HELP - HUMAN ASSISTED EFFICIENT LEARNING PROTOCOLS

BY KAUSHIK SUBRAMANIAN

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Zoran Gajic

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2010

ABSTRACT OF THE THESIS

HELP - Human assisted Efficient Learning Protocols

by Kaushik Subramanian

Thesis Director: Prof. Zoran Gajic

In recent years, there has been a growing attention towards the development of artificial agents that can naturally communicate and interact with humans. The focus has primarily been on creating systems that have the ability to unify advanced learning algorithms along with various natural forms of human interaction (like providing advice, guidance, motivation, punishment, etc). However, despite the progress made, interactive systems are still directed towards researchers and scientists and consequently the everyday human is unable to exploit the potential of these systems. Another undesirable component is that in most cases, the interacting human is required to communicate with the artificial agent a large number of times, making the human often fatigued. In order to improve these systems, this thesis extends prior work and introduces novel approaches via Human-assisted Efficient Learning Protocols (HELP).

Three case studies are presented that detail distinct aspects of HELP - a) representation of the task to be learned and its associated constraints, b) the efficiency of the learning algorithm used by the artificial agent and c) the unexplored "natural" modes of human interaction. The case studies will show how an artificial agent is able to efficiently learn and perform complex tasks using only a limited number of interactions with a human. Each of these studies involves humans subjects interacting with a real robot and/or simulated agent to learn a particular task. The focus of HELP is to show

that a machine can learn better from humans if it is given the ability to take advantage of the knowledge provided by interacting with a human partner or teacher.

Acknowledgements

I would like to thank Prof. Michael Littman for all the support he has given me throughout the entire duration of my graduate program. From the very beginning, he gave me the freedom to work in the field of my interest and simultaneously ensured that I was always on the right path. I have learnt invaluable lessons during the course of my interactions with him.

I would like to thank the members of the RL^3 Lab at Rutgers University for all their support and guidance during the course of my research. In particular I would like to thank Tom and Monica. Their constant support and enormous patience to deal with many of my doubts and questions have set a strong foundation for this work. Working with them has been a great learning experience for me.

I would also like to thank Prof. Gerhard Lakemeyer and the members of the KBSG Lab in RWTH Aachen University, Germany. The experience I gained during my visit there played a pivotal role in building my potential for research.

Dedication

This thesis is dedicated to my parents, my sister and my grandparents for their invaluable support and encouragement.

Table of Contents

Abstract ii

Acknowledgements iv

Dedication v

List of Figures ix

1. Introduction 1

1.1. The Need for Human Guidance 2

1.2. Survey of Learning Methods and Modes of Human Interactions 3

1.2.1. Models of Human Learning 3

1.2.2. Machine Learning with a Human 5

1.2.3. Current Approaches Tailored for Human Input 7

1.2.3.1. By observing human behavior 7

1.2.3.2. Providing labels and feedback 9

1.3. Thesis Overview 11

2. Experiments in HELP 12

2.1. Case Study 1 - Learning by Demonstration by Supervised Learning . . . 12

2.1.1. Learning Machine - Nao Humanoid Robot 13

2.1.2. Demonstration Method - Kinesthetic Teaching 14

2.1.3. Learning Algorithm - Gaussian Mixture Regression 16

2.1.3.1. Task Constraints 16

2.1.3.2. Gaussian Mixture Models 17

2.1.3.3. Trajectory Reproduction 20

2.1.4. Experiments and Results 20

2.1.5. Experimental Observations 25
2.1.6. Limitations 27
2.1.7. Summary 27

3. Experiments in HELP 29
3.1. Case Study 2 - A Novel Approach to Learning by Demonstration using Reinforcement Learning 29
3.1.1. Reinforcement Learning and Teachers 30
3.2. The Apprenticeship Learning Protocol 31
3.2.1. A Generalized Approach to Apprenticeship Learning 32
3.3. Sample Efficiency in Apprenticeship Learning 35
3.3.1. Model Learning Frameworks 35
3.3.1.1. KWIK - Knows What It Knows 36
3.3.1.2. MB - Mistake Bound 37
3.3.1.3. The Mistake Bounded Predictor Framework 38
3.3.2. Efficient Apprenticeship Learning 40
3.3.3. Learning Conjunctions 41
3.4. Experiments and Results 42
3.4.1. Simulated Taxi World 42
3.4.2. Robot Taxi 43
3.5. Conclusion 45

4. Experiments in HELP 47
4.1. Case Study 3 - Extending the Modes of Human Interaction 47
4.2. Experimental Setup 48
4.2.1. The Highway Car Domain 49
4.2.2. Learning Algorithms 50
4.2.2.1. Q-learning 50
4.2.2.2. R-Max 51
4.2.3. Forms of Human Input 52

4.3. Driving Instructor and the Various Modes of Interaction 54
4.3.1. The Motivational Instructor 54
4.3.1.1. Effect on the Learners 55
4.3.1.2. Effect on the Instructor 57
4.3.2. The Directive Instructor 58
4.3.2.1. Effect on the Learners 59
4.3.2.2. Effect on the Instructor 61
4.3.3. The Associative Instructor 61
4.3.3.1. Effect on the Learners 63
4.3.3.2. Effect on the Instructor 65
4.3.4. Which Agent?, Which Instructor? 65

5. Conclusions and Future Work 66

List of Figures

2.1. The Aldebaran's Nao Humanoid Robot and its Degrees of Freedom . . . 14

2.2. Screenshot of *Fawkes* GUI to enable plugins and control stiffness 15

2.3. Mouse Task - Mouse starting from random position has go to the File and move it to the Bin 16

2.4. A set of 4 demonstrations of the Mouse Task with the highlighted positions of the Mouse, File and Bin. 17

2.5. Task Constraints for the Mouse Task 17

2.6. Extracting the Mean, Covariance, Priors (GMM Parameters) from the Task Constraints 18

2.7. Task Constraints on the left have been generalized and represented as gaussian models on the right side 19

2.8. Regressed model of the generalized data 19

2.9. Gaussian Mixture Regressed representations with respect to system objects 20

2.10. This experiment contains a set of 3 demonstrations of the mouse task. The red line indicates the reproduced trajectory for new positions of the mouse, file and bin (indicated by the + signs). 21

2.11. Results for a mouse task with 4 demonstrations and new starting positions. 21

2.12. Shows the adaptability of the system to changes in the position of the file during execution of the trajectory. 22

2.13. By changing the position of the bin in the very last time step, the system is able to account for the new position. 22

2.14. A 2D obstacle avoidance task result 23

2.15. A 2D obstacle avoidance task result 23

2.16. Different positions of the object used for training and testing 24

2.17. A 3D demonstration using the right arm of the Nao to perform the object to goal task. 24
2.18. Obstacle Avoidance for the Nao after training it for 5,8,12 and 15cm heights. It was tested with 10cm 25
2.19. Perform GMM using three components. 26
2.20. Perform GMM using two components. 26
2.21. Three sets of demonstrations with one quite different from the remaining two. 26
3.1. Robot tracing a path through the obstacles to get the key that opens the door to the goal. 30
3.2. The Apprenticeship Learning Protocol by Pieter Abbeel and Andrew Ng. 32
3.3. The Generalized Apprenticeship Learning Protocol. 33
3.4. The Knows What It Knows Protocol. 36
3.5. The Mistake Bound Protocol. 37
3.6. The Mistake Bounded Predictor Protocol. 39
3.7. The Taxi Domain. 42
3.8. A KWIK learner (autonomous) and an MBP learner (apprenticeship) in the Taxi Domain 43
3.9. The taxi robot, its passenger and the goal. 44
4.1. Traditional RL Framework. 48
4.2. Screenshot of the Driving Simulator. 49
4.3. Bridging the communication gap between the instructor and artificial agent. 53
4.4. Different Configurations of the Driving Domain and the Reinforcement given by the Instructor. 55
4.5. Number of steps taken by the learners to reach the optimal policy (with Motivational Instructor). 56
4.6. Performance of the agent across 5 trials with 2 Motivational human subjects. 57

4.7. Different Configurations of the Driving Domain and the Directive Actions given by the Instructor. 59
4.8. Number of steps taken by the learners to reach the optimal policy (with Directional Instructor). 61
4.9. Different Configurations of the Driving Domain and the State Mapping given by the Instructor. 62
4.10. Number of steps taken by the learners to reach the optimal policy (Associative Instructor). 64

Chapter 1

Introduction

We have always imagined that one day robots would live among us and learn to adapt and behave in a manner similar to humans. This use of robots in everyday human environments has long been a goal for researchers and scientists all over the world. In order for this dream to be realized, robots must be able to interact, cooperate and coexist with humans. At present there are several academic groups that are actively at work towards this goal. For example, using robots to aid humans in household activities, to work alongside doctors and perform precise surgical operations, to assist astronauts in space. In each of the examples mentioned, it is the elements of human-robot interaction that have played a pivotal role for their success. In more specific terms, it is the ability of a robot to effectively interact and learn from a human that has lead to the successful application of robots in everyday human environments. This form of communication (human and robot) helps alleviate several of the problems faced by autonomous agents in similar task environments. For example, in this ever-changing world, autonomous agents must consistently and reliably adapt their behavior to achieve reasonable performance levels. In effect, their response time should match the rate of change of the nature of the environment in which they are acting. A key problem that is often faced with human-robot interaction protocols is that the underlying learning technique (used by the artificial agent) has not been designed for learning from non-expert users. This research is based on the belief that machines, designed to interact with people, should use behaviors that are relevant to the way in which humans naturally interact. This ability to utilize human knowledge will serve as a good interface for people as well as have a positive impact on the underlying learning mechanism.

This thesis discusses about Human-assisted Efficient Learning Protocols (HELP),

examining ways in which learning algorithms can efficiently utilize the knowledge that humans can provide. The features of HELP are highlighted in three case studies, involving human subjects, dealing with Supervised Learning and Reinforcement Learning (RL) algorithms. In each of the studies, software/robotic implementations and experiments are used to explore and validate the advantages of HELP. This work demonstrates the performance benefits of integrating human interactions with the learning process.

In this chapter, we will begin by motivating the need for human guidance in robot learning tasks, then provide a brief summary on the current models used for human learning and for human-robot interactions.

1.1 The Need for Human Guidance

In order to motivate the need for human guidance, let us briefly outline the approaches that researchers have taken in bringing artificial agents and humans together. In the 1940s Grey Walter, as described in Holland (2003), built a robot capable of interacting with another in a seemingly "social" manner, although there was no explicit communication or mutual recognition between them. Some years later, Beckers et al. (1994) pioneered the first experiments on stigmergy (an indirect communication protocol between individuals via modifications made to the shared environment) in simulated and physical "ant-like robots". Following these works and many others, there has been a continued interest in this field and research has advanced a great deal over the decades. The most recent work has been done by Cynthia Breazeal and Andrea Thomaz. They introduced a framework known as *Socially Guided Machine Learning* [Breazeal and Thomaz (2008); Thomaz et al. (2006)] that is aimed at designing robots that learn by taking advantage of the social protocols with which humans interact with one another. Such robots are referred to as social robots and they have a wide range of applications in toys, as educational tools, as industry assistants, aid in surgical operations etc. A survey of the current applications of such robots is given in Fong et al. (2003) where Fong et al. analyze the aspects that need to be taken into account when designing a social robot.

Let us try to understand further what is it that humans expect robots to learn through their interactions. Consider the example of a humanoid robot that has been assigned the role of a household assistant. The robot must understand human social protocols, it must understand speech commands and extrapolate the intended emotions, the robot must remember, adapt and learn to perform new tasks, the robot should perceive the surroundings almost as well as humans, etc. This provides a broad perspective of some of the expected outcomes of human-robot interactions. Humans through the means of advice, guidance, motivation and correction can help the robots efficiently explore the world, adapt to dynamic environments, learn to perform new tasks, perceive the world in manner suitable for human interaction and so on. This research focuses on the efficient utilization of human task knowledge to enhance the learning process of an artificial agent.

1.2 Survey of Learning Methods and Modes of Human Interactions

In this section we briefly overview computational models that describe human learning, the role of humans in standard machine learning algorithms and a survey on the current machine learning approaches suited for human inputs.

1.2.1 Models of Human Learning

A child typically says its first words and takes its first steps in about a year after birth. It is quite remarkable that just by the age of five, a child can understand spoken language, distinguish a cat from a dog, and play a game of catch. Clearly the human child is the best learning system on the planet. Charles Kemp [Kemp et al. (2007)] explains that the key advantages of human learning are that it is guided by rich prior knowledge, takes place at multiple levels of abstraction, and allows structured representations to be acquired. Researchers are currently focusing on accurately building on such models of human learning so that they can be extended to Machine Learning domains. Machine Learning is a field that focuses on the ability to make artificial agents/machines efficiently learn to performs tasks in this complex and noisy world. It has been explained

in more detail in the next section.

Michael J. Pazzani [Pazzani (1994)] also believed that human learners are the closest approximations to general purpose learning machines that are available for study. He analyzed computational models by which people can acquire new skills without interfering with existing skills, can learn from incomplete and contradictory information, can use existing knowledge to aid the learning process, can learn in the absence of relevant background knowledge, can learn from complex, high dimensional visual and auditory data, etc.

In order to understand human learning models, a vast majority of research has been done by studying how children learn from a very young age. Some of the early work in this field include (also discussed in Pazzani (1994)) -

- Jones and VanLehn (1991) investigate a model that accounts for data on how young children learn to add.
- Kazman (1991) proposes a model of the child acquiring lexical and syntactic knowledge by being exposed to samples of adult's language usage.
- Martin and Billman (1991) investigate how a learner may acquire overlapping concepts from unsupervised or unlabeled data.
- Seifert et al. (1994) propose a model of how experiences are stored in memory so that they may be retrieved in appropriate situations.
- Shultz et al. (1994) explore how children learn to predict what will occur when weights are placed on a balance scale.

At this point, we ask ourselves, can machine learning algorithms help explain human learning? It has long been understood that the theories and algorithms from machine learning are relevant to understanding aspects of human learning. The vice-versa also holds true. Human cognition also carries potential lessons for machine learning research, since people still learn languages, concepts, and causal relationships from far less data than any automated system. There is a rich opportunity to develop a general theory of learning which covers both machines and humans, with the potential to deepen

our understanding of human cognition and to take insights from human learning to improve machine learning systems. Several machine learning algorithms have been built around human models of learning, including neural networks, case-based reasoning, probabilistic models and statistical induction. We now discuss approaches that combine the human learning system with a machine learner.

1.2.2 Machine Learning with a Human

Machine Learning has evolved from the broad field of Artificial Intelligence, which aims to mimic the intelligent abilities of humans by machines. In the field of Machine Learning one considers the important question of how to enable machines to "learn". Machine Learning can be divided into three categories, namely Supervised, Unsupervised and Reinforcement Learning. In supervised learning, there is a label associated with an example from a data-set. The label is supposed to be the "answer" to a question about the example. If the label is discrete, then the task is called a classification problem otherwise, for real-valued labels it is a regression problem. In unsupervised learning, the task is to uncover hidden regularities (e.g. clusters) or to detect anomalies in the data. In reinforcement learning, a task is characterized by a reward function and the agent must learn to act in the world such that it maximizes the cumulative reward received. Further details about some of these approaches are provided in the case studies.

There has been great success for standard machine learning techniques in a wide variety of applications. It has been applied to several hard problems that involve learning in real-time in environments that are continuous, stochastic and partially observable [Thrun and Mitchell (1995); Mataric (1997)]. In typical situations machine learning has not been designed for learning from a human teacher, but in some way or the other a human is part of the learning process. The traditional role of the human operator in machine learning problems is that of a batch labeler or task encoder, whose work in some cases is done before the learning even begins. In other cases, the human is interacting directly with the learning algorithm as it learns. Frequent problem scenarios which fall into this space include active learning, interactive clustering, query by selection, learning to rank, and others. Human designers are used to create successful

learning systems in the following manner:

- Build the data set on which Machine Learning is performed -

  The designer must collect training and testing data that is representative of the specific task. The quality and quantity of the data will determine the accuracy and the generalization characteristics of the resulting system.

- Provide Labels -

  Some methods of supervised learning like active learning, case based reasoning, inductive learning and so on require inputs from the human during training, in the form of labels and feedback. Providing these true labels allows the underlying algorithm to learn the characteristics of the data that are common to similarly labeled examples and this way it is able to generalize.

- Construct Features and Algorithm Parameters -

  The designer is often given the task of constructing features that can be utilized by learning algorithms. In supervised learning, the human uses his/her prior knowledge about the data and appropriately chooses features that will help in the classification. In reinforcement learning, feature selection has a great impact on how the agent will model the environment. Humans go through several iterations of testing various algorithm parameters, in search for the optimal set that yields the most accurate results. These parameters more often than not have to be tuned every time depending on the type of problem.

- Stopping Criterion -

  For a majority of the learning algorithms, there is no well defined point that indicates when learning should stop. In most cases, the designer decides that learning should stop when the performance of the algorithm, from his/her opinion, is “good enough”.

- Define a Reward Function -

In reinforcement learning problems, it is necessary for the designer to characterize the task by defining a reward function. This function encodes the task in the form of calculated rewards given to the learning agent. Without this function, the agent does not have a way to identify what the task is. It therefore necessitates that a human, who knows the task, explicitly defines a reward function.

Clearly the learning process is not currently feasible for non-experts. The aim of HELP is to bridge this gap and enable machine learning systems to succeed at learning via interaction with everyday people.

1.2.3 Current Approaches Tailored for Human Input

For years scientists and researchers working on robotic and software agents have been inspired by the idea of efficiently transferring knowledge about tasks or skills from a human to a machine. There are several related works that explicitly incorporate human input to a machine learning process. It is important to note that most of these works did not employ the use of everyday trainers. Nonetheless a review of the methods would help understand how human input has been utilized.

1.2.3.1 By observing human behavior

The human through an appropriate interface communicates the task to the learning agent by "showing" it. The agent then utilizes this information and generalizes to understand the specifics of the task. The complexity of the task, the transfer of human behavior and the choice of learning algorithm effect the ability for the agent learn. A few techniques are described below:

- Learning by Demonstration

  It is an extensive field of research focused on teaching an artificial agent behaviors and skills without extensive programming (e.g. a humanoid robot learning object manipulation tasks). At a high level, a human, playing the role of the teacher, has the task of effectively communicating the required behavior to the artificial learning agent through various methods of interaction It eliminates the need for

the designer to hard-code complex tasks which may vary depending on the application and the environment. A survey of the current methods of Learning by Demonstration can be found in [Argall et al. (2009)].

- Imitation Learning

  Imitation Learning [Mataric (1997); Nicolescu and Mataric (2001)] has often been used interchangeably with Learning by Demonstration. While this is not far from the reality, in the strictest sense of its meaning, Imitation Learning refers to the form of learning where an agent "observes" the teacher perform the task and uses that information to imitate the teacher. The quality of learning is purely based on the faithfulness of the reproduction of the action which has been demonstrated. It has been applied to several domains including development of fine robot motor control primitives [Pastor et al. (2009)], smooth robot walking gaits [Chalodhorn et al. (2007)], identifying object shapes [Montesano et al. (2008)], etc.

  Recent work in Imitation Learning [Ratliff et al. (2006)] took instances of a Markov Decision Process and demonstrations and tried to generalize these demonstrations based on assumptions about the linearity of costs with respect to the feature space. Such a task is considered as an inverse control problem (also known as Inverse Optimal Control).

- Inverse Reinforcement Learning (IRL)

  In a tradition RL setting [Sutton and Barto (1998)], the agent acts in the real-world by choosing actions that maximize its overall average reward. Here the reward function has to be manually designed by the user and this may vary depending on the application and it is also an arduous task for a non-technical user. To alleviate this problem, several researchers [Ng and Russell (2000); Ramachandran and Amir (2007); Ziebart et al. (2008); Lopes et al. (2009)] have developed algorithms that come under the category of IRL. In this setting, numerous examples of the task are given in the beginning (possibly in the form of demonstrations) along with the transition function (model of the environment). The agent uses these examples to automatically calculate the reward function of the expert. The

agent, with given transition function and the computed reward function, can plan and execute the required task.

- Apprenticeship Learning

  This form of learning has been worked on by several researchers [Abbeel and Ng (2004, 2005); Syed et al. (2008); Neu (2007)] and it is sometimes used in conjunction with IRL. The focus in Apprenticeship Learning is that the agent is trying compute the optimal policy by learning the transition function, given a set of features that define the world. In some cases, the reward function is learned as an intermediate step. It uses a set of examples provided by the expert in the beginning and then computes the optimal policy that matches the behavior of the expert. An application of this protocol is the Inverted Helicopter [Abbeel and Ng (2005)].

1.2.3.2 Providing labels and feedback

In other cases the human influences the experience of the machine with a set of higher level constructs, for example, providing feedback to a reinforcement learner or labels to an active learning system.

- Active Learning

  Active learning or learning with queries is a semi-supervised learning approach that utilizes a human "oracle" through queries [Schohn and Cohn (2000)]. An unsupervised learning algorithm identifies the most interesting examples, and then asks the oracle for labels. Thus, the algorithm is in control of the interaction without regard of what an ordinary human will be able to provide in a real scenario.

- Socially Guided Machine Learning

  This method explores ways in which human can teach agents using their natural social means of communication. In order to learn a particular task, the human trainer provides reinforcement, guidance, advice, motivation and other such social

protocols that are intermediately decoded to an agent understandable language and then utilized by it to generalize and learn the task. This method has been recently developed [Breazeal and Thomaz (2008); Thomaz et al. (2006)] and has been successfully applied to robotic domains in real-time.

- TAMER

  The TAMER Framework, Training an Agent Manually via Evaluative Reinforcement [Knox and Stone (2009)], seeks to transfer knowledge from the human to the agent through the means of human reinforcement. A human watches agent act in the world and has access to an interface that provides the agent feedback about its actions (negative or positive). This way, the agent learns the reward function of the human, extrapolates it to unseen states and accordingly modifies its policy in an attempt to satisfy the human's reward function. The framework has been successfully employed in teaching an agent to play a game of Tetris and solving the mountain car.

- Robotic Clicker Training

  It is based on a method that is commonly used for animal training. In this method a human provides feedback to robot through the means of a clicker. The clicker serves as an instrument of reinforcement. Clicker Training involves two kinds of reinforcement, the first one communicates to the agent that the current action is an important one (brings attention), the second reinforcement is given when the action has been completed in the desired manner. It has been used to learn complex tasks with the Sony Aibo robot [Kaplan et al. (2002)].

- Cobot - A Social Reinforcement Learning Agent

  It is a software agent that lives in an active online community called LambdaMOO. The goal of the agent is to interact with other members of the community and to become a real part of the social world. Cobot is allowed to take proactive actions in the social environment, and adapt behavior based on feedback from multiple sources. The agent builds models of those around it, based on statistics of who

performs what actions, and on whom they use them. It was shown [Jr. et al. (2001, 2006)] that Cobot learned nontrivial preferences for a number of users and learned to modify his behavior based on his current activity of other users.

1.3 Thesis Overview

This Thesis describes three novel case studies, each one detailing distinct aspects of HELP. The first study acts to set a base line by describing a standard human robot interaction experiment. It describes how the task to be learned is represented and the constraints associated with it. The studies that follow will build upon the pervious work and enhance the learning accuracy by a) modifying the structure of the learning algorithm and b) modifying the mode of human interaction for efficient learning.

Chapter 2 describes a case study where a human interacts with a humanoid robot and teaches it to perform a set of tasks by using supervised learning algorithms. The study details the task representation, the choice of learning algorithm, the mode of human interaction and finally the effectiveness of the interaction. Chapter 3, the second case study, explores the interactions of a human with novel reinforcement learning algorithm. This approach focuses on the characteristics of the algorithm and how it makes use of the human inputs. Chapter 4, the final case study, investigates how standard reinforcement learning algorithms perform when given different forms of input from the human trainer. It describes the aspects of the task that humans understand, the information that they are ready to provide and the performance of the learning algorithm using each of these inputs. Chapter 5 provides a summary of the three case studies, the challenges with human guided systems, some concluding remarks and future work. Each of the case studies involves human subjects interacting with the learning algorithm either through a robot or an equivalent software implementation.

Chapter 2

Experiments in HELP

This chapter details three case studies focusing on the different aspects of human interactions with learning protocols. The first case study describes a framework where a human uses kinesthetic means (direct control of a robot's actuators) to teach a task to a robot using a supervised learning algorithm. This case study will provide a basic introduction to the task representation, a method of human communication, a machine learning algorithm, the interaction between the two and their evaluation. We instantiate this approach in a real robot [1]. Further case studies will explore different aspects of the framework described here.

2.1 Case Study 1 - Learning by Demonstration by Supervised Learning

Learning by Demonstration (LbD) is an area of research that focuses on teaching robots new behaviors without extensive programming, rather by simply "showing" the robot how to do a task. The rationale behind this approach is that using "natural" means of teaching a machine minimizes the need for tedious programming of the task by a human user. The implementation of an LbD algorithm can be broken down into three steps:

1. Choice of Hardware - The first step selects the machine that the human is trying to teach. Robots have different degrees of freedom and they must be chosen by taking into account the complexity of the task to be learned. The type of robot used has a direct effect on the choices made in following steps. Recent advances

[1] Portions of this work have appeared earlier in Subramanian (2010)

in the field have made use of the HOAP^{TM}Humanoid Robot by Fujitsu [Calinon (2007)], Sony's Aibo^{TM}Robot [Chernova and Veloso (2007)].

2. Method of Demonstration - This step focuses on the different interfaces that can be used to transfer the task description to the machine. A few examples are Kinesthetic Teaching (manually controlling the robot) [Calinon and Billard (2007)], Observational Learning (watching the human perform the task) [Kang and Ikeuchi (1995)] and Vocal Direction (human describes the task and its actions vocally) [Dominguez et al. (2006)].

3. Learning Algorithm - The next step of the algorithm processes and generalizes the data acquired in Step 2. This step determines how well the robot can perform the task on its own. Popular methods include Supervised Learning [Calinon and Billard (2008a); Chernova and Veloso (2007)] and Reinforcement Learning [Abbeel and Ng (2004)].

By combining these steps with the efficient computing tools available today, LbD has proven to be a powerful mechanism for robot learning in complex spaces. Reviews of current advances in the field can be found in Argall et al.; Billard et al. (2009; 2008).

We will now elaborate on the implementation of an LbD algorithm on a humanoid robot to learn constrained reaching gestures. The gestures we are interested in learning are Obstacle Avoidance and Move Object to Goal.

2.1.1 Learning Machine - Nao Humanoid Robot

The robot that was used for this case study was Aldebaran's Nao^{TM}, shown in Figure. 2.1. It is an autonomous, programmable and medium-sized humanoid robot with 21 degrees of freedom (DOF). Being a humanoid robot with such high DOF, made it well suited for both efficient interaction with humans and learning complex tasks. The Nao has an inertial sensor and 4 ultrasound captors that provide stability and positioning within space. The robot interacts with the environment via text-to-speech synthesis, sound localization and facial recognition using its 4 microphones, 2 speakers and 2

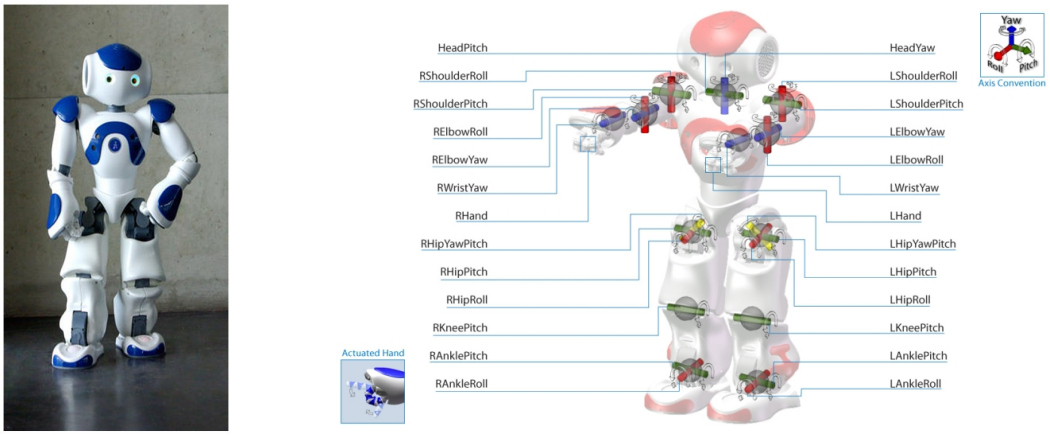


Figure 2.1: The Aldebaran's Nao Humanoid Robot and its Degrees of Freedom

CMOS cameras. The Nao is also the official robot used in the Robocup Competition [Baltes et al. (2010)].

2.1.2 Demonstration Method - Kinesthetic Teaching

After choosing the robot, we must now focus on the mode of demonstration. A human, playing the role of the demonstrator, has the task of effectively communicating the required behavior to the artificial learning agent through various methods of interaction. The mode of interaction used in this study is Kinesthetic Demonstration [Calinon and Billard (2008b)], whereby the human communicates behaviors by manually controlling the robots end-effectors. This form of interaction is less prone to error as the correspondence problem is simplified. The correspondence problem [Nehaniv and Dautenhahn (2001)] arises due to the differences in the body structure and dynamics of the teacher and the robot. The transfer of motion from the teacher to the robot is a non-trivial task. To overcome this issue, methods of demonstration like Kinesthetic Teaching, provide information from the robot's stand-point rather than the teacher's. In this form of demonstration, it is important to understand the limitations associated with the demonstrator as well as the imitator. Demonstrating complex tasks that involve controlling multiple robot actuators in parallel can be impossible for a single human and similarly the complexity of the task is limited by the hardware and software capabilities of the robot. Given a robot task that involves a set of motor actions to be performed

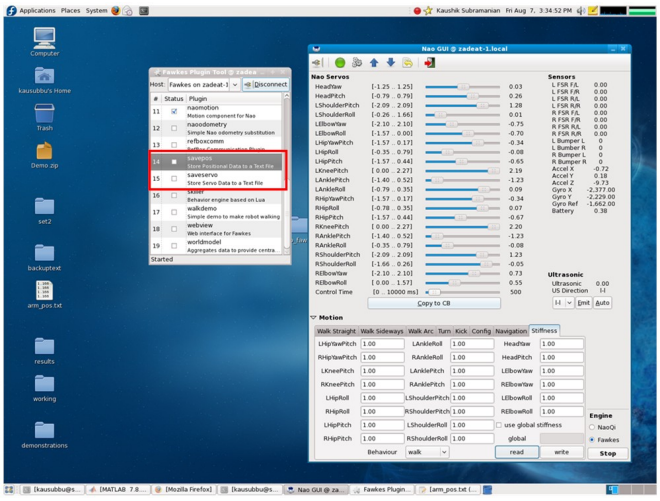


Figure 2.2: Screenshot of *Fawkes* GUI to enable plugins and control stiffness

in a sequential manner, we can sequence and decompose the tasks into known sets of actions, performable by both the demonstrator and the imitator.

Depending on the task, the user must first decide which end-effectors of the robot will be used. It can be the head, right/left arm, right/left leg or a combination. The extent of control is limited by the users capabilities to control multiple robot chains at the same time. Once decided, the stiffness associated with those joints is set to a value that allows free movement.

Fawkes, *http://www.fawkesrobotics.org* [Niemueller (2009)], an open source robot software designed for the Robocup platform was used to communicate with the robot. *Fawkes* allows the user to control the various motion, vision and communication components of the robot. It uses plugins to load and unload these components based on the user's needs. A screenshot of the *Fawkes* GUI is shown in Figure 2.2. Using *Fawkes* the human transfers the behavior in the following manner:

1. Enable the *motion* plugin and set the stiffness of the required joints to zero. Say, we are interested in using robot's left arm.

2. Enable the *savedemonstration* plugin and manually control the joints in the way required. For example, the user can now move the robotic arm towards a cup and then carry it towards the saucer. During the demonstration, the

savedemonstration plugin simply stores the raw servo data of the robot to a text file. In this case, it will form a dataset with each row having 21 Dimensions.

3. Convert the data to a $6D$ position and orientation vector using the in-built kinematics model.

2.1.3 Learning Algorithm - Gaussian Mixture Regression

We choose Gaussian Mixture Models (GMMs) for efficient learning of constrained reaching tasks as they are robust to noise, can efficiently generalize, and can capture correlations between continuous features [Chernova and Veloso (2007)]. Gaussian Mixture Regression (GMR) is performed on the generalized model to reproduce the task under different circumstances. After acquiring the set of task trajectories through the demonstrations, it is necessary for us to extract the associated constraints to better understand how to reproduce the task. The generalization using GMMs is performed on the extracted task constraints.

2.1.3.1 Task Constraints

Figure 2.3: Mouse Task - Mouse starting from random position has go to the File and move it to the Bin

Given a demonstrated behavior, it is necessary for us to describe the constraints associated with it in order to reproduce the task. Let us consider a simple example like the mouse task [Calinon (2007)] shown in Figure 2.3. The task is straightforward - the mouse starting from a random position has go to the File and move it to the Bin. The question is how to define constraints for such a task. Constraints are achieved by calculating the relative positions of the mouse with respect to the file and bin. Consider a set of 4 demonstrations shown in Figure 2.4. The highlighted oval portions show the

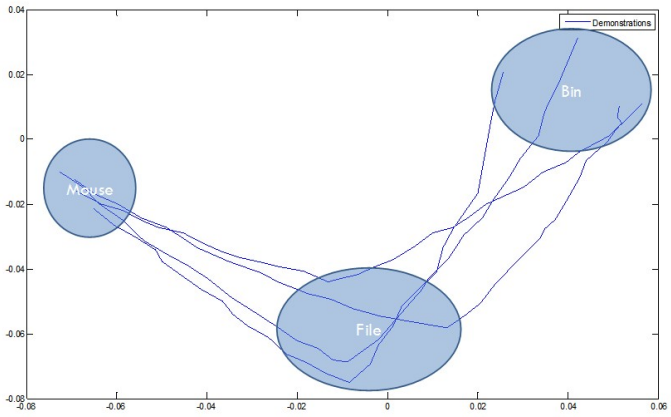
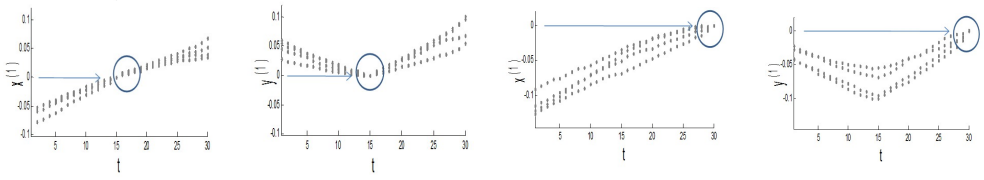


Figure 2.4: A set of 4 demonstrations of the Mouse Task with the highlighted positions of the Mouse, File and Bin.



(a) X coordinate position of Mouse relative to the File

(b) Y coordinate position of Mouse relative to the File

(c) X coordinate position of Mouse relative to the Bin

(d) Y coordinate position of Mouse relative to the Bin

Figure 2.5: Task Constraints for the Mouse Task

different positions of the mouse, file and bin used in the demonstrations. Given this dataset, we calculate the relative distance to extract the constraints. The constraints are shown in Figure 2.5. The highlighted sections show us the sample points where the mouse reaches the file and bin respectively. Therefore at these points the x,y distance coordinates are zero, meaning the mouse and file/bin are at the same location. This Mouse task can be extended to a number of scenarios and domains, like robot soccer, obstacle avoidance etc. These constraints are then temporally aligned and generalized using Gaussian Mixture Models.

2.1.3.2 Gaussian Mixture Models

A probabilistic representation of the computed constraint data is used to estimate the variations and correlations across the variables, allowing a localized characterization of

the different parts of the gesture. Mixture modeling is a popular approach for density approximation of continuous constraints data [Calinon and Billard (2008b)]. A mixture model consisting of K components is defined by the probability density function:

$$p(\epsilon_j) = \sum_{k=1}^{K} p(k)p(\epsilon_j|k) \tag{2.1}$$

where ϵ_j is the input dataset, K is the number of Gaussian components, $p(k)$ the priors and $p(\epsilon_j|k)$ is the Gaussian function represented by

$$p(\epsilon_j|k) = \frac{1}{\sqrt{(2\pi)^D|\Sigma_k|}} e^{-1/2((\epsilon_j-\mu_k)^T\Sigma_k^{-1}(\epsilon_j-\mu_k))} \tag{2.2}$$

From the above equation, we need to calculate the mean μ_k, covariance matrices Σ_k and priors $p(k)$. The value of K is task specific and can be estimated by trial and error or by using the Bayesian Information Criterion (BIC) [Calinon and Billard (2008b)].

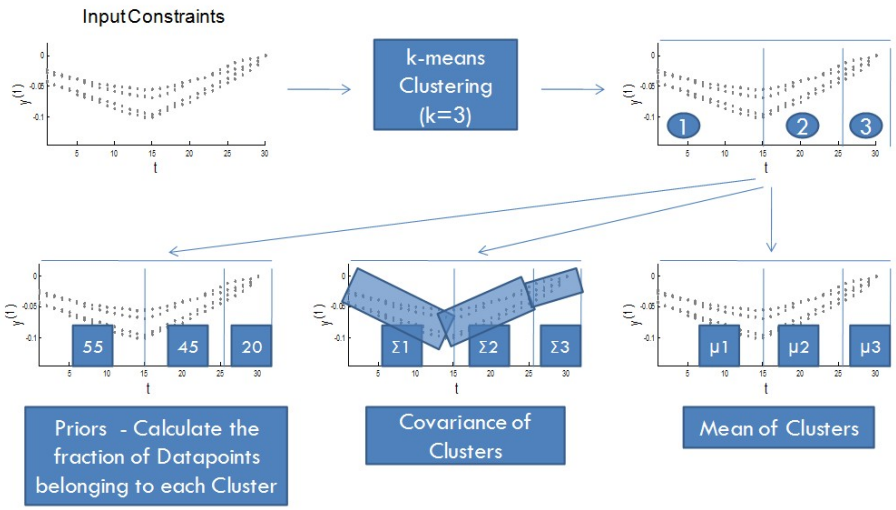


Figure 2.6: Extracting the Mean, Covariance, Priors (GMM Parameters) from the Task Constraints

The required Gaussian parameters are estimated using the setup shown in Figure 2.6. We perform Maximum Likelihood Estimation iteratively using the standard Expectation-Maximization (EM) algorithm. EM is a simple local search technique that guarantees monotone increase of the likelihood of the training set during optimization. The algorithm requires an initial estimate of the parameters, and to avoid getting trapped into a poor local minima a rough k-means clustering technique is first applied to the data [Calinon and Billard (2008a)]. The Gaussian parameters are then derived

from the clusters found by k-means. The computed Gaussian parameters are shown in Figure 2.7. Having estimated these parameters, the next step is to reproduce new trajectories given new situations. We perform Gaussian Mixture Regression for this purpose. The basis of regression is to estimate the conditional expectation of Y given

Figure 2.7: Task Constraints on the left have been generalized and represented as gaussian models on the right side

X on the basis of a set of observations (X, Y). Consecutive temporal values are used as query points and the corresponding spatial values are estimated through regression. In our case, X Time sample, Y Cartesian coordinates. Therefore, given time samples as input data, the regression algorithm outputs a smooth generalized version of the observed trajectories encoded in the GMM as shown in Figure 2.8. It should be noted that it is not equivalent to taking the mean and variance of the data at each time step, which would produce jerky trajectories and dramatically increase the number of parameters (the mean and variance values would be kept in memory for each time step). With a probabilistic model, only the means and covariance matrices of the Gaussians are kept in memory.

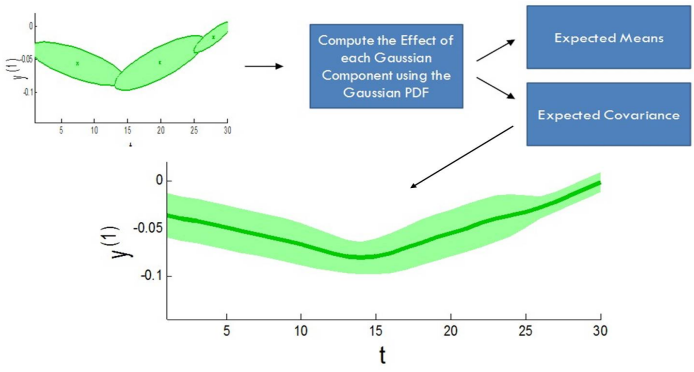
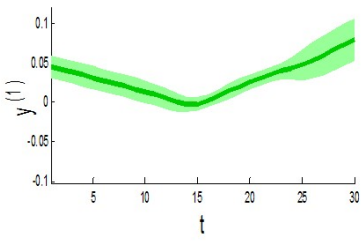
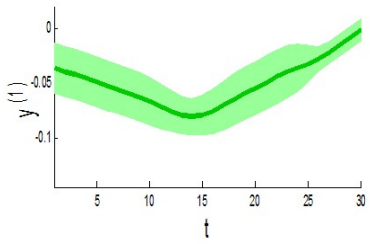


Figure 2.8: Regressed model of the generalized data

2.1.3.3 Trajectory Reproduction



(a) GMR in Task Space, relative to File



(b) GMR in Task Space, relative to Bin

Figure 2.9: Gaussian Mixture Regressed representations with respect to system objects

The regressed model that has been obtained is the generalized version of the relative positions of the File and Bin with respect to the Mouse (Task Constraints). It is shown in Figure 2.9. Given new positions of the Mouse, File and Bin, we must reconstruct the required trajectory. Using the GMM parameters, the reconstruction is performed using the equation below [Calinon and Billard (2008b)]:

$$x_{j+1}^{(n)} = (o^{(n)} + \hat{x}_{j+1}^{(n)}) - x_j \tag{2.3}$$

where j represents the temporal step, x_j is the position of the robot end effector at step j, $o^{(n)}$ is the position of the object and n is the number of system objects.

2.1.4 Experiments and Results

After performing the steps mentioned in the previous sections, we now instantiate the algorithm on a 2D representation of the Mouse task and a real robot experiment. The experiments are described below -

Experiment 1

We created a 2D simulator version of the mouse task explained earlier and provided the learning agent 3 to 4 demonstrations to train on. We then tested the agent's performance in completing the task for circumstances not encountered during the training. The resulting trajectories are shown in Figure 2.10 and Figure 2.11. The new positions of the mouse, file and bin, are indicated by the "+" signs. The dashed lines indicate the

demonstrations used for training and the red line indicates the reproduced trajectory. We can see the system has produced a trajectory that passes very close to the new positions.

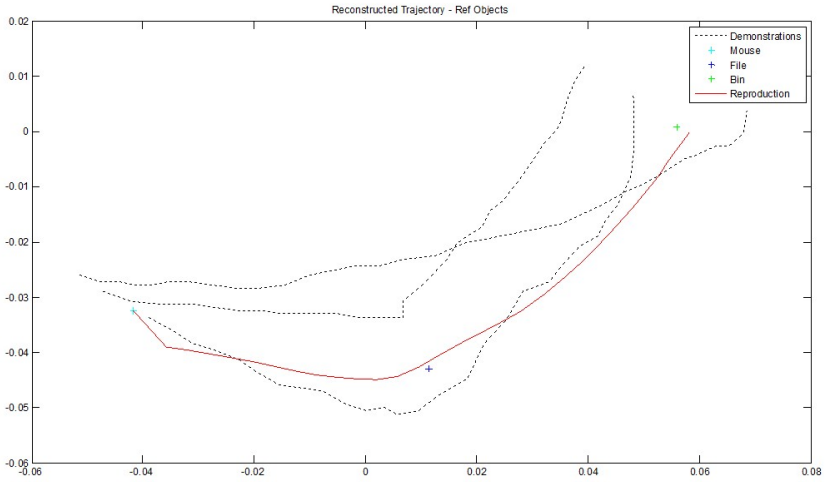


Figure 2.10: This experiment contains a set of 3 demonstrations of the mouse task. The red line indicates the reproduced trajectory for new positions of the mouse, file and bin (indicated by the + signs).

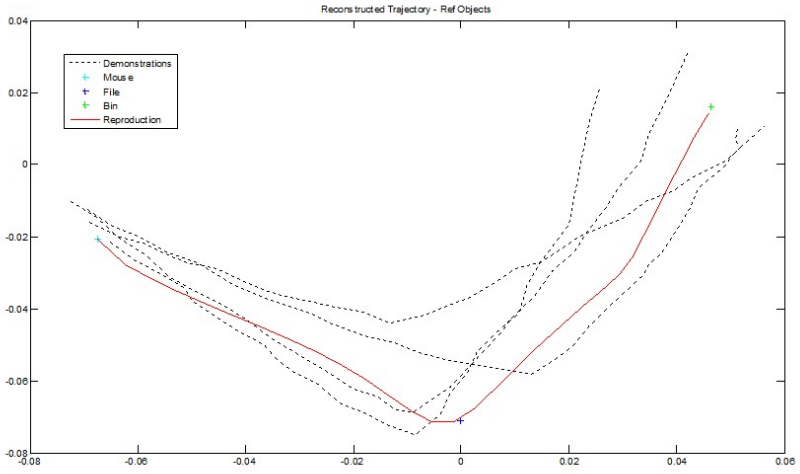


Figure 2.11: Results for a mouse task with 4 demonstrations and new starting positions.

Experiment 2

Given three demonstrations of the Mouse Task, we tested the agent's response to abrupt changes in the positions of the file/bin during trajectory reproduction. The resulting

trajectories are shown in Figure 2.12 and Figure 2.13. The red line, indicating the reproduced trajectory, shows the sudden change in its path to compensate for the unexpected change in the environment.

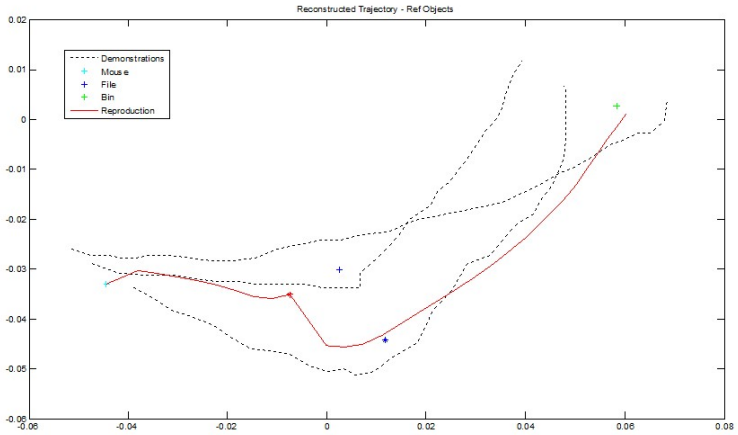


Figure 2.12: Shows the adaptability of the system to changes in the position of the file during execution of the trajectory.

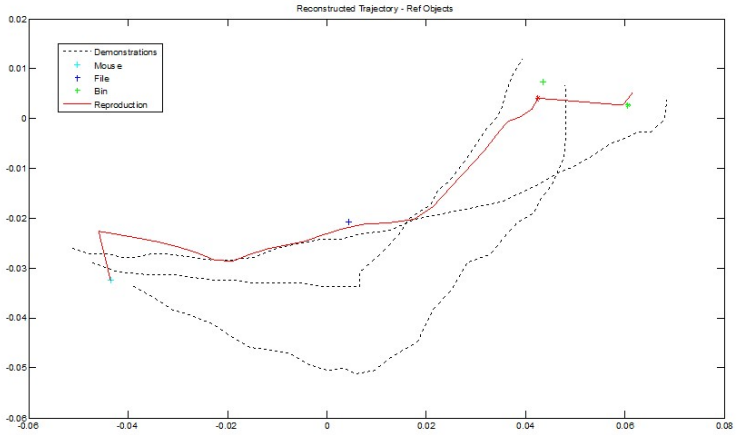


Figure 2.13: By changing the position of the bin in the very last time step, the system is able to account for the new position.

Experiment 3

We perform a 2D Simulated Obstacle Avoidance test. The agent is trained using six demonstrations to avoid objects of varying heights and orientation placed in the central region. We tested its response to avoid obstacles of heights not used in training. The

resulting trajectories are shown in Figure 2.14 and Figure 2.15.

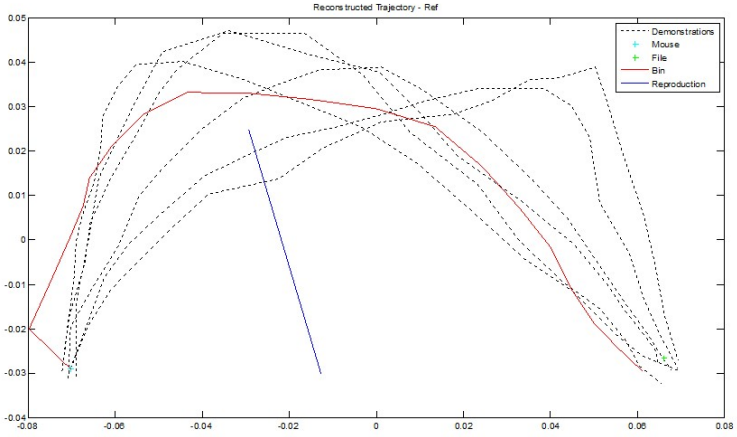


Figure 2.14: A 2D obstacle avoidance task result

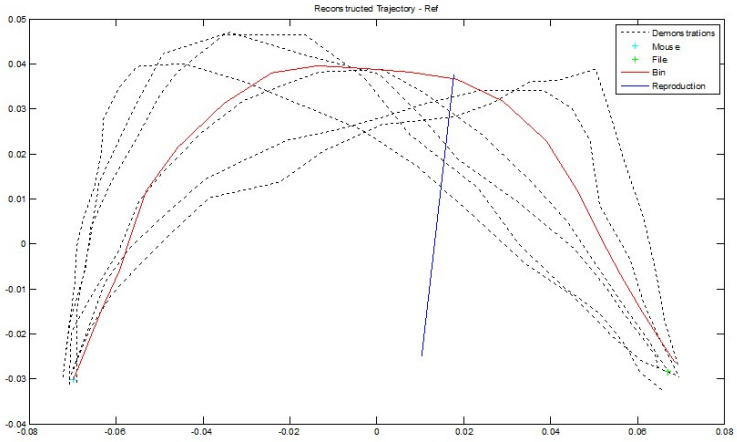


Figure 2.15: A 2D obstacle avoidance task result

Experiment 4

We used the Nao robot to perform this experiment. The task is for the robot to use its left arm (starting from a specific position) to move towards an object and carry it towards the goal. We performed 20 demonstrations of the task using different locations for the object. These are shown in Figure 2.16 where the "+" indicates the demonstration positions and "*" indicates the successful testing positions. The resulting trajectory from one these positions is shown in Figure 2.17.

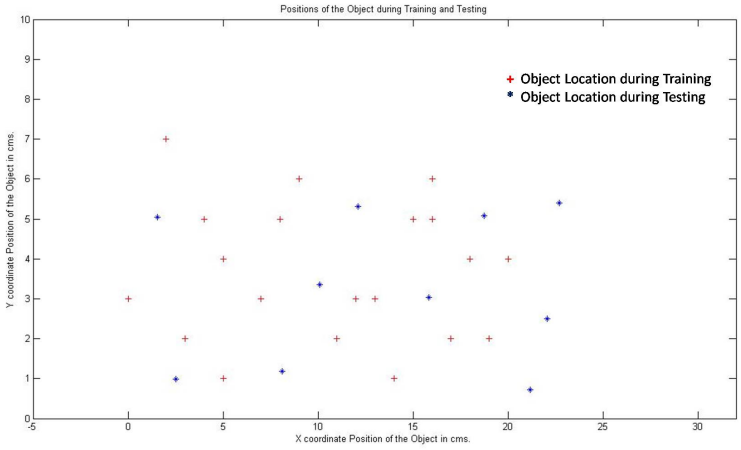


Figure 2.16: Different positions of the object used for training and testing

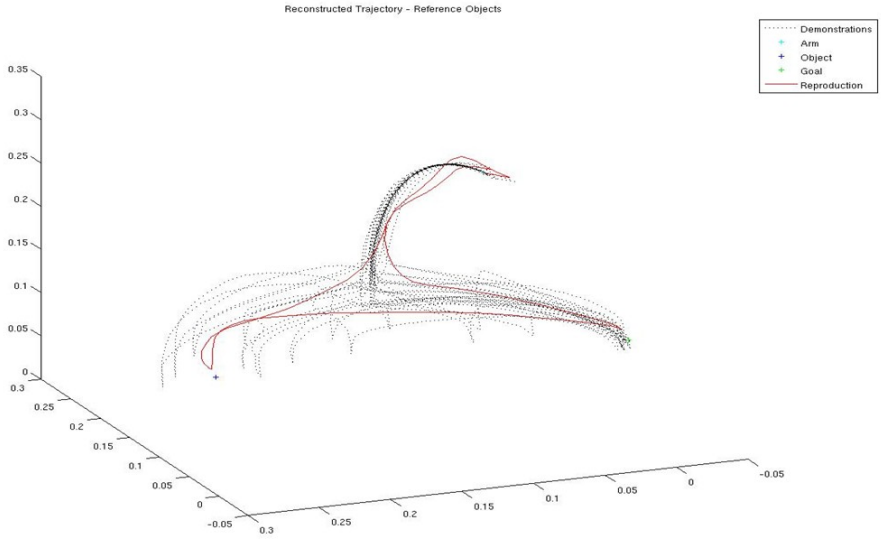


Figure 2.17: A 3D demonstration using the right arm of the Nao to perform the object to goal task.

Experiment 5

We used the right arm chain of the Nao to perform an Obstacle Avoidance task. The agent is trained using 6 demonstrations to avoid objects of varying heights (5, 8, 12 and 15cm), placed in the central region. We tested its ability to overcome obstacles of heights not used in training. The resulting trajectory for an obstacle of height $10cm$ is shown in Figure 2.18.

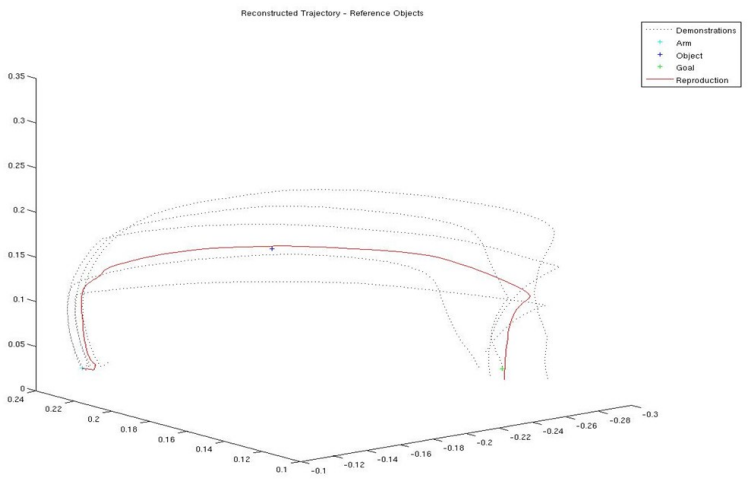


Figure 2.18: Obstacle Avoidance for the Nao after training it for 5,8,12 and 15cm heights. It was tested with 10cm

2.1.5 Experimental Observations

During experiments, two important observations were made with respect to the choice of the number of gaussian components (K) and the required number of demonstrations.

Choice of Gaussian Components

From the Figure 2.19 and Figure 2.20, we see that when choosing K=3, the regressed model becomes to narrow and constricted and therefore may not lead to accurate results in all positions. However, changing K=2 produces a regressed model that accounts for all the training demonstrations.

Choice of Demonstrations

It is important to keep the demonstrations similar or reasonably close to each other, otherwise the GMM will not be able to generalize the dataset if there is large variation as shown in Figure 2.21. If such a gap exists, more demonstrations should be performed focusing on those areas.

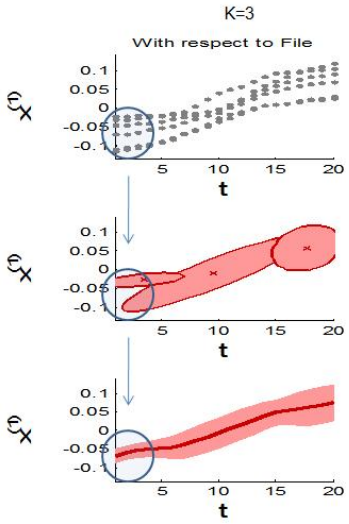


Figure 2.19: Perform GMM using three components.

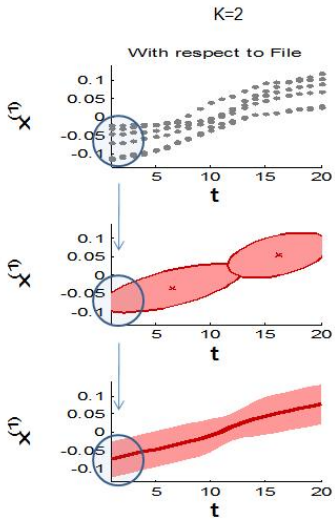


Figure 2.20: Perform GMM using two components.

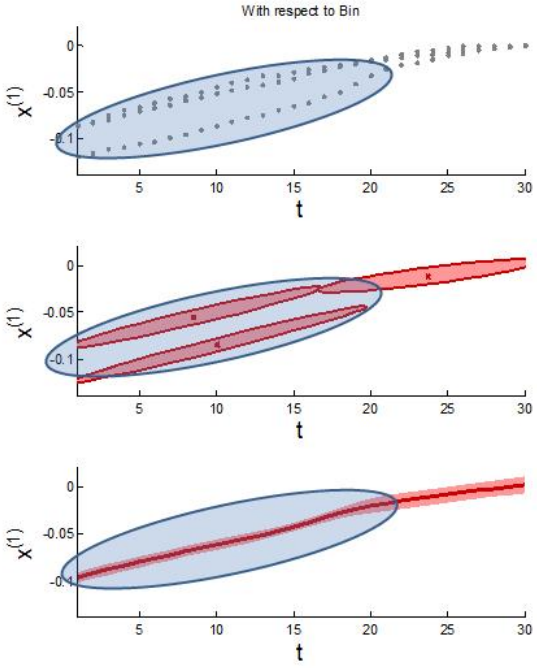


Figure 2.21: Three sets of demonstrations with one quite different from the remaining two.

2.1.6 Limitations

There are four main limitations associated with the present implementation of the algorithm:

1. Task space constraints alone do not satisfy the required level of performance. It is not enough to only generalize the coordinate positions of the objects, better accuracy can be achieved by taking the robot joint space constraints in account.

2. Every object in the system has to be defined by a six dimensional vector. As we are using task space constraints and working with the Nao robot, every object needs a 3D position and 3D orientation. It maybe difficult to calculate in all circumstances and the present solution manually takes the arm/leg to the destination position and registers (at each timestep) the coordinates and orientation.

3. The discrete reproduced trajectory may contain sharp turns. Less frequently, the algorithm tries to follow the path of the demonstrations already shown, in this attempt it sometimes creates sharp turns and jerky movements. This motion may harm the servo motors of the robot.

Overcome the Limitations

- Use of forward and inverse kinematics model allows us to employ joint constraints and do away with 6D object coordinates.

- Use of a vision system allows positions to be automatically registered.

- Use of the approximate nearest neighbor approach when using higher dimensions of the robot can help speed up computation of the generalizing model.

2.1.7 Summary

1. The generalized model was robust to the noise associated with the servo motors of the robot. The memory consumed in these experiments was very small as only the means and covariance matrices needed to be stored [Calinon (2007)].

2. The robot was successful in completing the defined tasks in positions and orientations that were not included in the demonstrations.

3. We tested the robustness of the model to sudden changes in the positions of the objects while the robot was executing the behavior. The model was able to overcome the online perturbations and go on to complete the task.

This case study showed how humans can teach a variety of object relocation tasks to a robot using kinesthetic means of interaction. It shows how the human, using a small number of demonstrations, was able to influence the underlying learning algorithm (GMM) in such a way so as to achieve more efficient learning. A autonomous agent, given the same task, will require an exponential amount of time before it is able to generalize across all combinations of the task. With HELP, the robot was able to take advantage of the knowledge of the human to complete the task.

Chapter 3

Experiments in HELP

In the previous study, we introduced the essential concepts of LbD and showed how humans and standard learning algorithms can influence the ability of a robot to learn reaching gestures and object relocation tasks. The case study described in this section will focus on advancing the learning algorithm used by the artificial agent. We describe a novel reinforcement learning algorithm that efficiently utilizes the inputs given by the human. This algorithm ensures that the agent will only require a limited number of interactions from a human in order to guarantee efficient learning. We instantiate the approach in a real robot experiment [1].

3.1 Case Study 2 - A Novel Approach to Learning by Demonstration using Reinforcement Learning

Consider the example shown in Figure 3.1. It describes a simple task where a robot starting at a random location, has to manoeuvre through obstacles, pick up a key and using that key open the door to the goal. Intuitively one can guess that an autonomous robot would be able to learn and perform such a task. However imagine if instead of using a key to get the goal, the robot had to solve a combination lock. A lock that has 2^n combinations. One can imagine that the robot would probably take a very long time to learn the task autonomously. Whereas with a human teacher, the learning problem becomes a lot easier to solve. In this case study, we introduce a novel protocol in which a Reinforcement Learning (RL) agent interacts with a human and learns to solve the combination lock and numerous other problems in different domains. The

[1] Portions of this work appeared earlier in joint work with Thomas Walsh, Michael Littman and Carlos Diuk

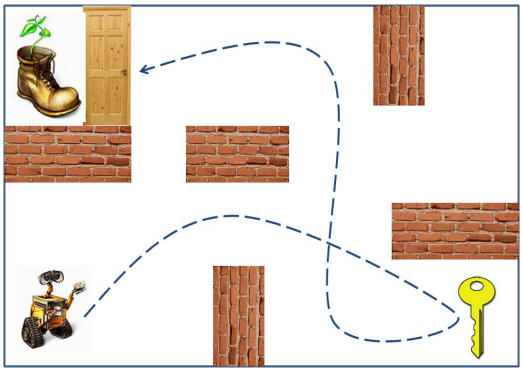


Figure 3.1: Robot tracing a path through the obstacles to get the key that opens the door to the goal.

method described is an extension of the Apprenticeship Learning Protocol [Abbeel and Ng (2004)]. We generalize the approach taken by Abbeel and Ng (2004) and show that it can be applied to several domains that were previously thought to be intractable. The study also details a model of sample efficiency by which the number of human interactions can be limited and yet ensure that the agent will converge to optimal behavior. The protocol is instantiated in both simulator and robot domains.

3.1.1 Reinforcement Learning and Teachers

A recent survey has shown that Reinforcement Learning has been used in conjunction with Teachers for a wide variety of applications. There has been work in the field of Learning by Demonstration [Atkeson and Schaal (1997)], Online Social Reinforcement Learning [Jr. et al. (2001)], Socially Guided Machine Learning [Thomaz et al. (2006)], Apprenticeship Learning [Abbeel and Ng (2004)] and others. Each of these approaches involves a teacher communicating with a reinforcement learning agent to solve simulated and real-world problems. The bulk of research in this field has shown that there are clear advantages with having a human teacher in the agent learning loop. These include the ability to help an artificial agent efficiently explore large environments, to alleviate real-world stochasticity, removing the need for hard-coding a behavior and others. These are just a few of the advantages, there is always the satisfaction of having taught a machine a complex task.

It also interesting to note the several different ways in which humans have interacted with RL agents to facilitate the process of learning. Research has shown that humans use demonstrations, reinforcement, social protocols like advice and guidance and other such methods. Each one focuses on making the communication more natural and intuitive for the human. The most popular approach has been by using demonstrations. Here, the human "shows" how to do the task, the robot processes that demonstration, extracts the required information and generalizes to perform the task again with different state configurations.

In this study, we will be extending the Apprenticeship protocol [Abbeel and Ng (2004)] to be able to learn tasks that were thought to be intractable for this algorithm.

3.2 The Apprenticeship Learning Protocol

We begin by defining the basic structures that encode reinforcement-learning domains. A reinforcement-learning [Sutton and Barto (1998)] agent interacts with an environment described by a Markov Decision Process (MDP). An MDP is considered to be a tuple $\langle S, A, T_{sa}, R, \gamma \rangle$, where

- $S = s_1, s_2..., s_n$ is a finite set of n states.
- $A = a_1, a_2..., a_k$ is a set of k actions.
- T_{sa} are the state transition probabilities associated with action a in state s.
- $R : S \rightarrow \mathbb{R}$ is the reward function bounded by R_{max}.
- $\gamma \in [0, 1)$ is the discount factor.

An agent's deterministic policy $\pi : S \mapsto A$ induces a value function over the state space, defined by the Bellman Equations:

$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s'} \mathcal{T}(s, a, s') Q^{\pi}(s', \pi(s'))$ and $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$.

An optimal policy π^* for an MDP is a policy such that $\forall s$, $V^*(s) = V^{\pi *}(s) \geq V^{\pi'}(s), \forall \pi'$.

The goal of a standard (autonomous) reinforcement-learning agent is to achieve behavior close to this optimal value from its own experience sampling $\mathcal{T}$ and R on each step.

We now describe the protocol introduced by Pieter Abbeel and Andrew Ng in 2004. In their framework, a agent is given a set of demonstrations of a task from an expert human. The framework then generalizes over those demonstrations, computes a model of the environment and computes the optimal policy that the agent should follow. This protocol is shown in Figure 3.2. The state space is defined by a set of features that characterize important information about different parts of the world. Here are some

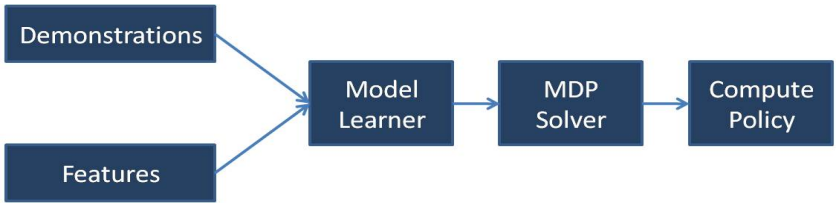


Figure 3.2: The Apprenticeship Learning Protocol by Pieter Abbeel and Andrew Ng.

of the constraints and assumptions that they made:

- All the start positions of the demonstrations are drawn from a fixed distribution.
- All the demonstrations are given in bulk upfront.
- The reward function is assumed to be a linear function of the feature weights.

We will describe a protocol by which we overcome these constraints and extend the framework to other learnable classes.

3.2.1 A Generalized Approach to Apprenticeship Learning

We now consider a paradigm where the agent's experience is augmented with experience produced by a teacher and the criterion is to find a policy whose value function is nearly as good as, or better than, the value function induced by the teachers policy. The value function here is indicative of how "good" the computed policy is. The components of the system include a standard RL agent, a Model Planner that builds a model of

the world based on the agents observations, a Planner to plan a path/trajectory that will maximize cumulative reward, the Environment in which the agent is acting and a Teacher who provides inputs to help the agent perform the task. These components are put together in a manner shown in Figure 3.3. The agent starts from a new initial

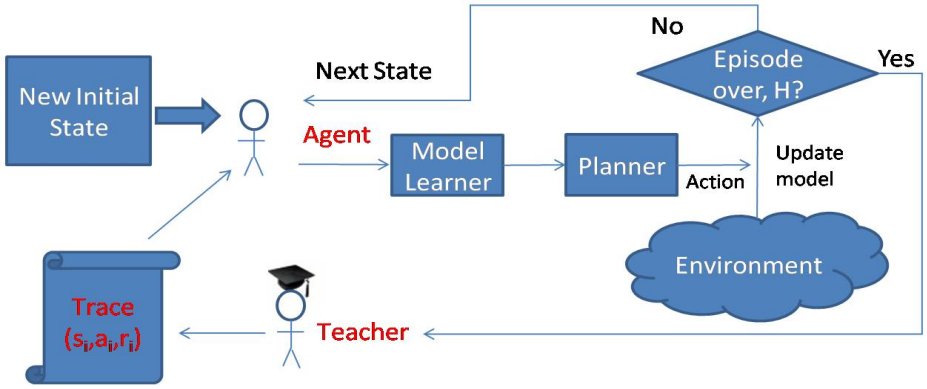


Figure 3.3: The Generalized Apprenticeship Learning Protocol.

state and accesses the Model Planner to compute a model of the world that it is in. The Planner using that model generates a path from start to goal that will maximize cumulative reward. The agent reviews the computed plan and takes actions accordingly. This loop continues until the Episode limit, H has been reached. After which the Teacher, depending on his assessment of the agent's performance, provides a trace from start to goal. The criteria the teacher uses to decide when to send a trace is left general here, but one specific test of value is for the teacher to provide a trace if at any time t in the episode, $Q^{\pi_T}(s_t, a_t) < Q^{\pi_T}(s_t, \pi_T(s_t)) - \epsilon$ (ϵ here is an accuracy parameter). That is, the agent chooses an action that appears worse than the teacher's choice in some state. Traces are of the form: $\tau = (s_0, a_0, r_0), ...(s_t, a_t, r_t), ...(s_g, r_g)$, where s_0 is the initial state, and s_g is a terminal (goal) state or some other state if the H cutoff is reached. Notice that the trajectory begins at the initial point where the agent first started, and may not even contain the state in which the agent made its mistake. The individual transitions in this trace must adhere to the environment's dynamics (the teacher cannot unilaterally pick next states).

Formally, we define the Apprenticeship Learning Protocol for episodic domains

where each episode has a length of at most $H = Poly(|M|, |A|, R_{\max}, \frac{1}{1-\gamma})$ in Algorithm 1. Here, $|M|$ is a measure of environment complexity.

Algorithm 1 The Apprenticeship-Learning Protocol

The agent starts with S, A and γ, a time-cap H and has access to episodic environment E
The teacher has policy π_T.
for each new start state s_0 from E **do**
  $t = 0$
  while The episode has not ended and $t < H$ **do**
    The agent chooses a_t.
    $\langle s_{t+1}, r_t \rangle = E.progress(s_t, a_t)$
    $t = t + 1$
  end while
  if the teacher believes it has a better policy for that episode **then**
    The teacher provides a trace τ starting from s_0.
  end if
end for

We take into account that fact that in the real world, no human is perfect. Therefore it is quite possible that teacher's trace might do worse that what the agent thought to be correct. Since the teacher's value function may not be optimal, we distinguish between these traces and their more helpful brethren with the following definition.

Definition 1 *A **valid trace** (with accuracy ϵ) is a trace supplied by a teacher executing policy π_T delivered to an agent who just executed policy π_A starting from state s_0 such that $V^{\pi_T}(s_0) - \epsilon > V^{\pi_A}(s_0)$.*

A trace in which the human teacher does better than what the agent thought was best is called a Valid trace. Defining valid traces in this way allows agents to outperform their teacher without being punished for it. With deterministic policies, this definition means that at no time in a valid trace does the teacher prescribe an action that is much worse than any of the actions the agent used in that state. Note that when the teacher enacts optimal behavior ($\pi_T = \pi^*$), only valid traces will be provided.

3.3 Sample Efficiency in Apprenticeship Learning

We now introduce a teacher into the learning loop in a way that allows us to characterize the *efficiency* of learning analogous to the PAC-MDP framework [Strehl et al. (2009)]. PAC-MDP stands for Probably, Approximately, Correct Markov Decision Process. It is defined by the accuracy parameters ϵ and δ in this manner - an agent is said to be PAC-MDP if with probability $1-\delta$, it is ϵ close to optimal behavior. This framework has been studied widely in a number of papers [Strehl et al. (2006, 2009); Asmuth et al. (2009)]and has been primarily used to characterize efficient behavior in the autonomous setting. In this study we define the PAC-MDP-Trace framework for models that involve a teacher in the learning loop. We define PAC-MDP-Trace learning as follows:

Definition 2 *A reinforcement-learning agent is said to be* PAC-MDP-Trace *if, given accuracy parameters ϵ and δ, and following the protocol outlined in Algorithm 1, the number of valid traces (with accuracy ϵ) received by the agent over its lifetime is bounded by Poly$(|M|, |A|, R_{\max}, \frac{1}{1-\gamma})$ with probability $1-\delta$, where $|M|$ measures the complexity of the MDP's representation, specifically the description of $\mathcal{T}$ and R (the environment's dynamics and reward functions).*

This definition allows us to characterize the efficiency of an agent based on its behavior. We show that the number of times a teacher can provide a valid trace in many domains is bounded. We now introduce a class of models where PAC-MDP-Trace behavior can be induced.

3.3.1 Model Learning Frameworks

We now provide a brief introduction to two model learning frameworks - KWIK and MB. We show how they can be used for efficient model learning and then go on to describe the model learner used in our framework. We will describe these model learners using the combination lock example described earlier.

3.3.1.1 KWIK - Knows What It Knows

In autonomous reinforcement learning, the recent development of the KWIK [Li et al. (2008)] or "Knows What It Knows" framework has unified the analysis of models that can be efficiently learned. The KWIK-learning protocol for supervised learning consists of an agent seeing an infinite stream of inputs $x_t \in X$. For every input, the agent has the choice of predicting a label ($\hat{y}_t \in Y$) or admitting $\perp$ ("I don't know"). If the agent predicts $\perp$, it sees a noisy observation z_t of the true label. This is shown in Figure

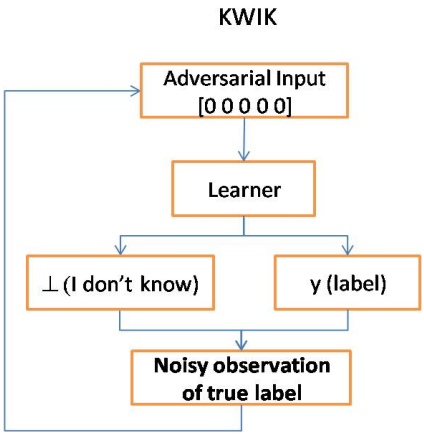


Figure 3.4: The Knows What It Knows Protocol.

3.4. A hypothesis class is said to be KWIK learnable if, with probability $1 - \delta$ two conditions are met:

1. Every time the agent predicts $\hat{y}_t \neq \perp$, $||\hat{y}_t - h^*(x_t)|| \leq \epsilon$.

2. The number of times $\hat{y}_t = \perp$ is bounded by a polynomial function of the problem description.

In autonomous reinforcement-learning, if $\mathcal{T}$ and R are efficiently KWIK learnable, efficient behavior can be achieved by optimistically filling in the $\perp$ predictions. These results cover a large number of models common to autonomous reinforcement learning, including flat MDPs and DBNs [Li et al. (2008)]. However, several relevant and intuitively appealing classes are not KWIK learnable. For instance, conjunctions of n terms

are not KWIK learnable because negative examples are so uninformative. An environment with a "combination lock" of n tumblers that need to be set to the correct digits for a high reward action ("unlock") to be effective, can require an exponential number of suboptimal steps. But, in the trace setting, learning to open such a lock is simple: the agent only needs the teacher to supply a single trace to learn the combination!

3.3.1.2 MB - Mistake Bound

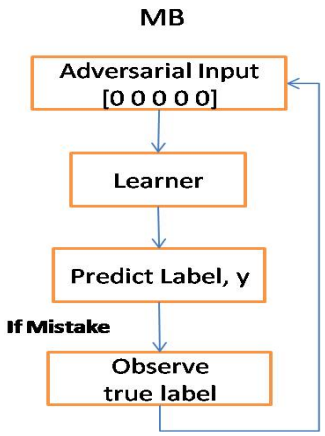


Figure 3.5: The Mistake Bound Protocol.

Mistake Bound [Littlestone (1987)] is a model learning framework which has been previously linked with domains that have a teacher in the learning loop [Angluin (1987); Goldman and Kearns (1992)]. The MB learning protocol for model learning is essentially the same as the KWIK protocol except for three changes -

1. In MB, there is no $\perp$ prediction. The agent must always predict a $\hat{y}_t \in \{0, 1\}$ and receives a true label when it is wrong.

2. MB is only defined for deterministic hypothesis classes, so instead of z_t, the agent will actually see the true label.

3. Efficiency is characterized by a polynomial bound on the number of mistakes made.

This is shown in Figure 3.5. It follows [Li et al. (2008)] that any efficient KWIK learning algorithm for a deterministic hypothesis class can become an efficient algorithm for MB by simply replacing all $\perp$ labels with an arbitrary element from Y.

3.3.1.3 The Mistake Bounded Predictor Framework

We now introduce the following criteria, called a mistake-bounded predictor (MBP)

Definition 3 *A* **mistake-bounded predictor** *(MBP) is an online learner with accuracy parameters ϵ and δ that takes a stream of inputs from set X and maps them to outputs from a set Y. After predicting any $\hat{y}_t$, the learner receives a (perhaps noisy) label z_t produced by an unknown function from a known hypothesis class. An MBP must make no more than a polynomial (in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and some measure of the complexity of the hypothesis class) number of mistakes with probability $1-\delta$. Here, a mistake occurs if, for input x_t, the learner produces $\hat{y}_t$ and $||h^*(x_t) - \hat{y}_t|| > \epsilon$, where h^* is the unknown function to be learned.*

Given below is a model-based RL algorithm (MBP-Agent, Algorithm 2) for the apprenticeship setting that uses an MBP learner as a module for learning the dynamics of the environment.

Algorithm 2 MBP-Agent

The agent knows ϵ, δ, and A and has access to the environment E, teacher T, and a planner P for the domain.
Initialize MBP learners $L_T(\epsilon, \delta)$ and $L_R(\epsilon, \delta)$
for each episode **do**
  $s_0 = E.startState$
  while episode not finished **do**
    $a_t = P.getPlan(s_t, L_T, L_R)$.
    $\langle r_t, s_{t+1} \rangle = E.executeAct(a_t)$
    $L_T.Update(s_t, a_t, s_{t+1}); L_R.Update(s_t, a_t, r_t)$
  end while
  if T provides trace τ starting from s_0 **then**
    $\forall \langle s, a, r, s' \rangle \in \tau, L_T.Update(s, a, s'), L_R.Update(s, a, r)$
  end if
end for

The Apprenticeship Learning Protocol with the MBP agent is shown in Figure 3.6.

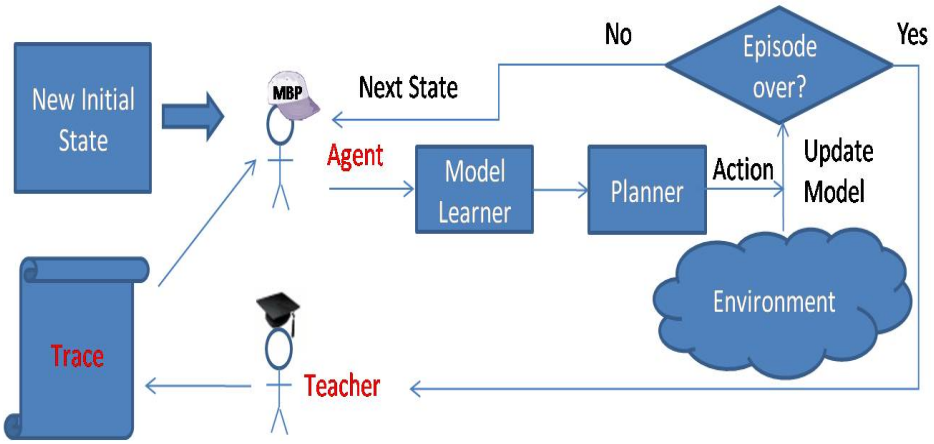


Figure 3.6: The Mistake Bounded Predictor Protocol.

Like KWIK, MBP observations can be noisy, and like MB, the learner is allowed to make a certain number of mistaken predictions. In fact, we can formalize this relationship with the following proposition.

Proposition 1 *Any hypothesis class that is KWIK or MB learnable is MBP learnable.*

Proof 1 *A KWIK learner can be used in its standard form, except that whenever it predicts $\perp$, the MBP agent should pick a $\hat{y}_t \in Y$ arbitrarily. Also, the underlying KWIK learner should not be shown new labels when it does not predict $\perp$, though the MBP agent does receive them. MB learners are defined for deterministic hypothesis classes, so we can assume no noisy observations exist. In this setting, the MB and MBP protocols line up.*

An important note here is that MBP learners never acknowledge uncertainty, our algorithm for the apprenticeship setting believes whatever its model tells it (which could be mistaken). While autonomous learners run the risk of failing to explore under such conditions, the MBP-agent can instead rely on its teacher to provide experience in more "helpful" parts of the state space, since its goal is simply to do at least as well as the teacher. Thus, even model learners that default to pessimistic predictions when little data is available, can be used successfully in the MBP-Agent algorithm.

3.3.2 Efficient Apprenticeship Learning

We now describe how MBP learners are PAC-MDP-Trace as long as the transition function $\mathcal{T}$ and reward function R for an MDP are learnable by an MBP. Algorithm 2 has the following property.

Theorem 1 *An MBP-learner is PAC-MDP-Trace for any domain where the transitions and rewards are MBP learnable.*

The heart of the argument is an extension of the standard Explore-Exploit lemma [Sutton and Barto (1998)], we call the *Explore-Exploit-Explain Lemma.*

Lemma 1 *On each trial, we can define a set of known state,action ($\langle s, a \rangle$) pairs as the ones where the MBP currently predicts transitions accurately. One of these outcomes occurs: (1) The agent will encounter an unknown $\langle s, a \rangle$ (explore) with high probability. (2) The agent will execute a policy π_t whose value is better or not much worse than the teacher's policy π_T (exploit). (3) The teacher's trace with encounter an unknown $\langle s, a \rangle$ (explain) with high probability.*

Lemma 1 proves Theorem 1 because MBP can only make a polynomial number of mistakes, meaning cases (1) and (3) can only happen a polynomial number of times. By any of several simulation lemmata, such as Lemma 12 from [Strehl et al. (2009)], Theorem 1 can be proved. A detailed description of the proof is available in [Walsh et al. (2010)].

In summary, KWIK-learnable models can be efficiently learned in the autonomous RL case, but MB learning is insufficient for exploration. MBP covers both of these classes, and is sufficient for efficient apprenticeship learning, so models that were autonomously learnable as well as many models that were formerly intractable (the MB class), are all efficiently learnable in the apprenticeship setting.

As an example, the combination lock described earlier could require an exponential number of tries using a KWIK learner in the autonomous case, and MB is insufficient in the autonomous case because it does not keep track of what combinations have been tried. But in the apprenticeship setting, the MBP-Agent can get the positive examples

it needs and will succeed with at most n (one for each relevant tumbler) traces. The next section explains this is more detail.

3.3.3 Learning Conjunctions

We now describe an MBP-learnable class (Conjunctions) that is not autonomously learnable. It has been shown in Li et al. (2008) that in KWIK, conjunctions of size $k = O(1)$ are efficiently learnable. The system simply enumerates all n^k conjunctions of this size, predicts $\perp$ unless all the hypotheses agree on a prediction, and eliminates wrong hypotheses. However, when the conjunction is of size $O(n)$, it can result in an exponential number of $\perp$ predictions. This situation arises because negative examples are highly uninformative. In the combination lock, for example, the agent has no idea which of the 2^n settings will allow the lock to be unlocked, so it must predict $\perp$ at every new combination. Note though that if it does see this one positive example it will have learned the correct combination.

In contrast, learners in the MB setting can efficiently learn conjunctions of arbitrary size by exploiting this asymmetry. Specifically, an MB agent for conjunction learning [Kearns and Vazirani (1994)] can maintain a set of literals $l_j \in L_H$ where $l_j = 1$ for every positive example it has seen before. For all $l_j \in L_H = 1$, the agent correctly predicts *true*, otherwise it defaults to *false*. By using such defaults, which KWIK agents cannot, and by only counting the highly informative positive samples, each of which subtracts at least one literal from L_H, polynomial sample efficiency is achieved. Having proved that Conjunctions can be MB-learned, using the earlier proposition, we say that the class is MBP learnable.

This methodology can be extended to learning a number of other classes like Stochastic STRIPS Operators, Deterministic OOMDP operators and many others [Walsh et al. (2010)].

3.4 Experiments and Results

In this section we describe the results of applying the Apprenticeship Learning protocol on a simulated and robot version of the Taxi Domain [Dietterich (2000)]. The domain, shown in Figure 3.7, describes a fixed grid with labeled cells. These cells indicate potential locations where passengers can be picked up and dropped off by a taxi. The thickened lines indicate obstacles between cells. The task is for a taxi, starting in any random cell, to navigate to one of the predefined cells (Y,R,B or G), pick up the passenger and then navigate to another cell (Y,R,B or G) to drop-off the passenger. There are many combinations of pickup and drop-off locations and several transition conditions (obstacles) to be learned. We believe this domain will be a good test for both autonomous agents as well as teacher-guided agents.

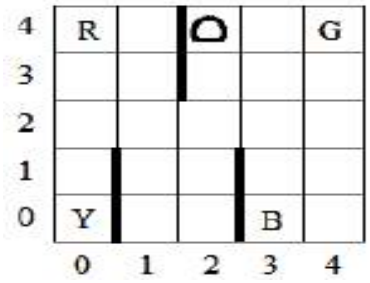


Figure 3.7: The Taxi Domain.

3.4.1 Simulated Taxi World

Our first experiment is in a simulated "taxi" domain (from [Diuk et al. (2008a); Dietterich (2000)]): a 5×5 grid world with walls, a taxi, a "passenger" and 4 possible destination cells. We compare the performance of the Apprenticeship Learning algorithm with one of the current state-of-the-art autonomous approaches, a KWIK-based algorithm known as DOORmax [Diuk et al. (2008b)]. In DOORmax, the world is represented in a deterministic object-oriented manner along with attributes, predicates and effects. There exist a set of conditions over the predicates that the agent uses to learn the effects of each of its actions. In the Taxi world, the agent as mentioned before must

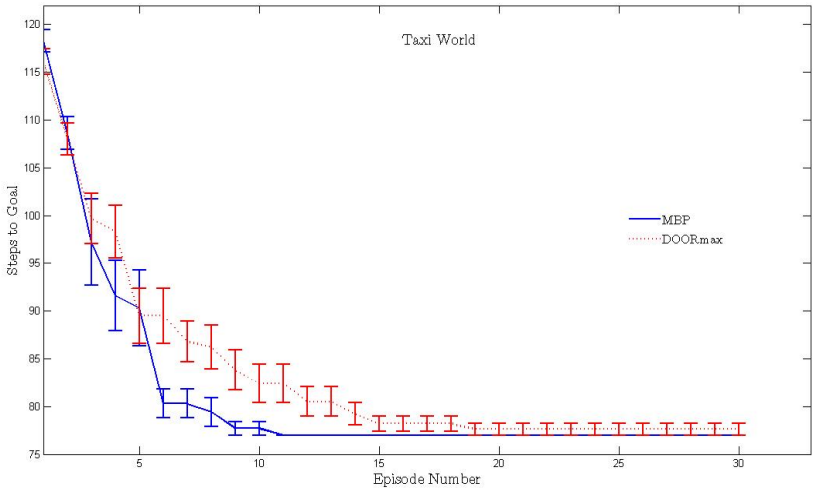


Figure 3.8: A KWIK learner (autonomous) and an MBP learner (apprenticeship) in the Taxi Domain

learn about navigation and the criteria necessary for picking up and dropping off a passenger. The agent has 6 actions, four of which move it one step along the 4 compass directions, one pickup action and one drop-off action.

Figure 3.8 shows our MBP-Agent approach with apprenticeship learning reaching optimal behavior ahead of the autonomous DOORmax approach. The apprenticeship learner acquires an optimal policy faster than the autonomous agent. It is important to note here that the autonomous agent is actively exploring, while the MBP agent solely relies on the traces provided by the teacher. This way there is no unnecessary exploration done by the MBP agent.

3.4.2 Robot Taxi

The second experiment involved a robotic version of the taxi domain. The robot (Figure 3.9) was constructed using a Lego Mindstorms^{TM} kit. It had a standard dual-motor drive train and a third motor controlling its "gripper" that could descend over an object and hold it next to the robot's body, and also raise up to "drop-off" the object. The passenger object was a standard 2.2-inch Rubik's Cube. The objects were identified and tracked using an overhead camera over a $40in \times 40in$ wooden platform.

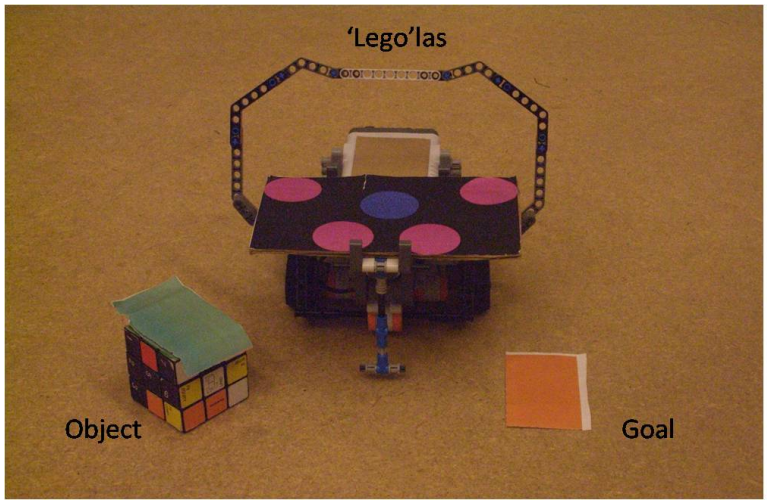


Figure 3.9: The taxi robot, its passenger and the goal.

The discrete actions for the robot were similar to the simulated taxi, but instead of being able to simply strafe to the right or the left, the robot could only move forward, backward, turn right 75 degrees, turn left 75 degrees (both turns moved it slightly forward), and pickup or drop-off the passenger (lower or raise the gripper). For the pickup action, which needs to be quite precise, the atomic action had a subroutine where the robot would move forward slightly and then, if its touch sensor was activated, close the gripper. Thus, the action was only successful if the agent was facing in the correct direction and near the object, otherwise it just moved forward slightly.

In the OOMDP representation used in the experiments, the attributes were the robot's x and y coordinates (discretized by 8-inch intervals) and its orientation, discretized in intervals of 30 degrees. The passenger object was placed in one of 4 predetermined locations (as in the simulated taxi domain) and its destination was one of the 3 other colored locations. The predicates for the OOMDP conditions were *WallToLeft*, *WallToRight*, *WallBehind*, *WallInFront* (all relative to the robot) as well as *HasPassenger*, and *PassengerReachable*, and *AtDestination*. The OOMDP effects were changes in the robot's attributes and the status of the passenger. The rewards of the domain were set as -1 per step and 1 for a successful dropoff at the correct location. Start states for the robot were picked arbitrarily. The MBP-Agent in this task filled in

transitions for conditions it had not witnessed by predicting "no change" (a pessimistic outcome, as described earlier in the paper).

A human demonstrated traces by controlling the robot directly. Unlike the simulated version, the real-world transitions are stochastic (due to noise). So in this experiment, the robot was given all the conditions it might encounter and had to learn the effects and probabilities of the actions. We provided traces after each episode until it completed the task twice on its own from random start states. We ran this experiment in two different settings: one where the dynamics of the environment's borders (physical walls) were given, and one where the robot had to learn about behavior near walls. The first setting contained 6 condition settings and required 5 traces and a total of 159 learning steps. In the harder setting, there were 60 conditions to learn and the agent required 10 traces and 516 learning steps. This experiment demonstrates the realizability and robustness of our apprenticeship-learning protocol in real world environments with sensor imprecision and stochasticity.

3.5 Conclusion

In this case study, we provided a novel approach to learning from a teacher using a reinforcement learning agent. We extended the Apprenticeship Learning Protocol in a number a ways.

- identified a number of model classes (KWIK, MB and combinations of them) that are learnable in the generalized paradigm.
- Provided a bound on the number human interactions required in the form of Valid Traces.
- Allow the agents to learn a better model than its Teacher.
- The agent was allowed to start from random states in the world.
- Traces need not be given in bulk up front.

There is a clear separation between Autonomous agents and ones with a Teacher. This case study reveals the advantages of using a smart learning algorithm as part of

HELP. The agent was able to guarantee efficient learning and the human was guaranteed limited number of interactions. These two results are essential for any human-robot interaction framework.

Chapter 4

Experiments in HELP

The final case study explores another dimension of HELP where the modes of human interactions are varied. The study details two standard reinforcement learning algorithms and describes the different forms of inputs that humans can provide and how they can be used by the algorithm. We instantiate the approach in a car driving simulator.

4.1 Case Study 3 - Extending the Modes of Human Interaction

Imagine a situation where an instructor is teaching people how to drive a vehicle. He shows them how to start and control the car, how to switch gears, parallel park, obey traffic rules and so on; he praises safe driving, discourages dangerous manoeuvres and gives his trainees advice and motivation. We are now faced with the task of being the driving instructor and not teaching a set of human trainees but a set of artificial agents how to drive using the mentioned techniques. At this point, we ask ourselves -

- Is it possible to represent the form of teaching used by the driving instructor in a mathematical model?
- Is there a mathematical equivalent to the words “show”, “praise”, “discourage”, “advice” and “motivation”?
- How can the artificial agent understand and utilize the information provided by the instructor?
- Given that information, can the agent learn and respond similar to a human?

This case study is aimed at exploring the answers to some of these questions. We focus primarily on the different types of information that the human instructor can

provide and how they affect the agent's ability to learn. This problem is investigated in the realm of Reinforcement Learning (RL) where the trainees are considered to be RL agents. An RL agent is one that takes actions in an unknown environment and tries to maximize its overall reward. The RL framework is explained in Figure.4.1. Formally,

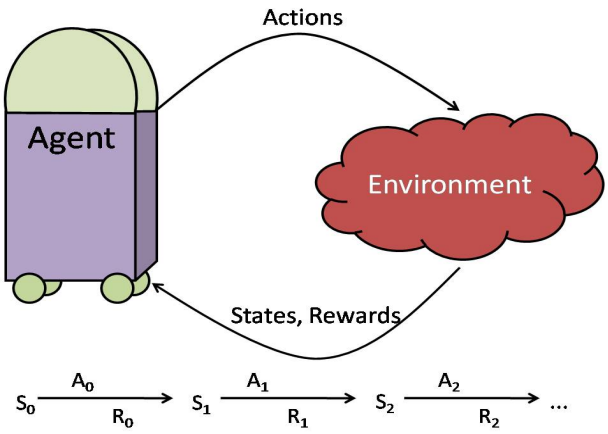


Figure 4.1: Traditional RL Framework.

the basic reinforcement learning model consists of: a set of environment states S; a set of actions A; and a set of scalar "rewards". At each time t, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$. It chooses an action and receives from the environment the new state s_{t+1} and a reward r_t. The environment is formulated as a Markov Decision Process (MDP) where state transitions and rewards probabilities are typically stochastic. Given this setup, the reinforcement learning agent must develop a policy $\pi : S \rightarrow A$ (mapping of states to actions) that maximizes the cumulative reward received. In a traditional RL setting the agent learns autonomously. However in this case study, we introduce a human instructor in the loop and investigate the effects of his interactions on the RL agent.

4.2 Experimental Setup

This section describes the various components of the experiment. We begin by formally introducing the car driving domain and its characteristics, then briefly describing two standard reinforcement learning algorithms and then a note on the different types of

instructors that will be assisting the artificial agent.

4.2.1 The Highway Car Domain

The environment used in this study is a simulated version of the car domain similar to one described in [Abbeel and Ng (2004); Syed et al. (2008)]. A descriptive screen-shot of the simulator is shown in the Figure 4.9. The highway consists of 3 lanes with an additional off-road lane, one on either side (indicated by the green patch). The agent (blue square) is driving on the highway at a constant speed in the presence of oncoming cars (red triangles). Every state of the agent in the driving world is described by using

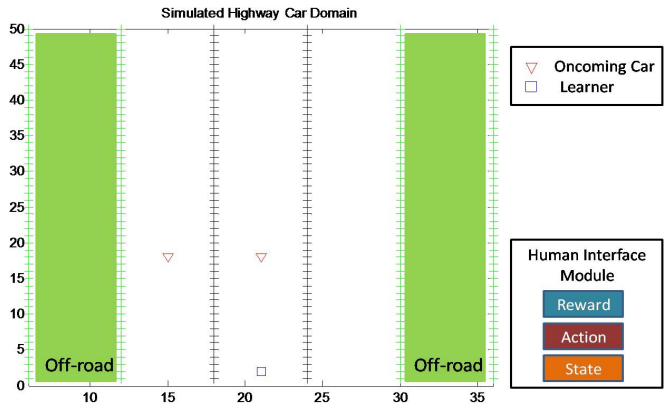


Figure 4.2: Screenshot of the Driving Simulator.

a set of features. We use three features in particular, one indicating whether the agent has collided with an oncoming car, two indicating whether the agent has gone off-road and a feature describing which lane the agent is currently in. The goal of the agent is to drive safely on the highway by learning to avoid collisions with oncoming cars, to avoid going off-road and to maintain lane discipline. To achieve this goal, the agent is equipped with three actions (left, right and stay in place) and has access to a human driving instructor who is assumed to be optimal at the given task. The instructor communicates his knowledge to the agent through a keyboard interface (shown in the figure).

4.2.2 Learning Algorithms

Here we describe the algorithms used by the agent to learn the driving task. We first give a more formal definition of an MDP and then detail two algorithms, Q-learning and R-Max, that have been widely used to solve MDPs.

An MDP is a 5-tuple $\langle S, A, \mathcal{T}, R, \gamma \rangle$ with states S, actions A, transition functions $\mathcal{T} : S \times A \mapsto \Pr[S]$, reward function $R : S \times A \mapsto [R_{min}, R_{max}]$, and discount factor γ. The function we would like to maximize is $\sum_{t=0}^{\infty} \gamma^t R_{at}(s_t, s_{t+1})$, where the actions a_t are chosen according to policy π. We now detail the learning algorithms using the above definition as the foundation.

4.2.2.1 Q-learning

The Q-learning algorithm [Watkins (1989)] is a model free learning algorithm that has been widely used to solve infinite horizon MDPs. Model free indicates that the agent can learn to perform the task without ever having to build a model of the environment in which it is taking actions. Given the formal definition of an MDP, this algorithm focuses on learning which action is optimal for each state. The algorithm computes the "Quality" of a state-action combination: $Q : SXA \rightarrow \mathbb{R}$. The recursive step of the Q-learning algorithm is given by the equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{4.1}$$

Using the above equation, the Q-values are iteratively updated as the agent takes actions in different states and receives rewards. The optimal policy π^* is computed using the formula:

$$\pi^*(s_t) = \arg\max_a Q(s_t, a_t) \tag{4.2}$$

This algorithm has been proven [Watkins (1989)] to converge to optimal policy given the following set of criteria -

- The system must be a deterministic MDP.
- The reward values are bounded by some constant R_{max}.

- The agent visits every possible state-action pair infinite times.

The algorithm variables, α (learning rate) and γ (discount factor) have a significant influence of the rate of convergence of the algorithm. The learning rate determines to what extent the newly acquired information will override the old information and the discount factor determines the importance of future rewards. The algorithm has several advantages: it is easy to implement, is exploration insensitive, one of the most effective model-free algorithm for learning from delayed reinforcement. On the other hand, algorithm does not scale to large complex domains and it takes a very long time to converge.

4.2.2.2 R-Max

R-Max [Brafman and Tennenholtz (2002)] is a model-based reinforcement learning algorithm which is said to attain near-optimal average reward in polynomial time. In contrast to a Q-learner, the R-Max agent maintains a complete, possibly inaccurate model of its environment and acts based on the optimal policy derived from this model. The algorithm initializes the model optimistically implying that all actions in all states return the maximum possible reward. This acts as a driving force causing the agent to explore different parts of the state space by taking different actions and updating its model based on its observations (feedback from the environment).

To summarize the algorithm, it starts with an initial estimate for the model and assumes that all states and all actions return the maximum reward and with probability 1, transition to a fictitious state S_o. On the basis of the current model, an optimal policy is computed using one of several methods. The one used in this implementation is Value Iteration. The transitions and rewards received for every state-action pair are stored and once sufficient information has been acquired, the model is updated and the optimal policy is recomputed. This process repeats. The optimality and convergence proofs of R-Max can be found in [Brafman and Tennenholtz (2002)]. This algorithm has the advantage of actively encouraging exploration and does not require the MDP to be deterministic. A more descriptive form of algorithm is available below:

Algorithm 3 R-Max Algorithm

Input:

- the number of states S, the number of actions A, the upper bound on the reward function R_{max} and the error bound ϵ.

Initialize:

- Construct a model M consisting of $S + 1$ states and A actions. Here the additional state S_o is considered to be a fictitious state.
- Every state and action in the model is initialized with R_{max} (optimistic) and transitions are initialized such that $T_M(S_i, a, S_o) = 1$ for all states S_i and for all actions a.
- Initialize a boolean value *known/unknown* to all state-action pairs indicative of how well the algorithm can predict the outcome given a state-action pair.

Repeat:

- Given a model compute the optimal policy (ϵ close) for the current state and execute it until a new state-action pair becomes known.
- Observe and record the number of times a state-action pair has been visited and the possible outcomes during each visit. Once it crosses a threshold, toggle the boolean value from *unknown* to *known*.
- Repeat.

4.2.3 Forms of Human Input

The role of the instructor in the driving domain is to transfer his knowledge of the task to the artificial agent using natural methods of interaction. When teaching humans, the instructor uses demonstration, motivation, advice, and other such social methods to train the humans. However we have seen from the previous sections that agents understand their world only in terms of states, actions and rewards. The question is then how can a human teach an agent to drive using natural means of interaction and still provide the agent with signals that it can comprehend. This dilemma is illustrated in Figure 4.3. To overcome the barrier, we introduce three types of human instructors:

1. The Motivational Instructor
2. The Directive Instructor
3. The Associative Instructor

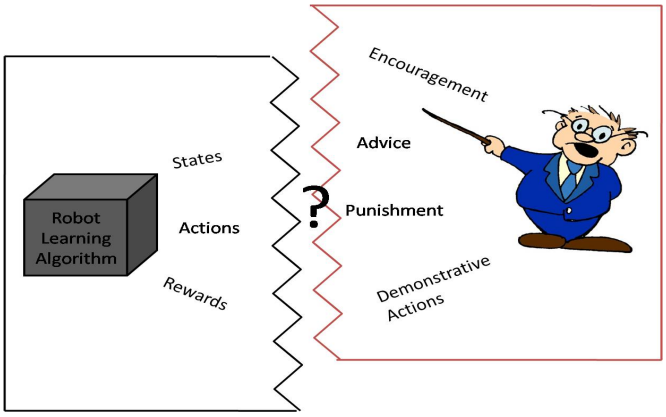


Figure 4.3: Bridging the communication gap between the instructor and artificial agent.

In order to understand the mapping of human interactions to agent-understandable language, let us analyze the contributions of each of the instructors:

- The first instructor (motivational) is one who commends the agent when it performs a desirable action and at the same time punishes the agent for performing sub-optimally. The inputs from this instructor are translated to the learner in the form of *Rewards*.

- The second instructor (directive) is one who directly commands the agent by telling it what to do. He manoeuvres the agent through the highway by controlling its moves until the agent has learned to perform autonomously. The inputs from this instructor are translated to the learner in the form of *Actions*.

- The third instructor (associative) is one who teaches the agent by relating the components of driving that require similar behavior. By doing this he allows the agent to link the different aspects of driving that require the same or associated responses. The inputs from this instructor are translated to the learner in the form of *States*.

Given that we now have a way to bridge the gap between the teacher and the agent, we explore the performance of the agent with each of these styles of instruction.

4.3 Driving Instructor and the Various Modes of Interaction

We have now setup the experiment by describing the car driving domain, the artificial agents and the types of human instructors. In this section, we execute the different styles of teaching on the two simulated agents (Q-learner and R-Max learner) and evaluate the learned policy of the agents. During each trial, we study the effects on the interaction on the two learners as well as on the instructor.

4.3.1 The Motivational Instructor

It is well known that right from the formative years a child is nurtured by its parents using a combination of reinforcement, praise, punishment and anger. Using this interaction, the child learns to do things that will produce positive reactions from its parents. We extend this form of interaction and learning, to the driving domain experiment. The instructor uses a combination of positive and negative rewards to indicate to the agent whether it is driving well or not. In the long run, the agent should learn to take actions that maximize the total received reward. This approach to machine learning has been successfully used by Thomaz et al. (2006) and Knox and Stone (2009). In Thomaz et al. (2006), the human observer uses social modes of interaction like rewards, guidance and advice to communicate the task to the agent. It was successfully applied to a simulated version of Sophie's Kitchen and a real world robot learning experiment. Knox and Stone (2009) describe a framework called TAMER, which stands for Training an Agent Manually via Evaluative Reinforcement. The human observer provides rewards to the agent depending on his assessment of the agent's policy. The underlying algorithm models the human's reward function using a function-approximator and extends this reinforcement to unseen states using a regression algorithm. This framework was successfully applied to a game of Tetris and the Mountain Car domain.

Let us now understand how this instructor teaches the agent to drive. The screenshot of the simulator, shown previously in Figure 4.9, shows a Human Interface Module through which the human provides his input to learner. When the experiment starts, the agent is performing randomly as it has no prior on the task to be learnt. The teacher

observes the agent act and when he feels that agent has performed a suboptimal action (causing it to collide or go off-road), he activates the interface module. At this point, the simulated driving world comes to standstill and awaits the assessment of the current environment configuration. The human uses the keyboard and provides a scalar value that could be positive indicating praise or negative indicating punishment. The agent receives and processes this reward using its reinforcement learning algorithm. For all states where the human does not provide reinforcement, the agent receives a default reward of 0. Figure 4.4 illustrates the various states of the driving world and types of reinforcement given by the optimal teacher.

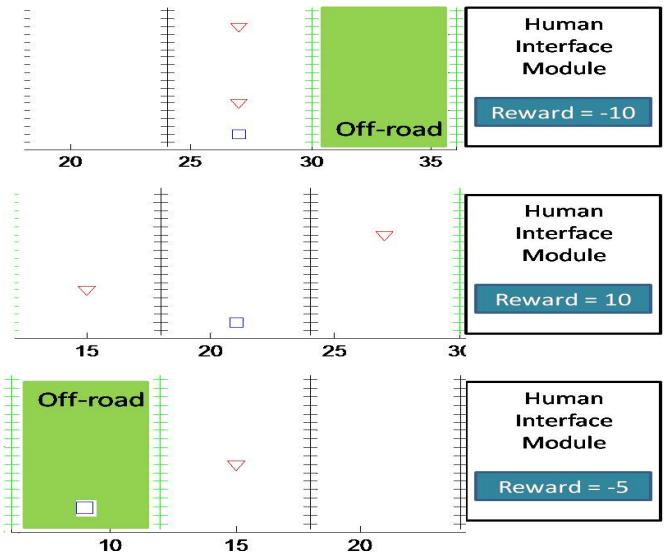


Figure 4.4: Different Configurations of the Driving Domain and the Reinforcement given by the Instructor.

4.3.1.1 Effect on the Learners

Q-learning

In the standard Q-learning algorithm, the agent has access to a reward function. In this trial, the motivational instructor assumes the role of the reward function and provides the agent with appropriate rewards. The Q-learning update now becomes:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[HumanReward_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (4.3)$$

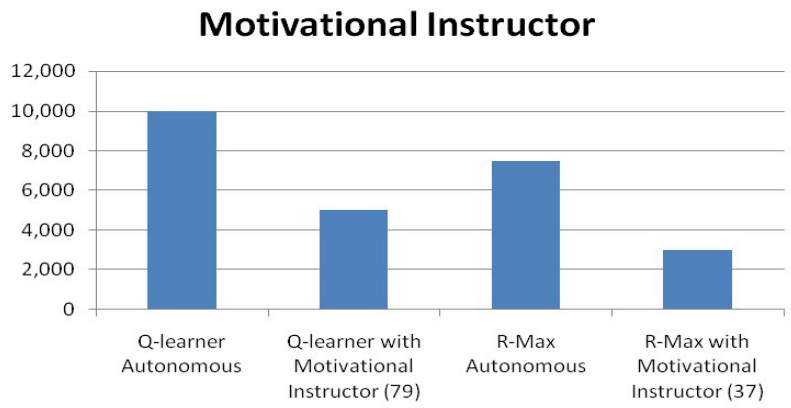


Figure 4.5: Number of steps taken by the learners to reach the optimal policy (with Motivational Instructor).

Every state in the environment has a set of features associated with it. Each reward provided by the human is linked to a state and in turn to a feature configuration set. This association allows the Q-learner to learn the quality of human reward expected for a particular feature set. Thus when the agent encounters an unseen state that has the similar feature values as a state previously encountered, it will act as though they are the same state. This action is then followed by positive or negative reward depending on the human's assessment of the agent's last action. The performance was further tested by modifying the variables of the algorithm.

R-Max

R-Max as described previously is an efficient model based algorithm. The model is initialized optimistically with high reward in all states, for all actions. As the agent acts, it receives reward from the environment and updates its list of state-action pairs visited. In this trial, the reward is no longer given by the environment but given by the instructor. Essentially the change made to the algorithm can be represented as $Environment_{reward} \Rightarrow Human_{reward}$. The agent associates the reward obtained with the feature configuration of concerned state. This information is extrapolated to unseen states and is executed by the agent the next time it updates its model. The advantage with this algorithm over Q-learning is that the agent actively explores and does not

depend on the human to explore. This way learning is much faster. The optimality of the obtained policy was tested by modifying the variables of the algorithm.

Figure 4.5 compares the performance of the two learning algorithms, that learns from a motivational instructor, the task of safely driving a car on a highway with two oncoming cars. The performance is calculated as a measure of the number steps taken before the agent is optimal and the number human interactions (shown in the bracket) that were required to reach an optimal policy.

4.3.1.2 Effect on the Instructor

This experiment was run using 2 human subjects. Their role as the motivational instructor was explained to them and they were given the task of teaching the agents to learn to drive safely. Figure 4.6 shows the performance of the agent across 5 trials with human subjects. The graph shows how the instructor was able to learn *when* and *how* to give the motivation in order to enhance the agent's learning.

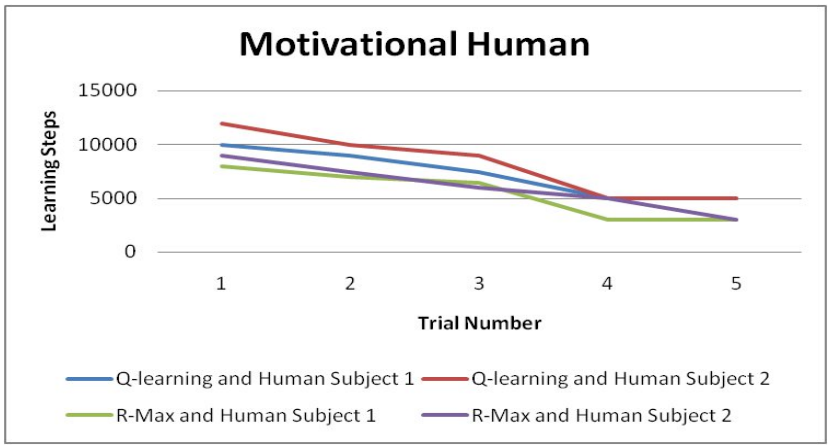


Figure 4.6: Performance of the agent across 5 trials with 2 Motivational human subjects.

Here are a few observations describing the effects of the learning interface on the human:

- The Q-learning agent is slow learner and takes a long time to converge. This can be attributed to that fact that it is exploration insensitive. The human observes that the agent is not exploring, so he modifies his method of providing rewards.

Humans alter their behavior, taking into account the agent's capabilities, and in a manner that makes the agent explores it world.

- Humans tend to give reward even before agent has performed the required action. The human is in effect protecting the agent from imminent danger. Although the human might think that he is correcting the behavior, the Q-learner, since it doesn't have a model, is unable to associate received reward with future states.

- Humans like to see immediate effects of their rewards. If this is not seen, there is a tendency to punish the agent (negative reward) for the mistake. A mistake that the Q-learning agent is still to recognize.

4.3.2 The Directive Instructor

A Directive Instructor is one who teaches the agent by explicitly telling it what set of actions to take given the current configuration of the world. The instructor does not provide a measure of reinforcement but instead a direct action that should be taken. This type of learning is analogous to way the teachers help students solve mathematical problems. They help the children by telling them how to solve the first few set of problems until the students are able to solve them on their own. Behavioral Cloning [Bain and Sammut (1996)] is another similar technique. In this method, learning agents first watch either a human or agent expert execute the required behavior and then try to replicate the patterns of behavior observed (clones). The authors of [Bain and Sammut (1996)] point out some disadvantages to the method of Behavioral Cloning. The first is that the clones were brittle and difficult to understand. Cobot, A Social Reinforcement Learning agent [Jr. et al. (2001)] is an agent that lives in an online community called LambdaMOO [Jr. et al. (2001)]. Within the scope of the online community, it learns in a manner similar to driving agent. Cobot watches other people take actions, watches them interact and forms associations with this knowledge. Using the model it has built, Cobot takes actions in the community, interacts with other people and further refines its model based on the feedback it receives from other users.

The interface of the directive instructor is similar to the previous one where the

communication takes place through the human interface module. Once the driving simulator is started, the agent begins by performing randomly. The instructor, observing the agent, steps in by clicking on the simulator window. The world stops and awaits an directive action as input from the human. The human now has one of three choices, corresponding to the three actions of the agent, stay in place, left and right. The human selects the correct action to be taken and the agent follows that directive. For all states where there has been no human intervention, the agent takes actions autonomously as defined by its policy. This process continues until the agent has learned to perform the task autonomously. Figure 4.7 illustrates the various states of the driving world and types of directive actions given by the optimal teacher.

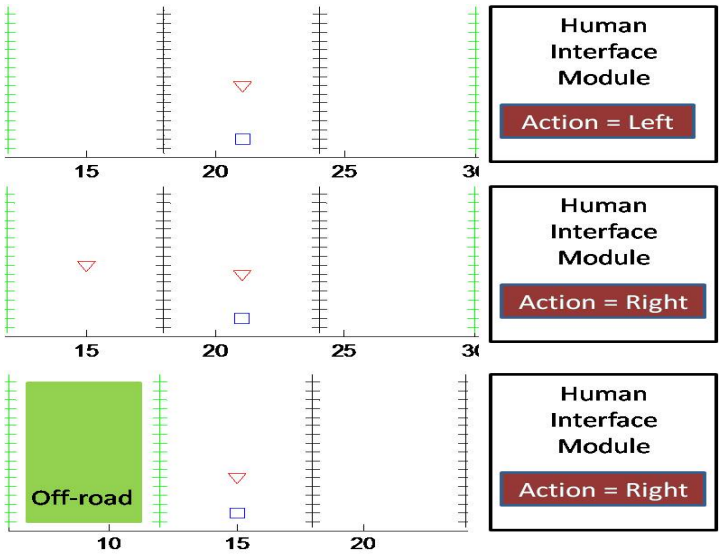


Figure 4.7: Different Configurations of the Driving Domain and the Directive Actions given by the Instructor.

4.3.2.1 Effect on the Learners

Q-learning

The Q-learning agent takes actions by following its policy, represented as $\pi^{(s_t)} = \arg\max_a Q(s_t, a_t)$. It chooses the actions that is said to provide the maximum utility. The Directive Instructor has the ability to overrule the action suggested by the agent's

policy, forcing it to take the teacher's choice of action. The algorithm then uses the reward acquired from taking the new action to update its Q-value. It is important to note here that, the reinforcement now comes from a standard reward function. The human only provides directive actions. In Q-learning, the agent does not actively explore, however since the teacher has direct control of the agent, the agent can be comfortably steered to parts of the driving world where there is relevant information to be learned. Learning tends to be quicker when comparing the Directive Expert with the Motivational Expert. This is attributed to the fact that the directive teacher explicitly controls exploration while the motivator controls exploration implicitly via the cumulative reward received.

R-Max

The R-Max Learner acquires the actions to be performed by computing the optimal policy. As explained previously, this step is completed using Value Interaction. It produces Q-values for every state-action pair and the agent acts by choosing the action that has the maximum value/utility. The instructor has the ability to override the action suggested by the policy and provide a more appropriate action for the current state configuration. The agent uses the reward obtained from taking that action and updates its list of known and visited states. During the next iteration, the recomputed model will encode the new information given by the instructor. We know that the R-Max learner initially builds an optimistic model and starts by actively exploring it. However, when interacting with the Directive Instructor, this tends to be counter-productive as agents sometimes explore parts of the state space where there is nothing relevant to be learned. This in turn results in the instructor having to step in more often just to correct the unnecessary exploration of R-Max learner.

Figure 4.8 compares the performance of the two learning algorithms, that learns from a directional instructor, the task of safely driving a car on a highway with two oncoming cars. The performance is calculated as a measure of the number steps taken before the agent is optimal and the number human interactions (shown in the bracket) that were required to reach an optimal policy.

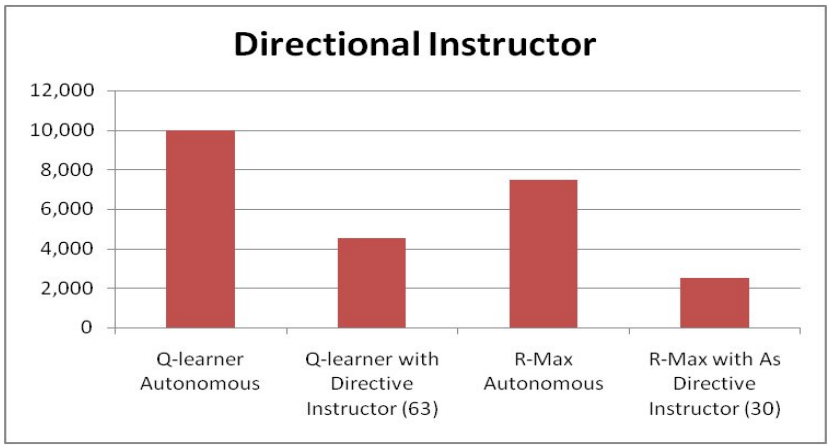


Figure 4.8: Number of steps taken by the learners to reach the optimal policy (with Directional Instructor).

4.3.2.2 Effect on the Instructor

This experiment was run using myself as the human instructor. I found the interaction with the Q-learner to be more engaging as the agent seemed like a student waiting to receive orders from the teacher. The R-Max learner on the other hand tends to explore actively and therefore required numerous interactions with me in order to correct its behavior. Clearly the value of exploration is dependent on the type of information given by the human. Another interesting observation was that when the expert is teaching the agent, it is important for the agent to be a shown a negative reward. The reason for this is that if the agent is acting optimally right from the start, it will always receive high reward. It will think that all its actions are good because it doesn't have a notion of what is bad. Once the instructor purposefully shows a negative reward, it allows the agent to differentiate states with good reward and those with low reward. The policy computed by Q-learning or R-Max will then be able to better characterize the task.

4.3.3 The Associative Instructor

The Associative Instructor uses a novel approach to interact with an artificial agent. In this form of interaction, the human has the task of linking/clustering states that have similar properties, for example, states that require the same action to be performed.

This method although subtle is often seen when humans interact and teach one another. Consider the task of teaching a child to solve basic math problems. Let the number of problems indicates the number of states and the 4 arithmetic operators indicate the number of actions. During the course of interaction, one can imagine the teacher telling the student "Use the same technique that you used to solve Problem 1". The teacher has now become an associative instructor. The teacher has indicated that the current problem (state n) can be solved by the same arithmetic operation (action) that was used to solve an earlier problem (state 1). The student now begins to forms links across the problems in an attempt to understand their implicit similarities. Figure 4.9 illustrates the same analogy in the car driving domain. It shows three different states of the world, all of which have one thing in the common: the agent can avoid a collision if it goes right. For that reason, the instructor is mapping all the three simulator states to a single cluster.

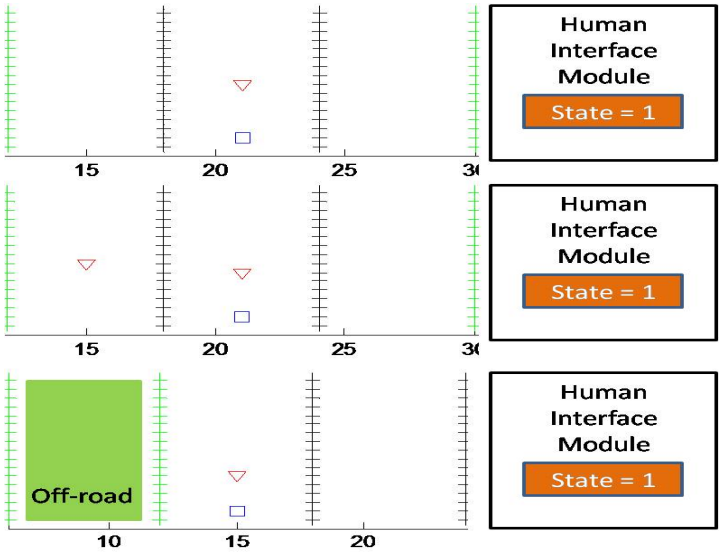


Figure 4.9: Different Configurations of the Driving Domain and the State Mapping given by the Instructor.

Using this method of instruction, it would be possible to cluster large state spaces to a smaller subset characterized by some common features. The human gives the input using the interface module. A question that we would like to answer at this point: "is it necessary for the expert to label every state to a cluster?". The answer is no.

We overcome this by introducing a notion of continuous clustering. To understand this better, consider the following state transition - $S_1, S_2, S_5, S_9, S_{12}$ and S_{28}. At state S_1, the instructor activates the interface module and maps that state to cluster 1. After having done this, all following states are mapped onto cluster 1 until the instructor accesses the interface module and indicates otherwise. In this manner, states S_2, S_5, S_9, S_{12} and S_{28} are sequentially linked to cluster 1 until the teacher activates the interface. In the driving domain, a cluster is defined by the action to be taken and it virtually contains a set of states in which the same action has to be performed. Using this method a large number of states can be clustered with very little interaction from the associative instructor. Intuitively, the instructor has converted the state space of the driving domain to small subset of N clusters. We formally define π^* as the policy of Associative Instructor. In our case, π^* consists of N cluster states each of which is associated with a single action.

4.3.3.1 Effect on the Learners

Q-learning

The Q-learner uses its standard representation to learn the task. When the human steps in, he enters a cluster number (say 2) to which the current state is to be virtually linked. The underlying algorithm now makes note of the link as well as the action taken (say right) in that state. Henceforth, cluster 2 is characterized by the right action and all states in which the right action is to be performed is referenced to cluster 2. Algorithmically, this is achieved by replacing the Q-values of the agents policy, π with Q-values of the instructors's policy π^*. Initially the instructor's policy will be a random matrix, but it gets refined as the agent continues to take actions in the world. There exists a two way communication between the two policies thats lasts long enough until the agent can perform autonomously. As we have noted before, the Q-learner does not actively explore and this has a direct effect on the different states visited. The learner when interacting with associative instructor was able to perform well on small state spaces, but as the driving world grew bigger with more oncoming cars, the Q-learning with its poor exploration strategy was unable to help the instructor.

R-Max

R-Max learner as expected assigns clusters similar to the Q-learner and explores the state space in much more efficient manner. Large state spaces can be easily covered in no time as the algorithm drives visits to new states and the instructor assigns clusters to blocks of these states. Every move initiated by the associative instructors opens a two way channel between the agent's policy and that of the teacher. The states are appropriately linked to clusters and the Q-values are modified in both policies. Value Iteration is performed on this new information and the model produced encodes the state assignments of the instructors implicitly in the form of Q-values. The R-Max learner was found to converge quickly to the optimal policy.

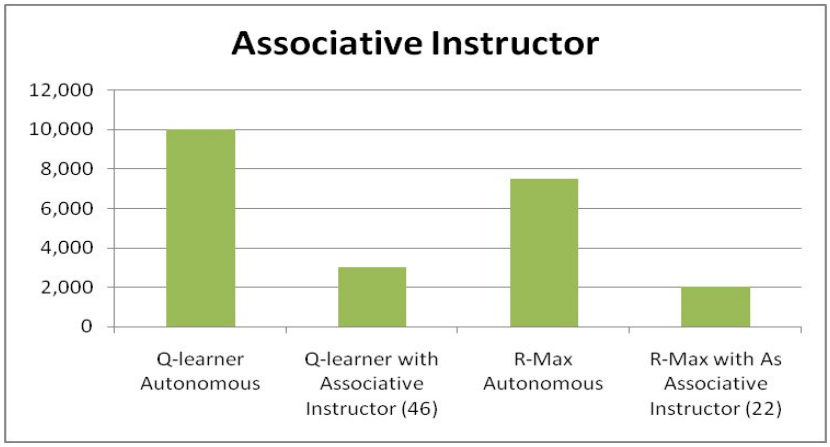


Figure 4.10: Number of steps taken by the learners to reach the optimal policy (Associative Instructor).

Figure 4.10 compares the performance of the two learning algorithms, that learns from a directive instructor, the task of safely driving a car on a highway. The performance is calculated as a measure of the number of collisions encountered before the agent is optimal and the number human interactions that were required to reach an optimal policy.

4.3.3.2 Effect on the Instructor

This experiment was carried out with two human subjects both of whom were unaware of this form of interaction. After providing the preliminary details, the humans played associative instructors to the learning agents for about 3 to 4 mins. During this time, they reported that it was very difficult to assign clusters to states by assigning numerical values. They felt that it was better to express the same thoughts using verbal commands, in a more socially engaging manner. They also reported that the R-Max agent was able to utilize the information they provided better than the Q-learner.

4.3.4 Which Agent?, Which Instructor?

After having described three different modes of interaction, we now ask what type of algorithm along with what type of instructor would be perfect to teach an RL agent to drive a car on a highway. From the plots shown in the previous sections, the R-Max learner seems to the better of the two learners and the Optimal Associative Instructor seems to be the best form of teaching. The drawbacks with respect to the R-Max learner is that it explores every part of the state space even if there is nothing important and the associative instruction method seems to be a very less natural mode of interaction as compared to rewards and actions. We now make a tradeoff between the best learner and the most natural form of interaction - the Motivational Instructor with the R-Max Learner.

In this study, we have extended the standard forms of human interaction to other modes and instantiated it in a car driving simulator. We were able to relate the inputs of a learning algorithm to natural modes of communication that humans are familiar with. This completes the last aspect of HELP in this Thesis.

Chapter 5

Conclusions and Future Work

In this Thesis we have discussed three case studies, each focusing on the different dimensions of HELP:

1. A standard human-robot interaction framework using supervised learning (applied on the Nao Humanoid robot) - Case Study 1.

2. A reinforcement learning algorithm that focuses on efficiently utilizing the human's inputs and reducing the total interaction time - Case Study 2.

3. Varying the modes of human interaction beyond the standard protocols and studying the effects on standard reinforcement learning algorithms - Case Study 3.

In our first case study we introduced a supervised learning framework where a human provides demonstrations of a task through kinesthetic means (direct control over the robot's actuators). The robot using a supervised learning algorithm was able to learn constrained reaching gestures. The Nao robot was able to generalize and solve the task given different placements of the objects using a set of demonstrations from the human.

The second case study focused on the aspect of limiting the number of human interactions with the robot. One can imagine that infinite interactions will allow the robot to completely learn the task. In this study, we introduced a reinforcement learning agent that learned a model of the world by making predictions. The role of the human is to provide a trace (a trajectory from start to goal) that helps the agent correct its mistakes and teaches it something new about the world. We guarantee that efficient learning is possible with only a polynomial number of mistaken predictions by the robot and only a polynomial number of interactions from the human teacher. This work extended the previous protocol such that it can be applied to a wider class of

problems. The system was instantiated in a simulator and a real robot experiment. The results showed that the protocol is efficient as solving the Taxi task and can be applied to a number of other domains.

In the third case study we extend the modes of human interaction and apply them to two reinforcement learning approaches - model free (Q-learning) and model based (R-Max) algorithms. The basic forms of input taken by these algorithms is *states*, *actions* and *rewards*. We designed a system where we linked these inputs to natural means of communication between humans. It was instantiated in a driving simulator and we found the best learner was the model based R-max learner and the best teacher was the Associative (state-clustering) Instructor.

During the course of these case studies, there were several challenges associated with their functioning. Here we list some of the primary challenges with incorporating human guidance as a part of learning protocols:

- Task complexity and representation

  In many situations, the task by itself is very complex. For example, a task like balancing on a rope, traversing uneven terrains etc. These tasks are often in continuous domains, have a very large state space, inherent stochasticity and are complex to learn when represented by conventional algorithms. In such tasks it is important to find a representation of the task that can be easily interpreted and learned by an artificial agent.

  When considering the task, it is also important to take into account the limitations of robot. The sensors and actuators of robots vary to a very great deal and therefore it requires a careful analysis of the capabilities of the hardware before assigning a task.

- Noisy human teachers

  Human teachers in most cases act as noisy sources of data. They are inconsistent, unreliable and are prone to errors. When using their inputs for learning, it is important to take these factors into account. The algorithm must be robust to noise and must be able to generalize the data provided by the humans. During

the course of the studies, it was noticed that humans loose attention fast and are impatient. The information they provide deteriorates as the number of interactions with the agent increases. It is therefore essential that the human have limited and yet informative interactions with the agent.

Sometimes the human may not know how to solve the task. The teacher might also be in a position where they are unable to evaluate the performance of the artificial agent. In these cases, the human cannot be considered to be a good source of information to help the agent learn. As an alternative, if there is no knowledgeable human present, the agent can resort to autonomous methods of learning.

- Efficiency of the Interaction Language

  The interaction language consists of the human mode of communication and the learning algorithm used by the agent. They have a significant effect on agent's learning. The algorithm must be chosen such that it can learn and generalize over the available representation of the task. Following this, the mode of interaction must be chosen such that it provides sufficient information about the task, can be provided by a human naturally and can be understood by the learning algorithm (agent).

At this point, we say that HELP has significant advantages over conventional autonomous approaches. Using HELP, we can extend prior work and design novel protocols by which artificial agents can learn complex tasks with the help of an informed human. As part of future work, we can direct our research towards:

- Investigating further modes of interaction, utilizing more social cues that humans are accustomed to (for example using verbal commands with emotions).
- Allowing the agent to actively interact with the human and requesting specific information about the task that the agent feels will help it generalize better.
- Learning to generalize when taught multiple tasks from multiple teachers.

- Transfer Learning - Exploring the ability to transfer the learned behavior of one robot to another. This can have significant impact on the learning time in multi-agent systems.

At the end, there is no definitive rule stating the best algorithm and the best mode of interaction. The purpose of this research in HELP has been to explore new approaches to the field of human-robot interaction and to provide a glimpse of the capabilities of novel approaches that satisfy both efficient learning as well as a satisfied human.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *ICML*.

Abbeel, P. and Ng, A. Y. (2005). Exploration and apprenticeship learning in reinforcement learning. In *ICML*.

Angluin, D. (1987). Queries and concept learning. *Machine Learning*, 2(4):319–342.

Argall, B., Chernova, S., Veloso, M. M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483.

Asmuth, J., Li, L., Littman, M. L., Nouri, A., and Wingate, D. (2009). Pac-mdp reinforcement learning with bayesian priors.

Atkeson, C. G. and Schaal, S. (1997). Robot learning from demonstration. In *ICML*, pages 12–20.

Bain, M. and Sammut, C. (1996). A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129. Oxford University Press.

Baltes, J., Lagoudakis, M. G., Naruse, T., and Shiry, S., editors (2010). volume 5949 of *Lecture Notes in Computer Science*, Berlin. Springer.

Beckers, R., Holland, O. E., and Deneubourg, J. L. (1994). From local actions to global tasks: Stigmergy and collective robotics. pages 181–189. MIT Press.

Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008). Robot programming by demonstration. In *Springer Handbook of Robotics*, pages 1371–1394.

Brafman, R. I. and Tennenholtz, M. (2002). R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.

Breazeal, C. and Thomaz, A. L. (2008). Learning from human teachers with socially guided exploration. In *ICRA*, pages 3539–3544.

Calinon, S. (2007). *Continuous extraction of task constraints in a robot programming by demonstration framework*. Phd thesis, EPFL Lausanne University.

Calinon, S. and Billard, A. (2007). Incremental learning of gestures by imitation in a humanoid robot. In *HRI*, pages 255–262.

Calinon, S. and Billard, A. (2008a). A probabilistic programming by demonstration framework handling constraints in joint space and task space. In *IROS*, pages 367–372.

Calinon, S. and Billard, A. (2008b). A probabilistic programming by demonstration framework handling constraints in joint space and task space. In *IROS*, pages 367–372.

Chalodhorn, R., Grimes, D. B., Grochow, K., and Rao, R. P. N. (2007). Learning to walk through imitation. In *IJCAI'07: Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 2084–2090.

Chernova, S. and Veloso, M. M. (2007). Confidence-based policy learning from demonstration using gaussian mixture models. In *AAMAS*, page 233.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res. (JAIR)*, 13:227–303.

Diuk, C., Cohen, A., and Littman, M. L. (2008a). An object-oriented representation for efficient reinforcement learning. In *ICML*.

Diuk, C., Cohen, A., and Littman, M. L. (2008b). An object-oriented representation for efficient reinforcement learning. In *ICML*.

Dominguez, S., Casanova, E. Z., García-Bermejo, J. G., and Pulido, J. (2006). Robot learning in a social robot. In *SAB*, pages 691–702.

Fong, T., Nourbakhsh, I. R., and Dautenhahn, K. (2003). A survey of socially interactive robots. *Robotics and Autonomous Systems*, 42(3-4):143–166.

Goldman, S. A. and Kearns, M. J. (1992). On the complexity of teaching. *Journal of Computer and System Sciences*, 50:303–314.

Holland, O. (2003). The future of embodied artificial intelligence: Machine consciousness? In *Embodied Artificial Intelligence*, pages 37–53.

Jones, R. M. and VanLehn, K. (1991). A computational model of acquisition for children's addtion strategies. In *ML*, pages 65–69.

Jr., C. L. I., Kearns, M. J., Singh, S. P., Shelton, C. R., Stone, P., and Kormann, D. P. (2006). Cobot in lambdamoo: An adaptive social statistics agent. *Autonomous Agents and Multi-Agent Systems*, 13(3):327–354.

Jr., C. L. I., Shelton, C. R., Kearns, M. J., Singh, S. P., and Stone, P. (2001). Cobot: A social reinforcement learning agent. In *NIPS*, pages 1393–1400.

Kang, S. B. and Ikeuchi, K. (1995). A robot system that observes and replicates grasping tasks. In *ICCV*, pages 1093–1099.

Kaplan, F., Oudeyer, P.-Y., Kubinyi, E., and Miklósi, A. (2002). Robotic clicker training. *Robotics and Autonomous Systems*, 38(3-4):197–206.

Kazman, R. (1991). Babel: A psychologically plausible cross-linguistic model of lexical and syntactic acquisition. In *ML*, pages 75–79.

Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA.

Kemp, C., Goodman, N., and Tenenbaum, J. B. (2007). Learning and using relational theories. In *NIPS*.

Knox, W. B. and Stone, P. (2009). Interactively shaping agents via human reinforcement: the tamer framework. In *K-CAP*, pages 9–16.

Li, L., Littman, M. L., and Walsh, T. J. (2008). Knows what it knows: A framework for self-aware learning. In *ICML*.

Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318.

Lopes, M., Melo, F. S., and Montesano, L. (2009). Active learning for reward estimation in inverse reinforcement learning. In *ECML/PKDD (2)*, pages 31–46.

Martin, J. D. and Billman, D. (1991). Variability bias and category learning. In *ML*, pages 90–94.

Mataric, M. J. (1997). Reinforcement learning in the multi-robot domain. *Auton. Robots*, 4(1):73–83.

Montesano, L., Lopes, M., Bernardino, A., and Santos-victor, J. (2008). Learning object affordances: From sensory motor coordination to imitation.

Nehaniv, C. L. and Dautenhahn, K. (2001). Like me?- measures of correspondence and imitation. *Cybernetics and Systems*, 32(1):11–51.

Neu, G. (2007). Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proc. UAI*.

Ng, A. Y. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In *In Proc. ICML*, pages 663–670.

Nicolescu, M. N. and Mataric, M. J. (2001). Learning and interacting in human-robot domains. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 31(5):419–430.

Niemueller, T. (2009). *Developing A Behavior Engine for the Fawkes Robot-Control Software and its Adaptation to the Humanoid Platform Nao*. Diploma thesis, RWTH Aachen University, Germany.

Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. (2009). Learning and generalization of motor skills by learning from demonstration. In *International Conference on Robotics and Automation (ICRA2009)*.

Pazzani, M. (1994). Computational models of human learning. Guest Editorial.

Ramachandran, D. and Amir, E. (2007). Bayesian inverse reinforcement learning. In *In Proc. IJCAI*, pages 2586–2591.

Ratliff, N. D., Bradley, D. M., Bagnell, J. A., and Chestnutt, J. E. (2006). Boosting structured prediction for imitation learning. In *NIPS*.

Schohn, G. and Cohn, D. (2000). Less is more: Active learning with support vector machines. In *ICML*, pages 839–846.

Seifert, C. M., Hammond, K. J., Johnson, H. M., Converse, T. M., McDoughal, T. F., and Vanderstoep, S. W. (1994). Case-based learning: Predictive features in indexing. *Machine Learning*, 16(1-2):37–56.

Shultz, T. R., Mareschal, D., and Schmidt, W. C. (1994). Modeling cognitive development on balance scale phenomena. *Machine Learning*, 16(1-2):57–86.

Strehl, A. L., Li, L., and Littman, M. L. (2009). Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, 10(2):413–444.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. L. (2006). Pac model-free reinforcement learning. In *In: ICML-06: Proceedings of the 23rd international conference on Machine learning*, pages 881–888.

Subramanian, K. (2010). Task space behavior learning for humanoid robots using gaussian mixture models. In *AAAI Student Workshop*.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Syed, U., Bowling, M., and Schapire, R. E. (2008). Apprenticeship learning using linear programming. In *In Proc. ICML*, pages 1032–1039.

Thomaz, A. L., Hoffman, G., and Breazeal, C. (2006). Experiments in socially guided machine learning: understanding how humans teach. In *HRI*, pages 359–360.

Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1-2):25–46.

Walsh, T., Subramanian, K., Littman, M., and Diuk, C. (2010). Generalizing apprenticeship learning across hypothesis classes. In *ICML*.

Watkins, C. J. (1989). Learning from delayed rewards. Technical report.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *In Proc. AAAI*.