alpha = 0.01/\#tiles$, $\beta = 0.05$, $\gamma = 0.9$, $\lambda = 0.5$, with ϵ -greedy action selection where $\epsilon = 0.05$, and $scaling = 1$ for the shaping reward.

We ran 10 trials for all five agents’ performance evaluation. The level our agent played in Mario was designed as an episodic task where the agent finishes a level either with success or the level ends with failure with the agent’s death, or a timeout of 200 seconds. Every episode, the agent plays a randomly generated level, starting from a randomly selected mode (small, large, fire-mario).

Fig. 6.5 shows the agents’ performance in 20,000 episodes, averaged over 10 trials. The vertical axis shows the average total reward (i.e. points) agents collected during each episode. The horizontal axis shows the number of episodes. Each line shows the performance of an agent. Again, for comparison, we also show the human demonstrator’s performance. Table 6.3 shows the average initial performance, a-symptotic performance, and cumulative reward for each agent. On average the demonstrator scored

1224 points in this domain.

Unlike the previous domain not every agent was able to achieve the same asymptotic performance after 20,000. Three agents' asymptotic performance (SIS, DIS, and $Q(\lambda)$) is significantly better than the remaining two (HAT and SBS). Again, if we consider the number of episodes each agent takes to surpass the performance of the user, then we see that $Q(\lambda)$, SIS, DIS, and SBS require 11,000, 10,000, 4,000 and 12,000 episodes respectively. The HAT agent was unable to reach or surpass the demonstrator's performance.

Similar to our previous experiment we observe a great difference in the learning process. With the exception of the SBS agent, most agents start with similar initial performance and negative total reward for their first several hundred episodes. The SBS agent leverages Q-function initialization and starts with a positive initial performance. The initial performance of the SIS agent was significantly better than the standard $Q(\lambda)$ -learning agent, the DIS agent and the HAT agent. That said, the SIS agent's asymptotic performance was only better compared to the SBS and the HAT agents.

In terms of cumulative reward, the SIS agent performed significantly better than all other agents but the DIS agent. Table 6.3 shows that the DIS agent has the best overall cumulative reward performance, exceeding a cumulative reward of 25M in 20,000 episodes. The DIS agent also is the fastest to reach to the demonstrator's performance after about 4,000 episodes, reaching the best asymptotic performance after about 8,000 episodes before any of the other methods even reach the level of the demonstrator.

The SBS algorithm shows quick convergence however its final performance remains limited at about 1,267 points, very close to demonstrator's performance, after 20,000 episodes. This is because SBS takes observed state's similarity with demonstrations and rewards the agent if the agent acts similar to demonstrations. As pointed out by Brys et al. [32] SBS performs better with minimum amount of consistent demonstrations (even with a single demonstration in the case of Cart-Pole domain). Although

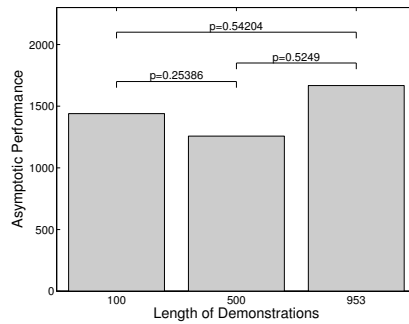


Figure 6.6: Asymptotic performance of the DIS agent after 20,000 episodes in Mario AI. Here the performance is a function of the length of demonstrations (the number of state-action pairs recorded per demonstration). Each condition has 10 demonstrations. P-values obtained with two-sample t -tests between marked conditions for 10 trials. Values for the bar plot are the average of 10 trials.

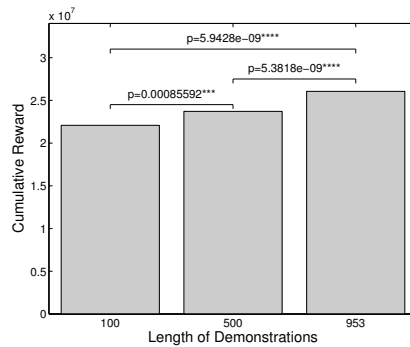


Figure 6.7: Cumulative reward the DIS agent received during 20,000 episodes in Mario AI. Here the cumulative reward is a function of the length of demonstrations (the number of state-action pairs recorded per demonstration). Each condition has 10 demonstrations. P-values obtained with two-sample t -tests between marked conditions for 10 trials.

given enough number of episodes the algorithm is guaranteed to reach the underlying $Q(\lambda)$ -learning's performance, suboptimal demonstrations affect the speed of learning. Finally, the HAT agent performs slightly better than standard $Q(\lambda)$ -learning only for the first 250 episodes. This is because it is difficult to summarize the demonstrator's policy based on a limited number of observations. This shows the slight performance boost that policy transfer brings, however for the rest of the trials, HAT agent shows slow improvement resulting in the significantly worst performance after 20,000 episodes.

After an analysis of learning curves and the agents' performance, we also briefly investigated the effect of the length of demonstrations on the asymptotic performance and the cumulative reward. For this set of experiments, we chose the Mario AI domain as it is a harder task and we used the DIS agent as it was the fastest learner with the highest cumulative reward in our previous experiment.

Fig. 6.6 shows the average asymptotic performance results for 10 trials using 100, 500 and 953 observations for 10 demonstrations. We did *not* find a significant difference between the asymptotic performance data for varying length of demonstrations. We conclude that this is because the DIS agent learns using a secondary Q-function, and even if the demonstrations do not contain enough information, in this case 20,000 episodes were enough for using the environment reward to eventually achieve the same asymptotic performance. However, we did find a significant difference in the cumulative reward of the agents. Fig. 6.7 shows that as the number of observations per demonstration increase, the cumulative reward significantly increases. The results in Fig. 6.6 and Fig. 6.7 imply that the length of demonstrations do affect the learning *speed* without affecting the asymptotic performance of the DIS agent.

6.4 Discussion

In this chapter, we introduced a novel, three step approach to use human demonstrations for potential-based reward shaping in reinforcement learning. Specifically, with our method, we collect demonstrations in a domain, we then use the demonstrations to recover a linear function using inverse reinforcement learning, finally we use this recovered function to compute a shaping reward for potential-based shaping in reinforcement learning. Our approach is model-free and scalable. To show the scalability of our approach we ran two sets of experiments in a simple two dimensional continuous Maze domain, and a complex, 27 dimensional Mario domain with discrete state-variables.

We compared our algorithm to standard RL and two previously introduced rein-

forcement learning from demonstration algorithms:

- HAT algorithm has the ability to learn rules from human demonstrations. In Chapter 5 we showed that for a domain that consists of 13 state variables, rule learning can be effective. In this chapter we found that for a domain that consists of 27 state variables, rule learning may have lost its efficacy. Even though HAT algorithm has the strength of combining demonstrations from multiple demonstrators without negatively affecting the asymptotic performance of the learner, the size of the state space may be a limiting factor for the usability of the algorithm.
- SBS algorithm assumes that each observed demonstration is a desired state-action pair. Even though in optimal demonstration sets this assumption may hold, often in real world domains human demonstrations are suboptimal and noisy. On the other hand, as Brys et al. [32] point out, the SBS algorithm has the ability to bias the reward function with very short demonstrations as the method uses a mixture of Gaussians to measure the similarity of the demonstrated states to the agent's observed state.
- SIS and DIS algorithms need demonstrations that are of equal lengths. Even though there are ways to comply with this requirement, often in real-world domains this can be impractical. In addition to the length of the demonstrations, currently there is no way for the algorithms to assess the fitness of the reward weights that are being recovered from the human demonstrations. As the number of state variables increases, interpreting the effect of reward feature weights on the shaping reward signal becomes infeasible. On the other hand, inverse reinforcement learning methods learn a reward function that represents a task in a very compact manner. This increases the robustness of the algorithm against the noise that suboptimal demonstrations may contain. In contrast, the SBS algorithm would be directly affected of the noise that SIS and DIS algorithms could

potentially filter out by generalizing over the set of demonstrations.

Even though the short list above brings up some of the advantages and usability concerns of these algorithms, a full empirical comparison of these different speed-up methods is beyond the scope of this thesis.

Our experiments show that SIS and DIS agents collect significantly more cumulative reward. Our agents' asymptotic performance surpasses the human demonstrator's average performance. One of our approaches, Static IRL Shaping results in faster learning with higher cumulative reward in the simpler Maze domain, whereas our second approach, Dynamic IRL Shaping results in faster learning with higher cumulative reward in the more complicated Mario AI domains. One possible explanation for this behavior is that the Maze task was not complex enough to observe the benefits of the secondary policy the DIS algorithm learns. This indicates that in simple domains, learning a reward function by demonstrations can be enough to improve a learner's performance compared to previously introduced transfer learning techniques. It is also likely that the high dimensionality and the state representation of the Mario AI domain makes it difficult for the IRL algorithm to recover a suitable reward function for all sections of the state space. The intuition behind the DIS algorithm is that the secondary policy can generalize over the state space and provide a more informative, adaptive shaping reward signal for the agent. Which may explain why the DIS agent performs better in this domain. Finally we also briefly investigated the effect of the number of observations on learning. We found that our approach learns faster as the demonstrations contain more observations as this results in a more complete representation of the task.

Chapter 7

Discussion and Conclusion

This thesis introduced implementations and evaluations of different Reinforcement Learning from Demonstration algorithms, particularly addressing three major challenges: i) enabling RL solvers to incorporate prior knowledge, ii) method of gathering data for the prior particularly from human demonstrators, and iii) the use of the resulting prior by agents. The algorithmic techniques introduced in this thesis leverage human demonstrations in various ways.

The core contributions of this thesis are as follows: First, we introduced the first implementation and evaluation of the previously introduced Interactive Reinforcement Learning algorithm on a robot. With Interactive Reinforcement Learning, a human demonstrator can provide guidance for focusing the action selection of a learner, and rewards for expressing the fitness of an action. Second, we presented quantitative and qualitative findings from a user study where naïve human teachers were asked to teach a real-world task to a humanoid robot with three different Learning from Demonstration algorithms. Next, we introduced a novel Transfer Learning algorithm for RL agents: Human-Agent Transfer (HAT). We investigated three different methods for biasing the agent’s learning process and the effect of the demonstrator’s task expertise on the agent’s learning performance. Finally, we introduced two novel Transfer Learning algorithms, Static Inverse Reinforcement Learning Shaping (SIS) and Dynamic

Inverse Reinforcement Learning Shaping (DIS) where we used human demonstrations for reward shaping with a reward function recovered with a model-free inverse reinforcement learning technique.

We evaluated our algorithms in various real-world and simulated domains, with human teachers with various skills and expertise in each domain. We showed that, for a robot that employs Interactive Reinforcement Learning, an expert human teacher’s guidance can remarkably reduce the number of explored states and the learning time, under the condition that each guidance message restricts the set of choices for action selection to a small number of actions. We also showed that guidance reduces the positive effects of guidance increase with state space size. In the presence of guidance the teacher reduces the number of interactions earlier in the training process and the teacher provides more positive reinforcement signals. With a comparative user study, we showed that, naïve human teachers preferred the simplicity of human-robot interaction that Interactive Reinforcement Learning provides compared to two other Learning from Demonstration algorithms. One critical finding we obtained as a result of the user study was that the duration of the experiment was not enough for users to finish teaching the task through Interactive Reinforcement Learning. This finding highlighted the main motivation of this thesis, which is to improve the learning speed of of RL agents. With the Human-Agent Transfer algorithm, we showed that, an RL agent can robustly use human demonstration data regardless of the expertise of the human teacher in a domain. Finally, with SIS and DIS algorithms, we showed that an RL agent can benefit from a learned human reward function through inverse reinforcement learning.

One important limitation of our work was the limited number of demonstrators we have employed for each evaluation. With the exception of the user study, we were limited with the demonstrations provided by the authors of each work. Even though each author had a different level of expertise in our domains, running wider user studies would be ideal for each algorithm.

Another limitation was the manual selection of features for state representation. Most benchmarking domains, such as the Keepaway domain or the Mario AI domain come with a set of observable features for agents. This set of observable state features does not have to be the exact set of features an agent uses for learning a task.

The choice of free parameters Reinforcement Learning algorithms use, such as the step size and the discount factor, require tuning for each domain. Even though previous work has employed ensemble methods to overcome this limitation, once we have found a satisfactory set of parameters for each agent we have not ran an exhaustive search over the space of parameters for our learners.

Algorithms we have introduced can be improved in many ways. Throughout the work we presented in this thesis one extremely important future work has come up after every evaluation: measuring the quality of human demonstrations. It is not immediately clear how to quantify the quality of demonstrations in a domain-agnostic manner. Ideal set of demonstrations would come from a diverse set of human teachers, categorized consistently –with some variance to represent possible options– by demonstrators’ level of expertise in a domain. Consistency of demonstrations is important, however while being consistent, demonstrations would ideally cover interesting parts of the state space that are rich in information for an agent. Collecting such demonstration data is very difficult and impractical, if not impossible. Often agents have different state representations than a human demonstrators. This difference makes it difficult for human demonstrators to comprehend what they are able to convey to an agent through the demonstrations. One way of helping humans give more informative demonstrations can be through providing feedback and active learning. However the content, the number, the timing and the frequency of queries that would be initiated by an agent is another topic for future work.

Most previous work, including the contribution of this thesis, focus on the algorithmic components and performance of agents, whereas Reinforcement Learning from

Demonstration can be studied from a human-centric perspective as well. Meaningful visualization of a high dimensional state-space, perhaps after dimensionality reduction, can result in improved user interfaces for human-agent interaction. There is a lot of room on research for developing tools and user interfaces for helping humans teach agents tasks, in ways that are most informative for agents.

Finally, many state-of-the-art algorithms have either a two-stage learning process, or an online learning process. A two-stage learning process consists: collecting demonstrations and learning autonomously with the help of demonstrations. An online learning process receives human input during runtime, similar to Interactive Reinforcement Learning and the TAMER framework. What previous work has not investigated is the use of the two-stage learning process repeatedly. An interesting future work in this direction would be the investigation of a learning process where the two stages of learning would be repeated many times, when deemed necessary by the agent or the demonstrator. If this process involves different teachers at some repetitions, as a future work we wish to investigate how demonstrations affect the agent's performance over time.

Bibliography

- [1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
- [2] H. Suay and S. Chernova, “Effect of human guidance and state space size on interactive reinforcement learning,” in *RO-MAN, 2011 IEEE*, 31 2011-aug. 3 2011.
- [3] H. B. Suay, R. Toris, and S. Chernova, “A practical comparison of three robot learning from demonstration algorithm,” *International Journal of Social Robotics*, vol. 4, no. 4, pp. 319–330, 2012.
- [4] M. E. Taylor, H. B. Suay, and S. Chernova, “Integrating reinforcement learning with human demonstrations of varying ability,” in *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, AAMAS '11*, (Richland, SC), pp. 617–624, International Foundation for Autonomous Agents and Multiagent Systems, 2011.
- [5] A. Boularias, J. Kober, and J. Peters, “Relative entropy inverse reinforcement learning,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pp. 182–189, 2011.
- [6] H. B. Suay, T. Brys, M. E. Taylor, and S. Chernova, “Learning from demonstration for shaping through inverse reinforcement learning,” in *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 2016.
- [7] S. Chernova and A. L. Thomaz, “Robot learning from human teachers,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 3, pp. 1–121, 2014.
- [8] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robot. Auton. Syst.*, vol. 57, pp. 469–483, May 2009.
- [9] S. Chernova and M. Veloso, “Interactive policy learning through confidence-based autonomy,” *J. Artificial Intelligence Research*, pp. 1–25, 2009.

- [10] D. H. Grollman and O. C. Jenkins, “Dogged learning for robots,” in *International Conference on Robotics and Automation*, (Rome, Italy), pp. 2483 – 2488, Apr. 2007.
- [11] W. D. Smart, *Making Reinforcement Learning Work on Real Robots*. PhD thesis, Department of Computer Science, Brown University, Providence, RI, 2002.
- [12] M. Nicolescu and M. Mataric, “Learning and interacting in human-robot domains,” *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 31, pp. 419 –430, Sept. 2001.
- [13] U. Nehmzow, O. Akanyeti, C. Weinrich, T. Kyriacou, and S. Billings, “Robot programming by demonstration through system identification,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, 2007.
- [14] A. Ijspeert, J. Nakanishi, and S. Schaal, “Movement imitation with nonlinear dynamical systems in humanoid robots,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02)*, 2002. (Received the ICRA '02 best paper award).
- [15] J. Aleotti and S. Caselli, “Trajectory clustering and stochastic approximation for robot programming by demonstration,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005)*, 2005.
- [16] S. Calinon and A. Billard, “Incremental learning of gestures by imitation in a humanoid robot,” in *Second Annual Conference on Human-Robot Interactions (HRI '07)*, (Arlington, Virginia), March 2007.
- [17] D. C. Bentivegna, A. Ude, C. G. Atkeson, and G. Cheng, “Humanoid robot learning and game playing using PC-based vision,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '02)*, (Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland), October 2002.
- [18] A. Chella, H. Dindo, and I. Infantino, “A cognitive framework for imitation learning,” *Robotics and Autonomous Systems, special issue on The Social Mechanisms of Robot Programming by Demonstration*, vol. 54, no. 5, pp. 403–408, 2006.
- [19] R. M. Voyles and P. K. Khosla, “A multi-agent system for programming robots by human demonstration,” *Integrated Computer-Aided Engineering*, vol. 8, no. 1, pp. 59–67, 2001.
- [20] M. J. Mataric, “Sensory-motor primitives as a basis for learning by imitation: Linking perception to action and biology to robotics,” in *Imitation in Animals and Artifacts* (K. Dautenhahn and C. Nehaniv, eds.), pp. 392–422, MIT Press, 2002.
- [21] C. G. Atkeson and S. Schaal, “Robot learning from demonstration,” in *ICML*, vol. 97, pp. 12–20, 1997.

- [22] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of the 21st International Conference on Machine Learning (ICML '04)*, (Baniff, Canada), 2004.
- [23] M. van Lent and J. E. Laird, “Learning procedural knowledge through observation,” in *K-CAP '01: Proceedings of the 1st international conference on Knowledge capture*, (New York, NY, USA), pp. 179–186, ACM Press, 2001.
- [24] A. L. Thomaz and C. Breazeal, “Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance,” in *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 2006.
- [25] M. N. Nicolescu and M. J. Matarić, “A hierarchical architecture for behavior-based robots,” in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, AAMAS '02, (New York, NY, USA), pp. 227–233, ACM, 2002.
- [26] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.
- [27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, London, England: The MIT Press, 1998.
- [28] G. A. Rummery and M. Niranjan, “On-line q-learning using connectionist systems,” Technical Report CUED/F-INFENG-RT 116, University of Cambridge, Department of Engineering,, 1994.
- [29] S. P. Singh and R. S. Sutton, “Reinforcement learning with replacing eligibility traces,” *Machine learning*, vol. 22, no. 1-3, pp. 123–158, 1996.
- [30] W. D. Smart and L. P. Kaelbling, “Effective reinforcement learning for mobile robots,” in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 4, pp. 3404–3410, IEEE, 2002.
- [31] M. P. Deisenroth, “A Survey on Policy Search for Robotics,” *Foundations and Trends in Robotics*, vol. 2, pp. 1–142, 2011.
- [32] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, “Reinforcement learning from demonstration through shaping,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.
- [33] D. H. Grollman and O. C. Jenkins, “Dogged learning for robots,” in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 2483–2488, IEEE, 2007.
- [34] M. Nicolescu, O. Chadwicke Jenkins, A. Olenderski, and E. Fritzing, “Learning behavior fusion from demonstration,” *Interaction Studies*, vol. 9, no. 2, pp. 319–352, 2008.

- [35] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, “Autonomous inverted helicopter flight via reinforcement learning,” in *International Symposium on Experimental Robotics*, vol. 21, pp. 363–372, 2004.
- [36] M. Saggarr, T. DSilva, N. Kohl, and P. Stone, “Autonomous learning of stable quadruped locomotion,” in *RoboCup 2006: Robot Soccer World Cup X*, pp. 98–109, Springer, 2007.
- [37] R. E. Kalman, “When is a linear control system optimal?,” *Journal of Fluids Engineering*, vol. 86, no. 1, pp. 51–60, 1964.
- [38] A. Y. Ng and S. J. Russell, “Algorithms for Inverse Reinforcement Learning,” in *in Proc. 17th International Conf. on Machine Learning*, pp. 663–670, 2000.
- [39] P. Abbeel, “Inverse Reinforcement Learning.”
- [40] N. Ratliff, J. Bagnell, and M. Zinkevich, “Maximum margin planning,” *Proceedings of the 23rd International Conference on Machine Learning*, 2006.
- [41] N. Ratliff, D. Bradley, J. A. Bagnell, and J. Chestnutt, “Boosting structured prediction for imitation learning,” in *Advances in Neural Information Processing Systems 19*, (Cambridge, MA), MIT Press, 2007.
- [42] J. Z. Kolter, P. Abbeel, and A. Y. Ng, “Hierarchical apprenticeship learning with application to quadruped locomotion,” in *Advances in Neural Information Processing Systems*, pp. 769–776, 2007.
- [43] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum Entropy Inverse Reinforcement Learning,,” in *AAAI*, pp. 1433–1438, 2008.
- [44] U. Syed and R. E. Schapire, “A game-theoretic approach to apprenticeship learning,” in *Advances in Neural Information Processing Systems*, pp. 1449–1456, 2007.
- [45] P. Abbeel, D. Dolgov, A. Y. Ng, and S. Thrun, “Apprenticeship learning for motion planning with application to parking lot navigation,” in *International Conference on Intelligent Robots and Systems*, pp. 1083–1090, 2008.
- [46] D. Ramachandran and E. Amir, “Bayesian Inverse Reinforcement Learning,” in *International Joint Conference on Artificial Intelligence*, vol. 51, pp. 2586–2591, 2007.
- [47] C. L. Baker, R. Saxe, and J. B. Tenenbaum, “Action understanding as inverse planning,” *Cognition*, vol. 113, no. 3, pp. 329–349, 2009.
- [48] G. Neu and C. Szepesvari, “Apprenticeship Learning using Inverse Reinforcement Learning and Gradient Methods,” in *Proceedings of The 23rd Conference on Uncertainty in Artificial Intelligence*, pp. 295–302, 2012.

- [49] E. Klein, B. Piot, M. Geist, and O. Pietquin, “A cascaded supervised learning approach to inverse reinforcement learning,” in *Machine Learning and Knowledge Discovery in Databases*, (Berlin, Heidelberg), pp. 1–16, 2013.
- [50] K. Muelling, A. Boularias, B. Mohler, B. Schölkopf, and J. Peters, “Learning strategies in table tennis using inverse reinforcement learning.,” *Biological cybernetics*, vol. 108, pp. 603–19, Oct. 2014.
- [51] M. E. Taylor and P. Stone, “Transfer learning for reinforcement learning domains: A survey,” *The Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.
- [52] M. E. Taylor and P. Stone, “Cross-domain transfer for reinforcement learning,” in *Proceedings of the 24th international conference on Machine learning*, pp. 879–886, ACM, 2007.
- [53] M. E. Taylor, P. Stone, and Y. Liu, “Transfer learning via inter-task mappings for temporal difference learning.,” *Journal of Machine Learning Research*, vol. 8, no. 1, pp. 2125–2167, 2007.
- [54] L. Torrey, T. Walker, J. Shavlik, and R. Maclin, “Using advice to transfer knowledge acquired in one reinforcement learning task to another,” in *Machine Learning: ECML 2005*, pp. 412–424, Springer, 2005.
- [55] W. B. Knox and P. Stone, “Combining manual feedback with subsequent mdp reward signals for reinforcement learning,” in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 5–12, International Foundation for Autonomous Agents and Multiagent Systems, 2010.
- [56] W. B. Knox and P. Stone, “Interactively shaping agents via human reinforcement: The tamer framework,” in *Proceedings of the fifth international conference on Knowledge capture*, pp. 9–16, ACM, 2009.
- [57] B. Price and C. Boutilier, “Accelerating reinforcement learning through implicit imitation,” *Journal of Artificial Intelligence Research*, pp. 569–629, 2003.
- [58] R. Maclin and J. W. Shavlik, “Creating advice-taking reinforcement learners,” *Machine Learning*, vol. 22, no. 1-3, pp. 251–281, 1996.
- [59] G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik, “Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer,” in *The AAAI-2004 workshop on supervisory control of learning and adaptive systems*, 2004.
- [60] B. F. Skinner, *The behavior of organisms: An experimental analysis*. Appleton-Century, 1938.

- [61] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in *In Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287, Morgan Kaufmann, 1999.
- [62] S. Schaal, “Learning from demonstration,” in *Advances in Neural Information Processing Systems 9*, pp. 1040–1046, MIT Press, 1997.
- [63] A. Thomaz, G. Hoffman, and C. Breazeal, “Reinforcement learning with human teachers: Understanding how people want to teach robots,” in *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pp. 352–357, Sept. 2006.
- [64] D. Gu, R. J. Howlett, and Y. Liu, *Robot Intelligence: An Advanced Knowledge Processing Approach*. Springer Publishing Company, Incorporated, 1st ed., 2010.
- [65] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. Intelligent robotics and autonomous agents, MIT Press, Sept. 2005.
- [66] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, London, England: The MIT Press, 1998.
- [67] S. Calinon, *Robot Programming by Demonstration: A Probabilistic Approach*. EPFL/CRC Press, 2009.
- [68] F. Kaplan, P. Y. Oudeyer, E. Kubinyi, and . Miklsi, “Robotic clicker training,” *Robotics and Autonomous Systems*, vol. 38, no. 3-4, pp. 197–206, 2002.
- [69] W. B. Knox and P. Stone, “Interactively shaping agents via human reinforcement: The TAMER framework,” in *The Fifth International Conference on Knowledge Capture*, September 2009.
- [70] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robot. Auton. Syst.*, vol. 57, pp. 469–483, May 2009.
- [71] T. T. Herbert Bay, Andreas Ess and L. V. Gool, “Surf: Speeded up robust features,” *Computer Vision and Image Understanding (CVIU)*, vol. 110, no. 3, pp. 346–359, 2008.
- [72] R. E. W. Rafael C. Gonzalez and S. L. Eddins, *Digital Image Processing using Matlab*. Prentice Hall, 2003.
- [73] G. Bradski and A. Kaehler, *Learning OpenCV*. O’Reilly Media, 2008.
- [74] W. D. Smart and L. P. Kaelbling, “Practical reinforcement learning in continuous spaces,” in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pp. 903–910, Morgan Kaufmann, 2000.

- [75] W. T. B. Uther and M. M. Veloso, “Tree based discretization for continuous state space reinforcement learning,” in *Proceedings of American Association for Artificial Intelligence(AAAI)*, 1998.
- [76] R. Loftin, J. MacGlashan, B. Peng, M. Taylor, M. Littman, J. Huang, and D. Roberts, “A strategy aware technique for learning behaviors from discrete human feedback,” in *AAAI Conference on Artificial Intelligence*, 2014.
- [77] W. B. Knox and P. Stone, “Combining manual feedback with subsequent mdp reward signals for reinforcement learning,” in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2010.
- [78] M. N. Nicolescu and M. J. Mataric, “Natural methods for robot task learning: instructive demonstrations, generalization and practice,” in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS ’03, (New York, NY, USA), pp. 241–248, ACM, 2003.
- [79] B. Akgun, M. Cakmak, J. W. Yoo, and A. Thomaz, “Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective,” in *Proceedings of the International Conference on Human-Robot Interaction (HRI)*, 2012.
- [80] H. Suay and S. Chernova, “A comparison of two algorithms for robot learning from demonstration,” in *IEEE International Conference on Systems, Man, and Cybernetics*, 2011.
- [81] S. Chernova and M. Veloso, “Confidence-based learning from demonstration using Gaussian Mixture Models,” in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AMMAS ’07)*, May 2007.
- [82] M. Cakmak, C. Chao, and A. Thomaz, “Designing interactions for robot active learners,” *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 108–118, 2010.
- [83] A. L. Thomaz and C. Breazeal., “Asymmetric interpretations of positive and negative human feedback for a social learning agent,” in *Proceedings of the 16th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2007.
- [84] A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.
- [85] O. G. Selfridge, R. S. Sutton, and A. G. Barto, “Training and tracking in robotics.,” in *IJCAI*, pp. 670–672, 1985.
- [86] F. Fernández and M. Veloso, “Probabilistic policy reuse in a reinforcement learning agent,” in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 720–727, ACM, 2006.
- [87] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, “Multi-task reinforcement learning: a hierarchical bayesian approach,” in *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022, ACM, 2007.

- [88] L. Torrey, J. Shavlik, T. Walker, and R. Maclin, “Relational macros for transfer in reinforcement learning,” in *Inductive Logic Programming*, pp. 254–268, Springer, 2008.
- [89] E. Wiewiora, “Potential-based shaping and q-value initialization are equivalent,” *J. Artif. Intell. Res.(JAIR)*, vol. 19, pp. 205–208, 2003.
- [90] F. Fernández, J. García, and M. Veloso, “Probabilistic policy reuse for inter-task transfer learning,” *Robotics and Autonomous Systems*, vol. 58, no. 7, pp. 866–871, 2010.
- [91] P. Stone, R. S. Sutton, and G. Kuhlmann, “Reinforcement learning for robocup soccer keepaway,” *Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, 2005.
- [92] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu, “Keepaway soccer: From machine learning testbed to benchmark,” in *RoboCup 2005: Robot Soccer World Cup IX*, pp. 93–105, Springer, 2006.
- [93] I. Noda, H. Matsubara, K. Hiraki, and I. Frank, “Soccer server: A tool for research on multiagent systems,” *Applied Artificial Intelligence*, vol. 12, no. 2-3, pp. 233–250, 1998.
- [94] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.
- [95] A. Barto, R. Sutton, and C. Anderson, “Neuronlike adaptive elements that can solve difficult learning control problems,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-13, no. 5, pp. 834–846, 1983.
- [96] R. S. Sutton, *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts - Amherst, 1984.
- [97] S. Devlin and D. Kudenko, “Dynamic potential-based reward shaping,” in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 433–440, International Foundation for Autonomous Agents and Multiagent Systems, 2012.
- [98] A. Harutyunyan, S. Devlin, P. Vrancx, and A. Nowe, “Expressing arbitrary reward functions as potential-based advice,” in *AAAI Conference on Artificial Intelligence*, 2015.
- [99] E. Wiewiora, G. W. Cottrell, and C. Elkan, “Principled methods for advising reinforcement learning agents,” in *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pp. 792–799, 2003.
- [100] K. P. Burnham and D. R. Anderson, *Model selection and multimodel inference: a practical information-theoretic approach*. Springer, 2002.
- [101] G. A. Rummery and M. Niranjan, “On-line q-learning using connectionist systems,” tech. rep., University of Cambridge, Department of Engineering, 1994.

- [102] S. Karakovskiy and J. Togelius, “The mario ai benchmark and competitions,” *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 55–67, 2012.
- [103] Y. Liao, K. Yi, and Z. Yang, “Cs229 final report reinforcement learning to play mario,” tech. rep., Stanford University, 2012.
- [104] A. Harutyunyan, T. Brys, P. Vrancx, and A. Nowé, “Shaping mario with human advice,” in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS ’15*, (Richland, SC), pp. 1913–1914, 2015.