Imitation Learning from Multiple Demonstrators Using Global Vision

A thesis presented

by

Jeff Allen

to

The Department of Computer Science

in partial fulfillment of the requirements

for the degree of

Master of Science

in the subject of

Computer Science

The University of Manitoba

Winnipeg, Manitoba

June 2009

Thesis advisor

John E. Anderson

Author

Jeff Allen

Imitation Learning from Multiple Demonstrators Using Global Vision

Abstract

Imitation learning enables a learner to expand its own skill set with behaviours that it observes from others. Most imitation learning systems learn from a single class of demonstrators, and often only a single demonstrator. Such approaches are limited, however: in the real world, people have varying levels of skills and different approaches to solving problems, and learning from only one demonstrator would be a very limited perspective. In the context of robots, very different physiologies make learning from many types of demonstrators equally important. A wheeled robot may watch a humanoid perform a task, for example, and yet not be able to perfectly approximate its movements (e.g. stepping over small obstacles).

This thesis describes an approach to learning a task by observing demonstrations performed by multiple heterogeneous robots using global (overhead) vision, incorporating demonstrators that are different in size, physiology (wheeled vs. legged), and skill level. The imitator evaluates demonstrators relative to each other, which gives it the ability to weigh its learning towards the more skilled demonstrators. I assume the imitator has no initial knowledge of the observable effects of its own actions, and begin by training a set of Hidden Markov Models (HMMs) to map observations to

actions. These HMMs provide a low-level basis for interpreting the observations of others. I then use forward models to construct more abstract behaviours that bridge the differences between highly heterogeneous agents. This approach is evaluated in the domain of robotic soccer, where it is found that the imitator can weigh its learning towards skilled demonstrators regardless of physiology.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to begin by thanking my advisor, my committee, my parents (thanks to my dad for helping me redesign the imitating robot), my friends, and all the people who have supported me along the way. Most of all I would like to thank Heather, for always being there for me.

This thesis is dedicated to Heather. You are my love, my life, and my inspiration. I love you with all my heart, and I always will.

Chapter 1

Introduction

Imitation learning - the ability to observe demonstrations of behaviour and reproduce functionally equivalent behaviour with ones own abilities - is a powerful mechanism for improving the abilities of an intelligent agent. Evidence of learning from the demonstrations of others can be seen in primates, birds, and humans [Demiris and Hayes, 2002; Matarić, 2002; Billard and Matarić, 2000]. From an Artificial Intelligence (AI) perspective, this is attractive because of its potential for dealing with the general problem of knowledge acquisition: instead of programming a robot for each individual task, robots should ultimately be able to gather information from human demonstrations [Matarić, 2000; Nicolescu and Matarić, 2003; Breazeal and Scassellati, 2002a], or from one another [Anderson et al., 2004b; Breazeal and Scassellati, 2002a; Riley and Veloso, 2001] with the result that the robot's performance at that task improves over time. Additionally, demonstrations do not have to be active teaching exercises: the imitator can simply observe a demonstrator with no communication necessary. That is, the demonstrator does not even need to be aware that it is being

observed.

Simply memorizing one precise instance of a behaviour and repeating it in exactly the same manner is not particularly useful (Matarić [2002] refers to this as *mimicry*). An agent mimicking a single instance of a demonstration cannot generalize it to a larger context or perform it in any way other than precisely the fashion in which it was demonstrated. To make imitation learning useful, an agent must first have an understanding of its own primitive motor skills, observe the demonstration and its outcomes, and ultimately interpret these within the context of its own primitives. In doing so, the agent develops new motor skills by creating hierarchical combinations of primitives [Matarić, 2002], providing a deeper understanding of the imitated behaviour.

Imitation learning is a supervised learning method, as it is receiving input from its demonstrators. The imitator also compares its own output in the form of its actions to the actions of the demonstrators to evaluate behaviours that it has learned. This is in contrast to unsupervised learning, which eliminates the concept of a teacher and requires that learners form and evaluate concepts independently [Luger, 2005].

True imitation is complicated by the fact that no demonstration will be performed *exactly* the same twice. An agent must be able to integrate multiple demonstrations to see a broad range of the ways in which any complex task can be performed, and the imitative process must be sophisticated enough to filter out small differences from the main body of the skill to be learned. In any real world setting, this will be complicated by the fact that multiple demonstrations will likely be performed by different agents. Arguably this *should* be the case, since seeing the full range of ways in which a task

could be accomplished is faster than the learner discovering these itself, and different agents will likely perform a task in different ways.

When working with multiple demonstrators, imitation learning involves two major problems. The first of these is that beyond variations in how they perform a task, demonstrators may have significantly different levels of skill. In soccer, for example, a player demonstrating a given move may be an expert who has played for many years, and another might be a beginner that is not much more skilled than the learner itself. In a situation such as this, the learner must be able to differentiate between the two. Beyond differentiating, a good learner will still learn what it can even from a non-expert demonstrator - increasing the breadth of its experience and the range of its sources of knowledge - as opposed to ignoring non-experts completely.

The second problem occurs when the imitator and its demonstrators have *heterogeneous physiologies* (distinct differences such as body types or sizes). Humans naturally deal with heterogeneous demonstrators. Even a small child can imitate the actions of an animal that is not bipedal, for example. If a child's first exposure to the game of frisbee is through observing a dog catching a frisbee in its mouth, when the frisbee is thrown to the child they will likely attempt to catch it in their hand instead. This way they are using the skills that are natural and available to them to complete the task, even if the demonstration displayed a different set of skills. In a robotic environment, physiological differences are generally much more broad. Robots have been developed for many purposes, and consequently differ in size, control programs, sensors and effectors. These differences result in a broader range of ways in which a single activity can be performed. A humanoid robot (or any bipedal robot), for

example, can step over obstacles that a wheeled robot cannot, but might trip over low obstacles that a wheeled robot could simply drive over. In order to increase the performance of a learner and allow it to learn from whatever demonstrators happen to be available (ultimately, a mixture of humans and other robots), overcoming differences in physiology is absolutely necessary [Nehaniv and Dautenhahn, 2000].

This thesis presents work toward overcoming both of these problems. I present a framework for imitation through vision, which models multiple demonstrators with no prior knowledge of their skill or physiology. Current imitation learning systems tend to lump all teachers together, making it impossible to determine the quality of what is learned from each teacher [Price and Boutilier, 2003; Calinon and Billard, 2007]. Having a rough idea of the competence of your teachers is very useful, especially if you are being taught one task by many teachers. People trust the teachers that have proven to be skilled at the task being learned, and mistrust teachers that seem to be less skilled. Individually modelling its teachers gives an imitation learning robot the ability to compare the quality of the teachers relative to each other. This enables the robot to be more resistant to bad demonstrations and focus on learning the desired task, as well as adapt to heterogeneous demonstrators.

The experimental domain I have chosen to ground my model is robotic soccer, a common domain in robotics because it presents most of the complex problems associated with intelligent mobile robotics, while remaining understandable to those outside the area. In my evaluation, an imitating robot learns to shoot the soccer ball into an open goal from a range of demonstrators that differ in size and physiology (humanoid vs. wheeled robots), as well as differences in skill level. While this problem

may seem trivial to a human adult, it is quite challenging to an individual that is learning about its own motion control. Manoeuvring behind the soccer ball and lining it up for a kick is a difficult task for an autonomous agent to perform, even without considering the ball's destination - just as it would be for a young child. It is also a task where it is easy to visualize a broad range of skills (demonstrators that have good versus poor motor control), and one where heterogeneity matters (that is, there are visual differences in how physiologically-distinct robots move).

This chapter introduces this topic, outlines my motivations in pursuing this work, defines terms necessary to understand my work, and presents a general outline of my approach. It then presents the research questions addressed by the thesis and an outline of the remainder of the thesis itself.

1.1 Motivation

The inherent problem with learning from one teacher is that the learner isn't necessarily learning how best to perform a task: rather, it is learning how to perform that task *in the same manner as its single teacher*. This means that the learning robot learns the good skills, but also the bad. An extreme example in the soccer domain is a teacher driving towards a goal, and instead of travelling in a straight line, doing a pointless loop in the middle of the trip in each demonstration. An imitation learner would learn to approach the goal in exactly the same way as its teacher, wasting time with the same looping action. If the imitation learner instead can integrate the demonstrations of many teachers, it will have the opportunity to see examples without this pointless loop, and filter this from its learned behaviour. The robot

therefore has the opportunity to learn the good and ignore the bad. Learning from multiple teachers enables the imitation learner to learn the most efficient method for obtaining the same major environmental effects (such as shooting a goal). This allows it to amalgamate the most effective observed actions into its own unique repertoire of actions.

The repertoire of available actions can vary greatly between physiologically different robots. This means that control programs for robots often need to be developed specifically for each physical robot type. The time taken to create or adapt a control program for a particular robot physiology is often wasted when robots are abandoned in favour of newer models, or different designs (e.g. switching from a wheeled explorer robot to a robot that has tank treads). A learning system needs to be able to learn from others that are physiologically different than the imitator if the knowledge of various demonstrators is to be passed on. It is unreasonable to expect imitators to be restricted to a homogeneous collection of demonstrators. Learning should be robust enough to allow any demonstrator to work, and should benefit correspondingly from a heterogeneous breadth of demonstrators. It may be possible to discover and adapt elements of a performance by a physically distinct demonstrator that have not yet been exploited by demonstrators of the same physiology. Further, imitating robots that can learn from any type of demonstrator can also learn from robots that developed their control programs through imitation. Imitation can therefore provide a mechanism for passing down knowledge between generations of robots, allowing researchers to focus on other problems, rather than spending a great deal of their time developing the base control programs for each new model of robot.

Being able to learn from multiple physiologically-distinct sources is particularly useful when a robot is a new addition to an existing team. Provided the learner can overcome differences in skill and physiology, it can learn more quickly how to optimally perform the tasks required by observing and imitating all of its teammates at the same time. If a replacement robot is needed, it can learn from the robots that it is to replace, preserving the experience that the original robot(s) acquired.

1.2 Terminology

Before moving from the motivation of my thesis to a description of the research it entails, there are a number of common English terms used in imitation learning and robotics in general whose meanings in the context of this thesis should be made explicit.

Mimicry[Matarić, 2002] is the exact copying of movement patterns of the demonstrator to achieve the same environmental effects.

Imitation, on the other hand, is described by [Matarić, 2002, pg. 395] as "...a more complex process capable of creating arbitrary new motor skills by composing complex hierarchical combinations of primitives and other already learned skills".

The *imitator* or *imitation learner* is a learning system that uses imitation learning. Approaches to imitation learning itself vary, but always involve observation of demonstrations and attempts to recreate and learn from those demonstrations.

The *demonstrators* are the individuals whose performance is to be imitated. They may be teachers in the sense that the demonstration is explicitly being provided to learn from, or simply neutral performers going about their activities without regard

to the learner.

Robots that use vision for sensing can use either *local* or *global* vision approaches. In a local vision approach, the robot's vision and visual interpretation software are self-contained (e.g. a camera and embedded system on-board). Each robot thus has its own perspective, and significant resources must be devoted to interpreting this perspective. In a global vision approach, a single camera captures the view of the entire environment (such as an entire soccer field from a spectator's perspective, rather than a player's) and a vision server interprets this perspective and sends global information (e.g. cartesian coordinates of objects of interest) to robots that subscribe to the server. Each robot in a global vision approach thus has the same perspective and does not have to perform local vision processing.

Robots that are distinct from one another in terms of sensory, effectory, or decision-making abilities are referred to as *heterogeneous* robots. In the context of this thesis, I focus mainly on physical differences such as method of locomotion or physical size. There are also sensory differences in that one of my demonstrator types uses local vision for its own control system (while the others use global vision), and control differences in that the physical differences between robots require changes in sensori-motor control as well.

The *environment* is where demonstrations take place, and where the imitation learner uses skills it has acquired from demonstrations.

The imitator's *primitives* are the imitator's available actions before any learning has taken place. They are called primitives because they are used as low-level building blocks to create more complex motions [Weber et al., 2000].

Behaviours are compositions of the imitator's primitives (and, recursively, other behaviours). While primitives usually only concern themselves with how they change the imitator's position, behaviours usually take additional environmental changes into effect, such as changes in other objects' positions (e.g. other robots, a ball, etc).

While additional technological terms will be introduced as I describe my work in detail, these provide the basic definitions necessary to understand a summary of my approach and the literature review that follows this chapter.

1.3 Method

Chapters 3 and 4 describe my research on imitation learning with multiple heterogeneous demonstrators, in terms of a framework for imitation learning and its implementation. Here I provide a brief overview of my approach to establish a context for the literature review in Chapter 2.

In my research the robot imitator is a two-wheeled robot built from a Lego Mindstorms kit. One of the three robot types used for demonstrators is physically identical (i.e. homogeneous) to the imitator, in order to provide a baseline to compare how well the imitator learns from heterogeneous demonstrators. Two demonstrators that are heterogeneous to the imitator are also used. The first is a Bioloid humanoid robot which uses a cellphone for vision and processing. The choice of a humanoid was made because it provides an extremely different physiology from the imitator in terms of how motions made by the robot appear visually. It should thus be a significant challenge to a framework for imitation learning in terms of adapting to heterogeneity. The humanoid also uses local vision as opposed to global vision for its own control model,

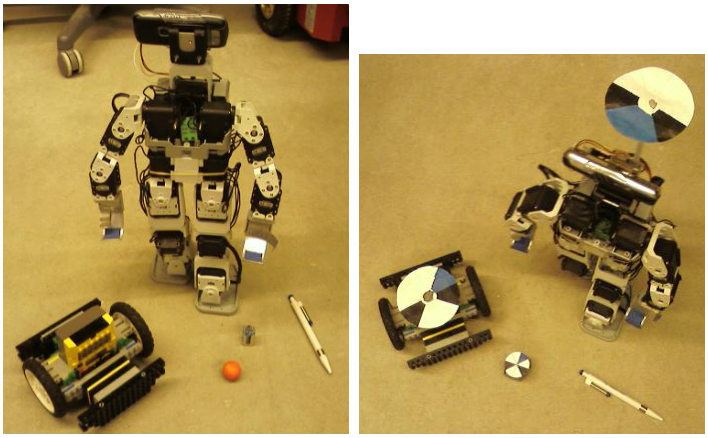

Figure 1.1: Two views of the heterogeneous robots used in this work (a standard ballpoint pen is used to give a rough illustration of scale).

which makes for a distinction in sensing and the impact of this on the robot's control model as well. It should thus be a significant challenge to a framework for imitation learning. The third demonstrator type is a two-wheeled Citizen Eco-Be (version I) robot which is about 1/10 the size of the imitator. This was chosen because while its physiology is similar, the enormous size difference makes for a different extreme of heterogeneity than the challenge presented by a humanoid robot. The robots used can be seen in Figure 1.1 with the visual tracking markers used for the global vision system in the right image. The robots are described in greater detail in Section 4.1.

To keep the scope of this thesis reasonable, the imitation learning robot only observes one demonstrator at a time, with the demonstrated task being that of shooting a ball into an empty goal, similar to a penalty kick in soccer. This task should allow for enough variation between approaches for the different skill levels of the demonstrators to be determined and modelled by the imitator. To ensure demonstrators are of varying skill, I use robots that have been previously used by our laboratory

in robotic soccer competitions (e.g. [Anderson et al., 2004a; Baltes et al., 2008]) as skilled demonstrations, as well as simple control programs (e.g. simply drive at the ball), for poorly skilled demonstrations. All knowledge of the task to be learned is gained by observing the demonstrators. No communication between the imitator and its demonstrators is allowed (or necessary). While this might be used to supplement imitation learning, my goal in this work is to learn solely through imitation.

To observe the imitator and demonstrators, my approach uses *Ergo*, a global vision system from the University of Manitoba Autonomous Agents Laboratory [Baltes and Anderson, 2007]. Ergo provides information about objects in three-dimensional space, such as orientation, location, and velocity. My approach uses the x and y coordinates of the demonstrators, imitator, and ball, as well as the orientations of the imitator and demonstrators. This data is sufficient for the imitator to learn the chosen task from the demonstrations.

An imitation learning system must begin with primitives, and in my implementation I have defined these as the atomic motor commands available to the wheeled imitator robot (*forward, backward, left, right* and *stop*). To properly imitate others, especially those of differing physiologies, an imitation learner must be able to recognize its own primitive actions from visual data. That is, it must understand the visual outcomes of its own primitive actions so that it can understand its observations of others in terms of these.

To develop an understanding of its own actions, the imitator starts out by collecting visual data of the outcomes of its own primitive actions using the Ergo vision system, by executing primitives on the field and creating a mapping between these

and the visual changes that result. The raw vision frames obtained as the robot moves on the field are converted into vectors that represent the change in position of the imitator between the first frame of the primitive action and the current frame. Each vector contains data relating to the x and y coordinates, as well as the orientation of the imitator. These frame change vectors are then clustered using the k-means algorithm [Hartigan and Wong, 1979]. This clustering generalizes the visual changes between frames during a portion of the execution of a primitive, such as removing the specific changes in the x and y coordinates involved. The result is a codebook of discrete symbols used to classify frame change vectors. Each symbol in the codebook is one of the centroids created by the k-means algorithm that defines a cluster of the frame change vectors. As a primitive unfolds, its visual outcome will then be able to be described as a sequence of symbols from the codebook.

An imitator's codebook allows it to describe low-level visual frame changes in terms of a language of symbols, and frees it from representing visual data internally. To recognize a complete primitive from a series of these visual changes, the imitator must associate the various legitimate strings of symbols that could make up a primitive. To do this, I employ Hidden Markov Models (HMMs) [Rabiner and Juang, 1986], a modelling mechanism often used to recognize time-sensitive events. The sequences of codebook symbols describing the visual outcomes of primitive actions are used to train one HMM for each primitive action type. Once each primitive has a HMM trained to recognize it, the HMMs can be used to recognize primitives from visual data obtained from demonstrations.

Demonstrations are recorded using the Ergo vision system, and are presented

individually to the imitation learner. My imitation learning framework breaks the demonstration data into segments, creating a segment whenever the demonstrator has moved (or turned) too far for any of the imitator's primitives to possibly match the motion. A segment is thus an atomic piece of a demonstration, in that it can be described by a single imitator primitive. Each HMM calculates the probability that the primitive it represents will match the current segment. The primitive that has the highest probability of a match is appended to the end of the sequence of primitives being built to approximate the demonstration. Once a demonstration has been converted into a sequence of primitives, this sequence can be used to crudely approximate the demonstration (mimicry). The primitive sequence is used to construct a more meaningful abstraction of the demonstration using behaviours. Behaviours provide mechanisms to integrate the important actions of the demonstrations, overcome differences in physiology, and measure demonstrator skill levels.

Behaviours are learned by combining primitives to produce more complex actions based on observations [Calderon and Hu, 2003; Billard and Matarić, 2000; Nicolescu and Matarić, 2003]. In my implementation a new behaviour is created from a combination of two primitives or existing behaviours when the frequency of the two occurring in sequence surpasses a threshold. For example, suppose the primitive *forward* is recognized in demonstrations, followed by the primitive *left* often enough that the frequency of their sequential occurrence surpasses the threshold. A *forward-left* behaviour is created, made from the primitive sequence *forward* followed by *left*. Similarly, a behaviour that causes the robot to drive in a square formation might be achieved by a behaviour that is made from four *forward-left* behaviours

in sequence. To keep the number of behaviours learned reasonable, behaviours will slowly decay over time, to the point where they are deleted. If they are predicted frequently enough, their decay will slow and they will become permanent. In my work, a behaviour only keeps track of the state change of the ball by the angle that the ball is moved relative to the demonstrator and the distance it travelled, and only if it is a frequently occurring effect of that behaviour (reproducibility). For example, if the ball was moved during the observation of a behaviour, but the next time that behaviour was observed the the ball did not move, this motion of the ball would not be reproducible, and therefore would not be attributed to that behaviour (a rough statistical model is used to define reproducibility in this domain). Once new behaviours are added to the imitator's repertoire, the imitator uses the new behaviours as well as its primitives when imitating.

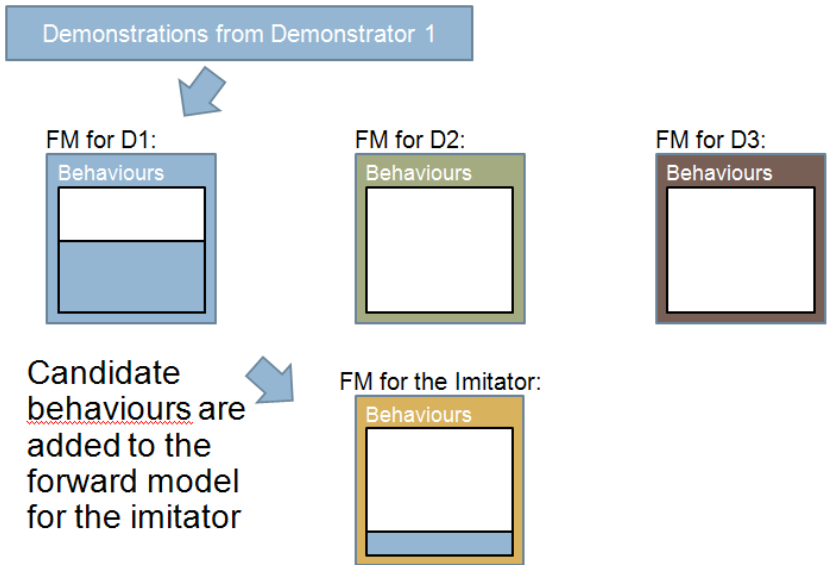

Figure 1.2: Demonstrations from demonstrator 1 are used to train the forward model representing demonstrator 1. Frequently occurring candidate behaviours are moved to the forward model representing the imitator.

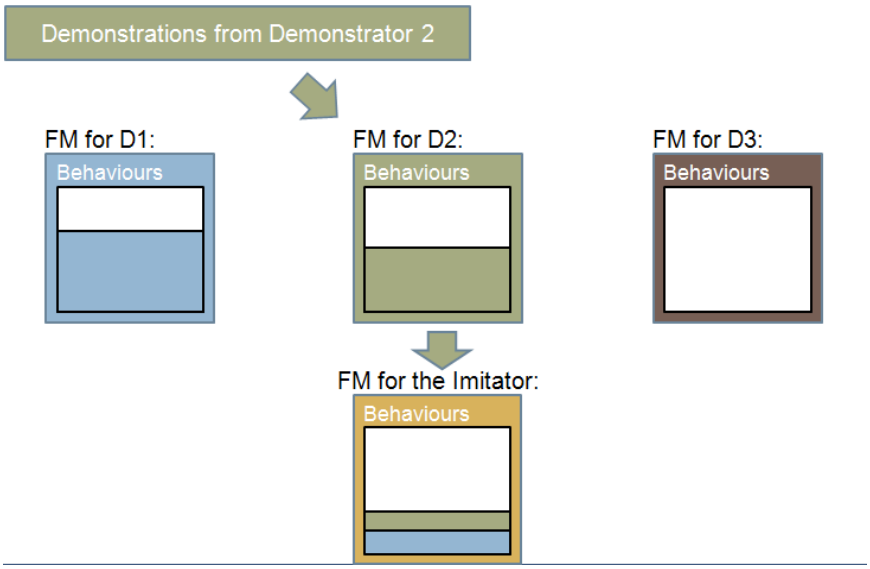

Figure 1.3: Demonstrations from demonstrator 2 are used to train the forward model representing demonstrator 2. Frequently occurring candidate behaviours are moved to the forward model representing the imitator.

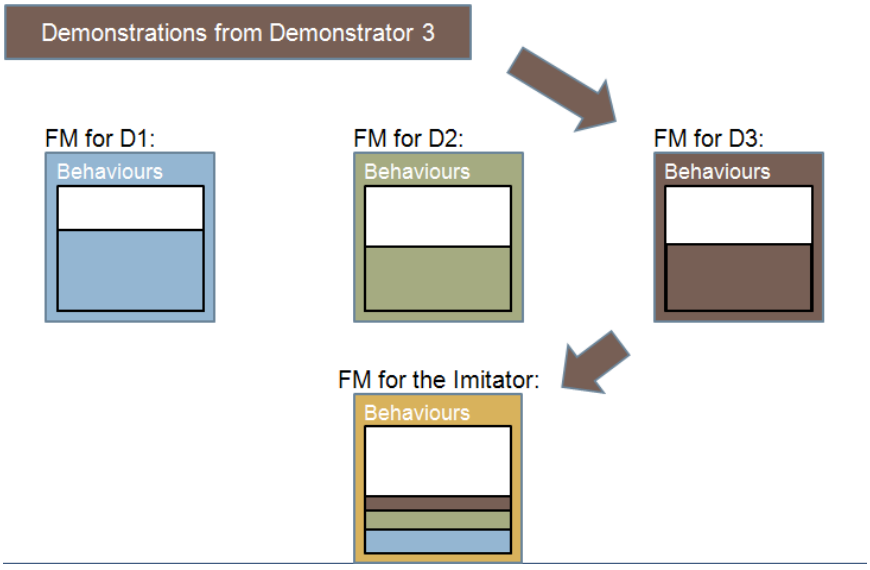

Figure 1.4: Demonstrations from demonstrator 3 are used to train the forward model representing demonstrator 3. Frequently occurring candidate behaviours are moved to the forward model representing the imitator.

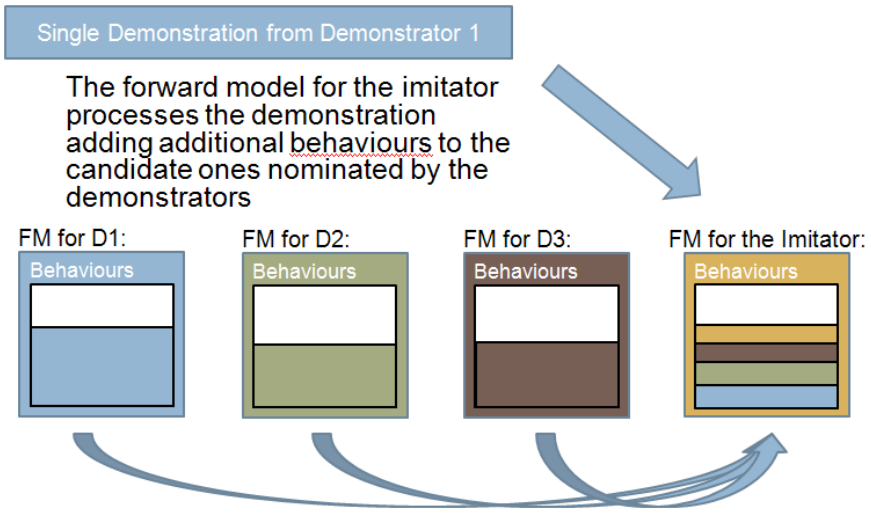

Figure 1.5: In the final phase of training, all demonstrations are first passed to the demonstrator models to elicit any candidate behaviour nominations before the forward model for the imitator processes the demonstration.

The behaviours are built and stored using a type of *forward model* which essentially represent frequencies of primitives and behaviours occurring in sequence. This idea is based on work by Dearden and Demiris [2005] who implement *forward models* that make predictions of the effects of the imitator's actions on its environment. In my approach, each demonstrator that the imitator is exposed to is assigned a unique forward model to learn that particular demonstrator's behaviours through imitation as shown in Figures 1.2:1.4. There is also an additional forward model for the imitator itself. The forward models representing demonstrators begin with only the imitator's primitives. Once behaviours for a particular demonstrator have been learned, the corresponding forward model acts as a predictive model for that specific demonstrator. That is, given the observed behaviour thus far, the model can be used to predict future behaviour of that demonstrator. Throughout the training of the demonstrator

forward models, frequently occurring candidate behaviours are added to the forward model representing the imitator as seen in Figures 1.2:1.4. Finally all models are used to process each demonstration from all demonstrators once before the forward model for the imitator does the final processing of the each demonstration using the candidate behaviours added by the forward models for the demonstrators as shown in Figure 1.5. By the time all the forward models representing all the demonstrators are trained, the model representing the imitator has a number of additional behaviours in its repertoire as a result of this process, and serves as a generalized predictive model of all useful activity obtained from all demonstrators. That is, the forward model representing the imitator will be able to predict the correct action to perform next while executing the task that has been learned from the demonstrators.

To determine the skill levels of the demonstrators relative to each other, each forward model also keeps track of a *learning preference* (LP) which is a demonstrator-specific learning rate. This learning preference is used as a weight when updating the frequencies of behaviours and primitives occurring in sequence. A high LP indicates that a demonstrator is more skilled than its peers, and so it should be learned from more quickly. A lower LP means that a demonstrator is less skilled, and so the imitator should be more cautious and learn from it more slowly. The rate that behaviours in a forward model decay is the inverse of the learning preference for that forward model, so a poorly skilled demonstrator will have the behaviours in the forward model representing that demonstrator decay at a faster rate. Changes to a forward model for a given demonstrator's learning preference happen for a variety of reasons. A forward model's LP is increased whenever the forward model has

one of its behaviours added to the model representing the imitator (from frequent use). When a behaviour is made permanent, the forward model's LP is increased, and when a behaviour is deleted, the LP is decreased. Additional ways to increase the LP are when a behaviour results in the demonstrator (ordered from highest LP increase to lowest) scoring a goal, moving the ball closer to the goal, or moving closer to the ball. The LP of a forward model for a given demonstrator can also be decreased by the opposite of these criteria (the opposite of scoring a goal would be scoring in the wrong goal, for example). I am using these criteria to shape and speed up the imitator's learning, a technique that has been shown to be effective in other domains [Matarić, 1997]. While this may seem similar to pure reinforcement learning, the difference is that these criteria are not being used to select individual behaviours: they are only used to adjust the LP of the forward models representing each demonstrator. This allows the forward models to learn behaviours more quickly from skilled demonstrators as opposed to selecting individual behaviours to learn based on the amount of reinforcement given to them.

After all demonstrations have been presented and forward models representing each demonstrator have been trained, the general forward model representing the imitator itself is then trained. This is done in the same fashion as the forward models for each demonstrator, but with all demonstrator visual data. The behaviours that have been added by training forward models for each demonstrator are also present, as opposed to beginning with primitives only. When the imitation learner is processing a single demonstration, it first passes that demonstration to the forward model it has for each of its demonstrators. Essentially this is the stage where the imitator

is using the forward models representing each of its demonstrators to predict what each individual demonstrator would do in the current situation. The same process of behaviour proposal and decay described earlier allows the imitator to keep some demonstrator behaviours, and discard others, while also learning new behaviours of its own as a result of the common behaviours extrapolated from multiple demonstrators. For example, pieces of behaviours learned from two different demonstrators may be combined at this stage into a new behaviour that is better adapted to the imitator's physiology. When the learning process is complete, the imitator is left with a final forward model that it can use as a basis for performing the tasks it has learned from the demonstrators. It is possible that additional guidance will be needed to ensure that the imitator uses the behaviours in its forward model at opportune moments. This and other research questions are addressed in the next section.

1.4 Research Questions

I will use the approach outlined above to answer the following research questions:

1. Can imitation learning be done from multiple heterogeneous demonstrators by modelling each demonstrator individually?

2. Can learning be favoured towards more skilled demonstrators by comparing their individual models?

3. Are atomic primitives adequate starting points for learning complex behaviours?

While the above questions are the focus of this thesis, I have additional questions specifically related to elements of the approach I have developed for imitation, as

outlined in Section 1.3:

1. Can counting the frequencies of primitives and behaviours occurring in sequence be used to adequately learn behaviours through imitation?

2. Is a behaviour decay system enough to keep behaviour growth in check?

3. Can a forward model designed to learn a behaviour repertoire through observations also be used to control a robot with minimal intervention?

1.5 Summary

In this chapter I have defined terms that are essential for discussing this research. I have introduced imitation learning, and the need for an imitative framework that can learn from multiple demonstrators of varying skills and physiologies. I have provided a high level overview of the methodology I have used in my approach to design and test an implementation of such a system. I have also posed the research questions that this thesis will answer.

1.6 Thesis Organization

The remainder of this document is organized as follows:

Chapter 2: Related Literature

Chapter 2 reviews the literature related to this thesis.

Chapter 3: Developing Imitation Learning for Multiple Demonstrators

Chapter 3 discusses my approach to developing an imitation learning system.

Chapter 4: Evaluation

Chapter 4 evaluates the results and provides an analysis of my experiments.

Chapter 5: Conclusions and Future Work

Chapter 5 provides answers to the research questions, and additional discussion.

Chapter 2

Background

This chapter discusses previous research relating to my thesis research. In Section 2.1 I will provide the biological inspiration for imitation learning, as well as motivation for the use of imitation learning to improve human-robot interaction. In Section 2.2 I will discuss various methods for obtaining primitive motions for imitation learners. In Section 2.3 I will detail various behaviour creation techniques in imitation learning. In Section 2.4 I will discuss the use of *forward models* in some imitation learning approaches. In Section 2.5 I will discuss other approaches to imitation learning using multiple demonstrators, including heterogeneous demonstrators. The remainder of the chapter will provide background on specific elements used in my approach. Section 2.6 will introduce Hidden Markov Models, and overview how they will be used in this work. Section 2.7 will discuss the k-means algorithm used for clustering data points in my implementation. Finally, Section 2.8 will summarize the chapter.

2.1 Biological Motivation for Imitation Learning

Humans learn from the time they are infants by observing the people around them. Making observations of the behaviour of others is important, but being able to reproduce these observed behaviours is invaluable. Infants need to experiment in order to determine the capabilities of their own bodies before they can hope to imitate the actions of others. Meltzoff and Moore proposed the existence of an underlying system of 'body babbling', where babies practice movement through self-generated activity [Meltzoff and Moore, 1997]. These experiments help infants learn from birth what their bodies can do in a general sense, but learning complex behaviours is greatly aided by external sources.

Some of the first things babies learn to imitate are facial expressions. For an infant to imitate an adult displaying a particular expression, the infant needs to somehow map the adult's expression on to the infant's own features [Meltzoff and Moore, 1997]. Infants seem to be able to do this with little difficulty, even though they have no visual feedback of their own face to compare to the visual input of the adult's expression. It is easier to evaluate attempts at imitating a hand gesture than a facial expression, since the infant can see its own hand, and compare it to how the adult's hand looks. When imitating facial expressions, the infant requires some internal representation of its own facial features, since it cannot see them directly [Meltzoff and Moore, 1997]. Meltzoff and Moore propose an *active intermodal mapping*, which enables infants to detect equivalencies between what they are doing, and what they are observing.

This active mapping between the observation and execution of a behaviour has been discovered in the form of a group of neurons called *mirror neurons* [Rizzolatti

et al., 1996]. These mirror neurons map observed actions to the structures of the brain that produce those same actions. For example, Ferrari et al. have found mirror neurons in monkeys that fire when the monkey performs certain actions with its mouth, and also when it observes other monkeys performing those same actions (e.g. lip smacking) [Ferrari et al., 2003]. This mirror neuron-based imitative process is used for skill acquisition by such animals as primates, birds, and humans [Demiris and Hayes, 2002; Matarić, 2002; Billard and Matarić, 2000]. This biological evidence has inspired researchers to develop imitation learning algorithms that are based on the concept of mirror neurons [Demiris and Johnson, 2003; Matarić, 2002; Weber et al., 2000].

Matarić used the idea of mirror neurons to develop an imitation learning system that was able to learn gestures from humans and reproduce them on a simulated 3D avatar [Matarić, 2002]. Demiris and Hayes also trained a simulated robot to learn gestures using an imitation system that closely modelled biological imitative mechanisms [Demiris and Hayes, 2002]. Similar work was done later by Calinon and Billard, though they validated their model using a real humanoid robot, which learned gestures as well as how to draw the letters 'A', 'B', and 'C' through imitation [Calinon and Billard, 2007]. In Demiris and Johnson's work, an imitation learning system is used to train a real robot with a gripper to open and close the gripper based on how far apart the demonstrating human's hands were [Demiris and Johnson, 2003]. In Demiris and Johnson's work, the robot was quite physiologically distinct from a human, but they relate the robot's grippers to human hands, making it easier for humans to interact with it. Relating demonstrator physiologies to the imitator's physiology is a

common concern in imitation learning systems [Breazeal and Scassellati, 2002b].

The problem of an imitator physically relating to its demonstrators (human or robotic) is referred to as the *correspondence problem* [Breazeal and Scassellati, 2002b]. If the demonstrator is facing the imitator, and raises its right hand, if the imitator is to learn the exact motion that the demonstrator just performed, it must also raise its right hand. This requires the imitator to map its own body onto the demonstrator's (to put itself in the demonstrator's shoes), instead of mirroring the demonstration (raising its left hand). Some work has used body suits to capture human demonstrations from the perspective of the demonstrator [Breazeal and Scassellati, 2002b], making the correspondence problem much easier to solve, as all the data is already relative to the demonstrator, so no additional mapping is needed. The trajectories of end-effectors can also be the focus, such as the demonstrator's hand when waving, or a pen while drawing [Calinon and Billard, 2007]. By focusing on these end-effectors, the correspondence problem is basically avoided, as the imitator needs to match one of its end-effectors to the observed trajectories of the demonstrator's end-effectors to achieve the same goal, though the actual physical pose of the imitator is largely irrelevant as long as the goal is obtained.

A global vision system (such as Ergo [Baltes and Anderson, 2007]) can also be used to simplify the correspondence problem, as it is in my work. With an overhead or oblique view, the demonstrator will occlude the environmental objects with its body far less often than if the imitator were viewing the demonstrator from its own local perspective (i.e. if it were using local vision). This constant view of the demonstrator and all objects in the environment makes it easier to map demonstrator positional

movements (coordinate movements) onto the imitator's own possible motions.

When demonstrators are similar in physiologies to the imitator, it is possible for the imitator to directly relate the physical states of the demonstrator to its own physical states (for example, a humanoid robot learning from a human). By relating physical states and imitating them, the robot can learn to copy exact movement patterns of the demonstrator to achieve the same environmental effects. This would be called *mimicry* by Matarić [Matarić, 2002], who distinguishes mimicry from imitation. Matarić describes imitation as the process of creating arbitrary new motor skills through the composition of already learned skills [Matarić, 2002]. Giving robots the ability to combine new motor skills they have learned from humanoids through imitation will make it easier for humans to interact with robots in day to day life.

Billard has focused a great deal of her work in using imitation learning to make human-robot interaction more streamlined. Billard puts forth the view that through imitation learning, robots could learn much like children do, through observations during regular human activities, such as play [Billard, 2000]. This would require little to no effort on the part of the humans playing, while the robot observing them could possibly gain large repertoires of movements and behaviours. Some imitation learning approaches have required the imitating robots to follow closely behind the demonstrator [Billard and Dautenhahn, 1999], though this work was focused on developing common labels for locations in the robots' environment, and as such was more location-specific than most imitation learning tasks. Imitation learning can be passive, in that the robot simply observes and updates its internal model, without the need for direct interaction with the demonstrator. The imitating robot can

refine its model at a later time (actually trying out the new skills it has learned) [Demiris and Hayes, 2002] without the need for direct supervision. The ability of imitation learning algorithms to work without constant (or any) communication with the demonstrator is appealing not only to save on communication overhead, but also because demonstrators do not have to actively teach the imitator, they may not even be aware they are being imitated. This allows an imitating robot to potentially learn a great deal from its observations, though imitating people that are unaware they are being learned from could make some people uncomfortable [Breazeal and Scassellati, 2002b].

It can still be a difficult problem to imitate the demonstrator's actions when they share physiologies with the imitator if only the visible effects of the action are known. For example, in a local vision system (where the robot views its environment from a camera mounted on its body) the actual action taken by a demonstrator may not be clear, just the results of the action. Occlusion of the demonstrator's body parts (often by the demonstrator's body itself, if the demonstrator has its back to the imitator, for example) can also hide the actual actions taken by the demonstrator [Billard and Dautenhahn, 1999]. This problem can actually become a benefit if imitation is done by not focusing on the exact movements of the demonstrator, but instead concentrating on the environmental changes caused by the demonstrator [Crabbe and Dyer, 2000; Calderon and Hu, 2003; Jenkins and Matarić, 2003]. Since the imitator is imitating the environmental effects of the demonstrator, it can achieve those effects through whatever physical mechanisms are convenient to it without worrying about exactly matching demonstrator actions. For example, a humanoid robot may kick

a ball to move it in a specific direction, but the imitator may be a wheeled robot without a kicking ability. To compensate for its lack of a kicking ability, the imitator could roll the ball in the desired direction by driving into it. With no dependencies on the physical nature of imitator and demonstrator, the imitator can theoretically learn from any physical type of demonstrator [Calderon and Hu, 2003; Billard and Matarić, 2000; Billard and Dautenhahn, 1999].

The imitator uses primitives to approximate the observed changes in the environment that were caused by the demonstrator's actions [Billard and Dautenhahn, 1999; Matarić, 2002; Nicolescu and Matarić, 2003]. Primitives (detailed in the next section) are the low level actions available to the imitator, while the hierarchical combinations of primitives are called behaviours (see Section 2.3). Behaviours are necessary for providing meaningful abstractions that would be impossible with primitives alone (using only primitives a robot is limited to mimicry, while behaviours provide the mechanisms for imitation). Behaviours can represent complex actions as well as relate those actions to changes in the environment. In the next section I will discuss various methods for choosing primitives when designing an imitation learning system.

2.2 Primitives

To be able to make behaviours through high level abstractions an imitator must first start with a set of primitives [Matarić, 2002; Weber et al., 2000]. A primitive should essentially be a link between the perception of a simple action occurring, and the means of the imitator to perform that same action [Calderon and Hu, 2003]. Primitives can be obtained by manual selection or automatic derivation of the primitives

from observation [Jenkins and Matarić, 2003].

Matarić employs visual sequence segmentation techniques, such as spatio-temporal structure extraction [Matarić, 2000] to detect primitives from visual data. In her work, human movement data is gathered using vision-based motion tracking, magnetic markers on arms, and a sensor suit for gathering full body joint angle data. The dimensionality of the data is reduced using principle component analysis, wavelet compression, and correlating across multiple degrees of freedom. She then uses clustering techniques to find patterns in the data that are translated into primitive motions [Matarić, 2000].

Walter et al. use Gaussian mixtures in their system to automatically segment and label visual streams of hand gestures [Walter et al., 2001]. They use Gaussian mixtures to model demonstrated gestures as repetitive sequences of atomic components (primitives). The k-means algorithm [Hartigan and Wong, 1979] and minimum description length criterion were used to determine the number of components for the Gaussian mixtures.

My work is different than the automatically generated primitive work, as my primitives are manually selected. Automatic generation techniques use some specific processing methods that are loosely related to some of the classification work I have done with primitives.

Similar techniques used to automatically generate primitives can also be used to classify them. Jenkins et al. [2000] chose primitive motions for their simulated humanoid imitator, and used techniques often used for primitive generation, such as a vector quantization algorithm to recognize the primitives. I use a similar method as

part of my recognition process, specifically the k-means algorithm, which is detailed in Section 2.7. My work is different because I process my data to represent the environment relative to the robot (see Section 3.1.2), and use it to train Hidden Markov Models (see Section 2.6) to perform the actual recognition.

When choosing primitives, care must be taken to ensure that the desired range of the imitator's motion can be represented. Primitives can be selected at a more behavioural level with some bias towards the learning environment. For example, Matarić [2000] used predefined primitives such as avoidance, following, homing, aggregation and dispersion. While this may result in the imitator missing out on lower level learning opportunities through over generalization, it can speed learning towards higher level behaviours. Often lower level primitives are used to open up the avenues of learning that an imitator can explore. Moving a step down from the behavioural level and focusing only on movement patterns, Weber et al. [2000] predefined three movement primitives: straight line, elliptical, and oscillatory.

In my work, I use hand-picked primitives, but my work is different in that I chose to keep the imitator's primitive repertoire as open to learning opportunities as possible for the robot by using the most atomic motor commands available to the imitating robot. My system uses the low level actions *forward, backward, left, right*, and *stop* as the primitive commands. I use a wheeled mobile robot for the imitator, which has considerably fewer motor commands than robots such as humanoids. I chose the wheeled robot for the imitator as it reduces the number of basic primitives the imitator starts with, and thus makes the scope of this research reasonable. By keeping the primitive motions atomic, the imitator's primitives can more easily match

observed demonstrations (though it may require more of them to do so).

However the primitives of an imitator are defined, imitation learning requires a mechanism of composing these primitives into more abstract behaviours [Matarić, 2002; Weber et al., 2000]. In the next section (Section 2.3) I will discuss various approaches in imitation learning used to build behaviours by observing demonstrations.

2.3 Behaviours

In imitation learning, behaviours are learned by combining primitives to produce more complex actions based on observations [Calderon and Hu, 2003; Billard and Matarić, 2000; Nicolescu and Matarić, 2003]. Behaviours often take the environmental effects of their execution into consideration, unlike most approaches to primitives. There are a number of different approaches for composing primitives into new behaviours.

Calderon and Hu [2003] define the primitive actions in their system with preconditions and postconditions. Higher level behaviours are created by building networks of these primitives, with the requirement that postconditions from one primitive match the preconditions of another. For example, the primitive *approach-box* must execute before the primitive *pick-up-box* can execute. An *approach-and-pick-up-box* behaviour can thus be created that encompasses both primitives. My work differs in that I only use preconditions when using the learned forward models to control a robot, as opposed to during the learning phase. My primitives are also much more atomic, with no goal intentions associated with them.

Nicolescu and Matarić [2003] also use behaviour preconditions to link simple behaviours into more elaborate ones. They focus on topologies of commonly occurring behaviours, and attempt to generalize these to create more meaningful abstractions. They also experiment with the use of teacher feedback to guide the creation of new behaviours. My work does not focus on topologies of behaviours, and has no communication between the demonstrators and the imitator.

Demiris and Hayes [2002] use predictive models (*forward models* which are described in more detail in Section 2.4) to create behaviours. Forward models are used to make predictions of the environmental effects of a behaviour or primitive. When no known behaviour has a sufficiently high confidence value (determined by its accuracy in predicting the current demonstration), the demonstrated behaviour is considered novel. The novel behaviour's representative postures are stored, and combined with a controller, are used to form a new behaviour that can approximate the postures [Demiris and Hayes, 2002]. My implementation does not focus on the control of behaviours, and learns behaviours by predicting and matching sequences of behaviours and primitives, as opposed to capturing posture / state information to create new behaviours.

Demiris and Johnson [2003] use *inverse models* as well as forward models to create new behaviours as an extension of the work done by Demiris and Hayes [2002]. Inverse models take a goal and current state of a controlled system as inputs, and output the necessary motor commands needed in order to achieve or maintain that goal [Demiris and Johnson, 2003]. The inverse models are their behaviours (basic inverse models are used to start off the system with primitive behaviours), where each inverse

model can output the necessary motor commands needed to achieve a desired goal [Demiris and Johnson, 2003]. While an inverse model is executing, a related forward model is used to predict what the expected outcome will be. The system can then refine the performance of an inverse model until it reaches the desired output. They then construct higher level behaviours by composing more complex inverse models created by arranging lower-level inverse models serially, or connecting them in graphs [Demiris and Johnson, 2003]. My work does not focus on the refinement of controlling individual behaviours. My forward models are representations of the entire behaviour repertoire of a specific demonstrator, in contrast to using a unique forward and inverse model to model each individual behaviour.

Recurrent neural networks have been used in imitation learning [Billard and Matarić, 2000] to create behaviours that are complex sequences of motor primitives. This connectionist approach models the spinal cord by including primitives that act as central pattern generators. Learning new behaviours is done by building connections based on spatio-temporal associations of inputs. My work differs by building behaviours through the composition of behaviours or primitives that frequently occur in sequence into more complex behaviours.

Through studies of rats, apes, and chimpanzees Byrne has developed a theory of imitation that uses string parsing [Byrne, 1996], where strings of primitive behaviours are used to compose more complex behaviours. I use a similar technique to create behaviours from sequences of primitives. This method of behaviour composition fits well with the atomic primitives in my implementation, and also helps keep the scope of this thesis reasonable. In my system, primitives or behaviours that frequently

occur in sequence trigger the creation of a behaviour that encompasses both of the sequentially occurring primitives. My architecture is also different from Byrne [1996]'s work because the primitives are not simply bound to each other by how they occur in a demonstration, rather they are bound together by which ones are sequentially predicted to match the demonstration. Byrne [1996]'s work deals with theoretical mechanisms underlying animal behaviour, whereas my work is an actual implementation of a behaviour creation technique that composes behaviours from sequentially occurring behaviours and primitives.

Simply composing behaviours from primitives is one thing, but to learn effective behaviours an imitator needs to know how its actions will affect its environment. The next section (Section 2.4) discusses how forward models can be used to predict the environmental effects of the imitator's actions.

2.4 Forward Models

Using forward models for imitation learning is one of many approaches, but as previous work with forward models has heavily influenced my own implementation, I will provide additional background for them here.

Demiris and Hayes [2002] developed a computational model based on the phenomenon of *body babbling*, where babies practice movement through self-generated activity [Meltzoff and Moore, 1997]. Demiris and Hayes devised their system using *forward models* to predict the outcomes of the imitator's behaviours, in order to find the best match to an observed demonstrator's behaviour. A forward model takes as input the state of the environment and a control command that is to be applied.

Using this information, the forward model predicts the next state and outputs it. In their implementation, the effects of all the behaviours are predicted and compared to the actual demonstrator's state at the next time step. Each behaviour has an error signal that is then used to update its confidence that it can match that particular demonstrator behaviour. As mentioned previously, Demiris and Johnson [2003] extended this work, but their use of forward models was essentially the same. My work differs because I use forward models to model entire behaviour repertoires of demonstrators, not individual behaviours. Their work also focuses more on refining individual behaviour controls, while my work does no behaviour refinement.

Demiris and Hayes [2002] use a combination of two types of imitation. *Passive imitation* in their system is a process of acquiring new behaviours. In passive imitation, the motor systems are only active during the phase where the imitator is reproducing the demonstrated behaviour. According to Demiris and Hayes *active imitation* is when the forward models are used to predict the demonstrated behaviour by selecting a currently known behaviour. In their combined *dual-route* model (active as well as passive), the imitator uses active imitation when observing demonstrations. When no known behaviour has a sufficiently high confidence value in predicting the current demonstration, the demonstrated behaviour is considered novel. When this happens, the passive imitation architecture takes over to learn the novel behaviour. Once the new behaviour is learned, it is added to the forward model, which will now correctly predict that behaviour if it arises in the future. My implementation, in contrast, learns behaviours by composing frequently occurring sequential behaviours and primitives into new behaviours. My architecture does not have a passive and active

mode: the prediction of behaviours to select and learning of new behaviours happens during the entire learning phase of my system. The prediction of new behaviours is the driving mechanism to behaviour creation, unlike in their work.

Demiris and Hayes [2002] use one forward model for each behaviour, which is then refined based on how accurately the forward model predicts the behaviour's outcome. By using many of these forward models, Demiris and Hayes construct a repertoire of behaviours with predictive capabilities. In contrast, the forward models in my framework model the repertoire of individual demonstrators (instead of having an individual forward model for each behaviour), and contain individual behaviours learned from specific demonstrators within them (the behaviours can still predict their effects on the environment, but these effects are not refined during execution). The implementation details of their forward models are also quite different from my work, as theirs are more focused on refining the exact movements of each individual behaviour. This provides the imitator with a model that can make predictions about what behaviours a specific demonstrator might use at a given time.

Dearden and Demiris [2005] also implement forward models, though they use Bayesian networks as the basis for them. Like Demiris and Hayes [2002], they use *motor babbling* (similar to the infant *body babbling*) to explore and control the motor systems. This involves randomly activating motor controls and observing and recording their effect on the environment. Dearden and Demiris [2005] use the Bayesian networks to learn the relationships between motor commands and the visual effects of executing those commands. In their work, the motor babbling did not have an excessively large state space to explore, as it was only done for a one degree of freedom

gripper on their imitating robot. The imitator learned to match the amount the gripper was open to the demonstrations of humans who held their hands apart at varying distances. The motor babbling approach is interesting, but it might be necessary to perform the bulk of it in simulation for robots with larger movement repertoires (e.g. humanoids) as the number of possible states for the motors to explore becomes intractable as more degrees of freedom are added. In my implementation, I gathered primitive data in a systematic manner in order for my imitation learner to have models of how each primitive affects the imitator's position and orientation. I chose to gather and process the primitive data manually as opposed to a motor babbling technique because of the relatively large state space of five primitives in various positions on a field compared to the smaller state space of their gripper robot work.

Unlike any other work with forward models in imitation learning, my imitation learner has a separate forward model for each unique demonstrator that it has observed. These separate forward models will eventually be used to learn a single forward model unique to the imitator containing the most useful demonstrator behaviours. The forward models in my work are used to predict the outcomes of behaviours as well as create new behaviours by combining primitives and existing behaviours. Forward models in my implementation contain all the behaviours that have been learned from a specific demonstrator. The forward model predicts the effects of all the behaviours it contains, and these are compared to the actual demonstrator actions at the next time step. The behaviour/primitive that has the best match is selected. My forward model implementation keeps track of the frequencies of behaviours or primitives occurring sequentially. When the frequency surpasses a threshold, a new

behaviour is created that is made of the two behaviours or primitives occurring in sequence. The behaviour or primitive that has the best match also has its *permanency* increased, a value that represents the usefulness of the behaviour. This is similar to the confidence value used by Demiris and Hayes [2002], though my permanency attribute is more of a general measure of a behaviour's usefulness, as opposed to its confidence of approximating a specific demonstrator behaviour. If the permanency surpasses an upper threshold, the behaviour will never be deleted. If a behaviour's permanency drops below a threshold, the behaviour is removed.

My forward model implementation has additional features that are designed to model multiple heterogeneous demonstrators. I will discuss these features as well as give a general overview of other work dealing with multiple demonstrators in the next section.

2.5 Imitating Multiple Demonstrators

Current imitation learning systems tend to lump all teachers together, making it impossible to determine the quality of what is learned from each teacher [Price and Boutilier, 2003; Calinon and Billard, 2007]. Having a rough idea of the competence of your teachers is very useful, especially if you are being taught one task by many teachers. People trust the teachers that have proven to be skilled at the task being learned, and mistrust teachers that seem to be less skilled. Individually modelling its teachers gives an imitation learning robot the ability to compare the quality of the teachers relative to each other. This enables the robot to be more resistant to bad demonstrations and focus on learning the desired task.

Prior work in imitation learning has often used a series of demonstrations from demonstrators that are similar in skill level and physiologies [Calinon and Billard, 2007; Nicolescu and Matarić, 2003]. The approach presented in this thesis is designed from the bottom up to learn from multiple demonstrators that vary physically, as well as in underlying control programs and skill levels.

Some recent work in humanoid robots imitating humans has used many demonstrations, but not necessarily different demonstrators, and very few have modelled each demonstrator separately. Those that do employ different demonstrators, such as [Calinon and Billard, 2007], often have demonstrators of similar skills and physiologies (in this work all humans performing simple drawing tasks) that also manipulate their environment using the same parts of their physiology as the imitator (in this case the imitator was a humanoid robot learning how to draw letters, the demonstrators and imitators used the same hands to draw). In general, few prior approaches to imitation learning with heterogeneous demonstrators have focused on non-humanoids learning from humanoid robots.

Nicolescu and Matarić [2003] motivate the desire to have robots with the ability to generalize over multiple teaching experiences. They explain that the quality of a teacher's demonstration and particularities of the environment can prevent the imitator from learning from a single trial. They also note that multiple trials help to identify important parts of a task, but point out that repeated observations of irrelevant steps can cause the imitator to learn undesirable behaviours. They do not implement any method of modelling individual demonstrators, or try to evaluate demonstrator skill levels as my work does. By ranking demonstrators relative to each

other and choosing the best approaches across all demonstrator approaches, I believe that my system can minimize the behaviours it learns that contain irrelevant steps.

The ability of imitation learning systems to individually model demonstrators and evaluate their relative skill levels is rare. The little previous research into the area of modelling multiple heterogeneous demonstrators includes Yamaguchi et al. [1996]'s work as well as Yamaguchi et al. [1997]'s work (which is an extension of [Yamaguchi et al., 1996]). They develop an imitation learning system that allows agents on a team to imitate teammates that are more skilled. Their work also models the individual skills of others, though not by building repertoires of each demonstrator's individual behaviours as I do in my implementation. In their work, if there are no teammates of higher skill, the agents learn through reinforcement learning. They make the assumption that the agents are homogenous, and recognize the ability of others by the amount of positive reinforcement they have received. My research differs in that I have designed my system to deal with heterogeneity in both physiologies and skill levels. I also evaluate the relative skill of the demonstrators using a combination of the reinforcement they receive, but in addition to this, the imitator evaluates demonstrators based on how many behaviours have been learned from them that are frequently used.

Many imitation learning systems assume that demonstrators are reasonably skilled at the task being demonstrated, even systems that are designed to learn from multiple demonstrators. For example, Price and Boutilier [2003] implement imitation learning from multiple demonstrators, though they extract information about the demonstrators simultaneously and combine it without comparing their abilities. While this is

a faster method of learning from multiple sources, it would not handle poor demonstrators very well. Since the information about the demonstrators is simply combined (with each demonstrator contributing equally), a very bad demonstrator would greatly decrease the imitator's ability. My research compares the demonstrators and their behaviours to determine their overall quality. The imitator can use these comparisons to weigh its learning towards more skilled demonstrators.

The problems involved in imitation learning from multiple heterogeneous demonstrators are similar to those encountered when deciding who to interact with in a group of agents when a choice is available. This problem is common coalition formation, and was explored in a soccer domain in [Anderson et al., 2004b]. There, reinforcement learning was used to create a model of teammates, and is used to both select a passing target when options present themselves, and to allow an agent to cover for others on a team that are known to be weak. That work involved only keeping track of an agent's quality of performance, however, rather than attempting to actually learn domain behaviour. It also did not consider physical heterogeneity at all.

Aside from learning from trusted demonstrators (possibly teammates), learning behaviours of opponents is also useful, as it allows the robot to predict their actions and attempt to counteract them [Riley and Veloso, 2001]. My research is not geared specifically to this goal, though in my system the imitator can learn from demonstrators without communicating with them. This makes learning from opponents feasible, and since my imitator learns models of the behaviours of its demonstrators, it could predict the behaviours of its opponents using the models it has learned by observing

them.

An area of research closely related to imitation learning is *plan recognition*. In classic plan recognition, a sequence of actions performed by an actor is used to infer the goals being pursued by that actor, and the action sequence is organized into a plan structure [Kautz, 1991]. In contrast, imitation learning does not construct a plan structure. Instead, an imitation learner acquires new behaviours through the observation of the demonstrator's actions. The imitator can use its newly learned actions to achieve the same outcomes as the demonstrator; however, it can improvise instead of just following a plan extracted from observing the demonstrator.

In addition to the background on imitation learning presented thus far, in order to describe my own methodology (Chapter 3) the reader must also be familiar with a number of tangential areas of research. In particular, I make use of Hidden Markov Models [Rabiner and Juang, 1986], and the k-means clustering algorithm [Hartigan and Wong, 1979] in order to implement portions of the imitative process. Background on these particular topics is presented in the following two sections.

2.6 Hidden Markov Models

Hidden Markov Models (HMMs) are just one approach used to map complex data (such as visual data in my implementation) to low level primitives. Chernova and Veloso [2007] use *Gaussian mixture models* to generate an action policy based on training data. They separate all data points by the actions that created them to cluster all the observations that led to the same action. A separate Gaussian mixture is used to represent each class of action in their work as well. I did not use Gaussian

mixtures, as they seemed much more complex than necessary for my approach given the preliminary results and symmetric clustering I achieved in my early experiments (see Section 3.1.2) with the relatively simpler k-means clustering technique described in the next section. HMMs can also be trained with Gaussian mixtures, but they take longer to train and run than HMMs trained with discrete values, such as those in my work [Rigoll et al., 1996].

Saunders et al. [2006] use a nearest-neighbour approach to match current states of their robot with previously visited training states. In their work, the training data is used to create state vectors that are hand labelled with the desired primitives to be used at those states. The robot then uses its current state to find the nearest-neighbour (closest vector) in the training state vectors, to determine its current state. The primitive associated with the training state vector that is the closest to the robot's current state vector is selected. I did not explore a technique such as this, as my work focuses on classifying primitives from the low level x, y, and orientation values of the robots contained in the visual frames, unlike this work where the state vectors are combinations of sensory inputs. I did not want to manually associate primitives with any given state either, as my framework is designed to attempt to learn any associations by constructing abstract behaviours out of the primitives.

I chose to use HMMs in my research because I believed that their ability to recognize time-sensitive patterns in data [Rabiner and Juang, 1986] would work well with the visual frames captured by the vision system I use, and I also believed that the HMMs would be able to properly model the random motion jitter that is typical in global vision systems (from issues such as frame interlacing). In my implementation,

HMMs are used to convert a demonstration into a sequence of imitator primitives, so that the demonstration is represented in terms the imitator can relate to.

Hidden Markov Models have been used extensively in other research. HMMs are often used in systems that recognize sequences of events with underlying stochastic elements, such as speech recognition [Rabiner and Juang, 1986]. HMMs have also been used successfully in research areas such as hand-drawn character recognition [Yang et al., 1997], control systems for a sun-seeking device and inverted pendulum balancing [Yang and Xu, 1994], and assembly tasks learned from human demonstration [Hovland et al., 1996]. I train HMMs in my work by individually recording the imitation learner executing each of its primitives. These individual observation sequences are similar to the partitioning of a sequence of control signals into individual patterns, as done by Yang and Xu [1994], where each smaller pattern is used to train an HMM. Calinon and Billard [2007] use Hidden Markov Models (HMMs) as part of their imitation learning system, which aids in the labelling of their hand gesture and letter drawing visual data streams.

HMMs model stochastic processes where one of the stochastic processes is hidden, and can only be observed through another set of stochastic processes that produce a sequence of observations [Rabiner and Juang, 1986]. Rabiner and Juang use the analogy of a coin-tossing experiment, where the observer is only told whether the coin toss results in *heads* or *tails*. The result of the tosses produce a sequence of observations. The hidden stochastic element is how the coin toss results are being generated. Rabiner and Juang use the example of the person tossing the coin doing so behind a curtain, so they are hidden from the observer, and using multiple coins:

some of them regular *fair* coins, and others fixed or weighted so that the outcome is biased toward one of *heads* or *tails*. Beyond the stochastic element of the actual coin toss, there is also the stochastic element of which of the multiple coins is used in any particular toss. The transition probabilities are used to illuminate patterns (if any exist) in how the coins are selected. In an HMM each coin would be a state, and each transition between states is done randomly, according to the probability of each specific transition. The observations all have probabilities associated with them as well. Each state has its own set of observation probabilities associated with it. Following the coin toss example, if one of the coins (states) is a fair coin, then its observation probabilities would be 0.5 for *heads* and 0.5 for *tails*.

HMMs consist of states (coins in the previous example), distinct observation symbols (*heads* or *tails* in the above example), and an initial state probability distribution (the probability of any coin being selected first in the coins example), all of which are tied together using tables of probability distributions [Rabiner and Juang, 1986]. HMMs use two tables: a *transition* probability table, and an *observation* probability table. The transition probability table is often seeded with random values (and is in my work), and the transitions are refined through the training process [Rabiner, 1989]. Each element in the transition probability table represents the probability of transitioning from one state to another. All rows in the transition table represent a transition probability distribution, so they should sum to 1. The observation probability table is usually seeded with the training data (as it is in my work) and then refined through the training process [Rabiner, 1989]. All elements in the observation probability table represent the probability of observing a specific output symbol from

a specific state (in keeping with the coin toss example, the probability of observing a *heads* from a specific coin). All rows in the observation table represent an observation probability distribution, so they should sum to 1. The initial state probability distribution is simply used to determine which state the model should start in. The HMM is optimized during training to match the training data as closely as possible.

There are two main types of Hidden Markov Models: *discrete* and *continuous* [Rabiner and Juang, 1986]. These terms refer to the way that the observations are dealt with [Rabiner, 1989]. Discrete HMMs label the output as specific symbols, whereas continuous HMMs use continuous multi-dimensional vectors [Inamura et al., 2004]. I chose to use discrete HMMs for my work, as they are generally faster to train and use than continuous HMMs [Rigoll et al., 1996; Inamura et al., 2002]. This is because continuous models use Gaussian mixtures, as opposed to the relatively simpler vector quantization technique often used with discrete HMMs [Rabiner, 1989]. Rigoll et al. [1996] also found that for certain tasks such as handwriting recognition, discrete models were superior (continuous models are often used for more complex signal processing in tasks such as speech recognition [Rabiner, 1989; Inamura et al., 2004]). My vision data is much more similar to handwriting (especially when only considering the x and y coordinates of the robot) than it is to speech or other complex signals.

To train the discrete HMMs in my work, the vision data must first be converted into discrete observation symbols. In the next section I will briefly describe the k-means clustering algorithm that I use to create these discrete observation symbols from vision data.

2.7 The K-Means Clustering Algorithm

The visual data obtained from the imitator and its demonstrators in my work is clustered using the k-means algorithm [Hartigan and Wong, 1979]. Preliminary experiments with k-means resulted in very distinct clusters and classification results using my training data (see Section 3.1.2), which led me to adopt it as the clustering algorithm for the entirety of this thesis work. This clustering is used to create a codebook made up of the centroids created by the k-means algorithm, which in turn is used to classify visual sequence frames as discrete symbols (the centroids). My implementation uses the centroid splitting technique, also referred to as *binary vector quantization* or the *LBG* (Linde, Buzo, Gray) algorithm [Linde et al., 1980]. The algorithm starts by creating a single centroid that is a point at the center of all data points. This centroid is split into two points very close to the original centroid, and each data point is classified by the closest of these two new centroids. With this centroid splitting method, the number of centroids generated is a power of two. After all the data points are divided into groups, the groups need to have new centroids calculated so that the new centroids are the points most central to each group. The new centroids are calculated using the k-means algorithm [Hartigan and Wong, 1979]. This is done by choosing the most central points in each group that minimize the distortion, where distortion is defined as the sum of squared distances of each data point from its group's center point [Moore, 2004]. After each group has its centroid calculated, the algorithm can continue until a desired number of centroids is reached.

Data points in my implementation are vectors that have dimensions relating to the x, y, and θ of a robot (the imitator, or one of the demonstrators). Each visual

frame is converted to one of these data points. To classify a visual frame, the closest centroid is chosen as the discrete observation symbol used to represent that visual frame. The codebook of centroids is needed to create sequences of discrete symbols to represent the primitives' sequences of visual frames. These sequences of discrete symbols are used to train Hidden Markov Models (Section 2.6) to recognize the imitator's primitives. In my implementation I found 512 centroids to be an optimal number of discrete observation symbols for training the HMMs.

2.8 Summary

In this chapter I have provided the background necessary to understand my implementation. I have contrasted existing research in imitation learning with my own approach, and provided additional motivation for the need of imitation learning systems that can learn from multiple demonstrators with heterogeneous physiologies and skill levels. In the next chapter (Chapter 3), I will provide a detailed description of my implementation.

Chapter 3

Developing Imitation Learning for Multiple Demonstrators

This chapter describes the development of my imitation learning system for modelling multiple heterogeneous demonstrators. The development and training of the Hidden Markov Models is covered in Section 3.1. These Hidden Markov Models are used to recognize primitives from unknown visual sequences (demonstrations), and this process is detailed in Section 3.2. Section 3.3 elaborates on the creation of behaviours from sequences of primitives using forward models. The development process of the forward models is discussed in Section 3.4. Finally, in Section 3.5 I will provide a summary of this chapter.

3.1 Understanding the Visual Effects of Actions

At its most basic level, imitation learning must couple the observation of an action or behaviour to the imitator's ability to execute an equivalent action or behaviour. The imitator must be able to convert a demonstration into a sequence of the imitator's own actions, reproducing the demonstration through mimicry, not imitation (imitation requires composition of higher level behaviours, as detailed in Section 3.3). An imitator's ability to recognize the effects of its own actions is therefore core to its ability to reproduce the actions of others.

The following subsections will discuss the approach I use to give an agent the ability to recognize its own actions visually using Hidden Markov Models. These Hidden Markov Models can then be used to recognize primitive sequences in the visual data gathered from demonstrations by others. This involves several steps: collecting visual data, representing it for the purposes of training an HMM, and finally implementing and training the HMM. In practice, my work on each of these steps was highly interrelated: a poor result in one area led to changes in another. For the purposes of a logical explanation of my approach however, I explain these in this sequence in the subsections that follow.

3.1.1 Gathering Primitive Visual Data

The imitator used in my work is a two-wheeled robot built from a Lego Mindstorms kit previously shown in (Figure 1.1). The imitator's low level actions are five primitive motions available to a wheeled robot: *forward, backward, left, right, and stop.* While a model associating these actions with the visual effects of executing them could be

hand-developed and supplied to the agent, part of my work involves creating such a model through exploration. The visual effects of carrying out these primitives are recorded using *Ergo*, a global vision system. Ergo gathers data from an oblique view [Baltes and Anderson, 2007], and is used throughout my work to obtain x and y coordinates of the ball and robots (imitator and demonstrators) as well as the orientation of the robots. The imitator's primitive visual data is obtained with the imitator on the field by itself.

I gather vision data from all of the primitives as the imitator executes them in a random order, instead of gathering data from each primitive separately. This is done to ensure that the model is robust enough to deal with residual motion between primitives. For example, a *forward* primitive may have almost finished moving the imitator, but just before the imitator has stopped completely, a *right* primitive may start. The fact that forward momentum did not allow a visually clean break between primitives needs to be accounted for, or the recognition of primitives would require artificially discontinuous motion to be successful (e.g. visualize someone walking normally vs. having to pause and separate each motion distinctly). Because these subtle changes will differ depending on the combination of motions, successful recognition depends on training the system to recognize all the various combinations. In the end, the outcome of each primitive execution ends up occupying 25-30 visual frames (one second) of video, with each frame recording the x and y position and orientation of the robot.

The visual sequences are recorded from many different positions on the field, because the robot wanders while executing random primitives. When the robot gets

Primitive	**Average Distance**	**Average Rotation**
Forward	42.25mm (12.98mm)	-1.15 degrees (0.57 degrees)
Backward	-48.58mm (12.33mm)	2.29 degrees (12.6 degrees)
Left	3.95mm (5.16mm)	30.37 degrees (18.33 degrees)
Right	3.28mm (4.46mm)	-29.22 degrees (8.59 degrees)

Table 3.1: The average distances and rotations collected from all training data for the various primitives (excluding the *stop*) primitive. Corresponding standard deviations are given in parentheses.

close to the edge of the field, it is placed in the center and allowed to continue.

I gather 1000 visual sequences for each primitive type, for a total of 5000 sequences, to be used as training data for the HMMs. Table 3.1 shows the average motion of all primitives (with the exception of the *stop* primitive) which were calculated from the primitive training data sets. These average motions can be used to predict how the imitator would move if a certain primitive command were to be executed. The use of these predictions in imitation learning is detailed in sections 3.3 and 3.4.

I also kept more than 300 visual sequences of each primitive type, which were not used for training, to form a test data set for validating the classification accuracy of the HMMs. In order to train a hidden Markov model with this captured data, it must be represented in a form where field positions are specified relative to the robot itself, rather than absolute, so that the robot will not associate specific areas of the field while learning the visual effects of its primitives. The next section deals with the alternatives I explored while developing such a representation that would be suitable for training HMMs. Following this, I describe my implementation of these Hidden Markov Models, and the HMM training process.

3.1.2 Alternative Visual Data Representations

While training the HMMs, I experimented with different visual data processing approaches in an attempt to develop a suitable representation that gave good results. In this section I will discuss the alternatives I explored, and then describe in detail the two final visual data processing techniques chosen to train the HMMs.

To process the visual data captured as described in Section 3.1.1, the sequences depicting the random imitator primitives are first sorted into data sets consisting of 1000 examples of each individual primitive. Each data set thus covers the full range of differences in primitive visual outcome as a result of appearing in sequence with various other primitives. The mean and standard deviation of each primitive's data set is also calculated, to be used later when predicting the environmental effects of the primitives (Section 3.4). I tried a number of different visual data conversions, and evaluated them by first clustering the converted data into discrete symbols needed to train the HMMs in my implementation (as described in Section 3.1.4). I then use this clustered data to train HMMs and test to see how many unknown primitives can be recognized from the test data set.

I first attempted to process the primitive sequences by representing them as the changes in each of the values x, y, and θ between each frame. The differences between frames proved to be a dead end, as the changes were too small to differentiate between one type of sequence and another. The average HMM classification accuracy for each primitive type was around 60%. I had hoped that the time-sensitive nature of HMMs would account for the changes over time, but this was not the case. Also, the jitter inherent in the global vision data from Ergo [Baltes and Anderson, 2007] was an

issue, since it introduced small changes in the vision data (especially in θ) that made determining the cause of differences between each frame unreliable. I also tried using the differences between frames for only x and y, and then using the actual value of θ, but this did not improve the results. It was the failure of this approach that led to the final vision data processing techniques, so it was still a valuable learning process.

The vision processing technique that I settled on I refer to as an *incremental frame change* approach, where the origin frame is represented as having no changes from the origin (i.e. no differences from itself), and all other frames are described relative to this origin frame. This concept is very similar to polar coordinates. For example: if you go for a walk, the place your walk started is the origin. Each position after can be related relative to the origin (if the orientation is temporarily ignored), by the number of steps needed to get from the origin to the current position. The x coordinate in an incremental frame is the difference between a given frame's x coordinate and the origin frame's x coordinate. The y coordinates are treated analogously.

Figure 3.1 illustrates an example of this approach in the context of an imitator's *forward* primitive. The robot's movements during the execution of the *forward* primitive are shown as four separate frames with their corresponding incremental frame change vector below. The first frame on the left shows the origin frame, with a vector of size zero. Each subsequent frame shows the position of the robot in the origin frame in gray, and the current position of the robot in that frame in black. As the robot moves further from the origin, the incremental frame change vectors also grow in size. Note that the largest vector occurs at the end of the sequence of incremental frames, much as it would in any forward primitive.

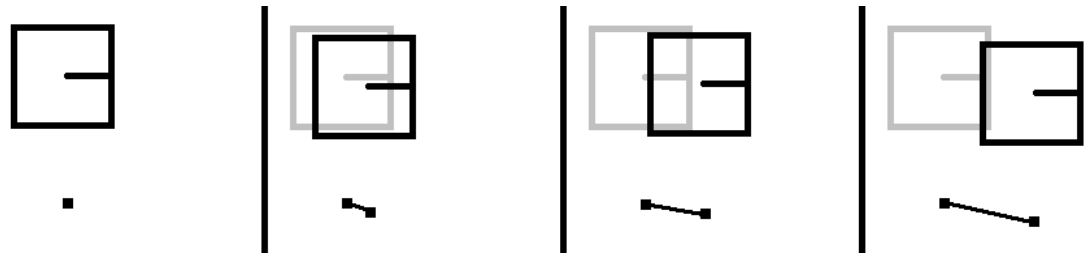

Figure 3.1: Four overhead frames (L-R) from the execution of the *forward* primitive, with the origin overlayed in grey, and the resulting frame change vector underneath.

In my attempts at training HMMs using incremental changes for all three dimensions of the frame change vectors (x, y, and θ) gave poor results overall. The *forward*, *backward*, and *stop* primitives were extremely poorly classified at less than 50% accuracy. The *left* and *right* primitives on the other hand were both recognized with very high accuracy (over 90% for both). I realized that there was some value to processing the vision frames using incremental changes for all three dimensions (at least for the *left* and *right* primitives), but I needed a better processing technique for the remaining primitive types. I took a closer look at the incremental vectors by plotting them, putting the x and y dimensions of the frame change vectors on these axes, and using θ for the z-axis. The same plotting method was used to validate my clustering algorithm, and as a result all the images displaying these incremental vectors also show colour-coded clusters. Since incremental change for the x, y, and θ values of the vectors classifies the *left* and *right* sequences so well, I call this data processing technique *rotational* processing. Figure 3.2 shows three views (a frontal view, side view, and top view) of all the data points from all of the primitive training data when it is processed using this rotational method. The x, y, and θ values of each data point represent the incremental change from that sequence's originating

data point, and only visible axes are labelled (this is also the case in Figures 3.3 and 3.4 that follow).

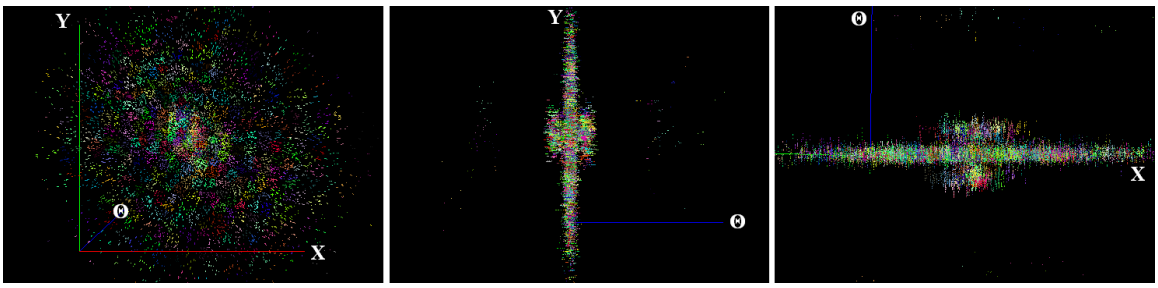

Figure 3.2: The rotational visual sequence processing method, shown for all data points from all primitive training sequences. Left to right: front view, side view, top view.

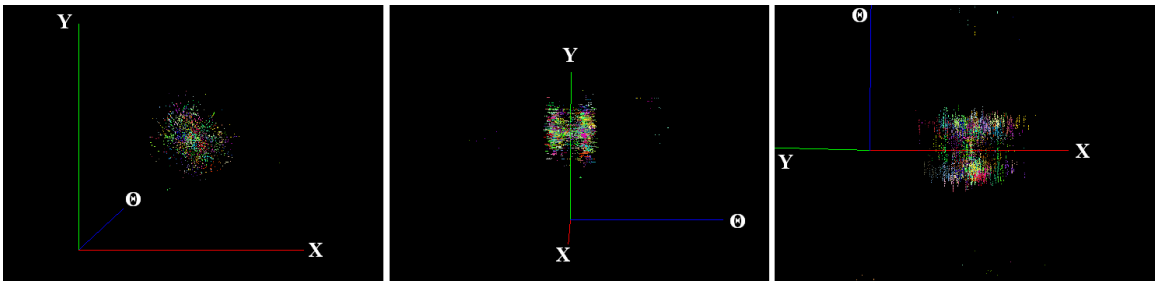

Figure 3.3: The rotational visual sequence processing method, shown for only the *left* and *right* primitives' data points. Left to right: front view, side view, top view.

Figure 3.3 shows the same rotational processing method, but with only the data points from the *left* and *right* primitives displayed. These two primitives can be seen to be quite distinct from the aggregated primitives in Figure 3.2: the left and right data points are in almost entirely separate locations when clustered using the rotational technique. This is because left and right turns make very distinct changes to the robot's orientation (left increases the angle, right decreases it). Figure 3.4 shows the data points from the *left* and *right* primitive sequences separately, processed using the rotational method, to further illustrate the distinct separation. The *left* and *right*

primitives essentially grow in increments from a θ value of 0 in opposite directions. I believe it is the separation of the data points that leads to a higher classification accuracy, and is what led me to experiment further with data processing methods to separate the *forward*, *backward*, and *stop* primitives in a similar fashion.

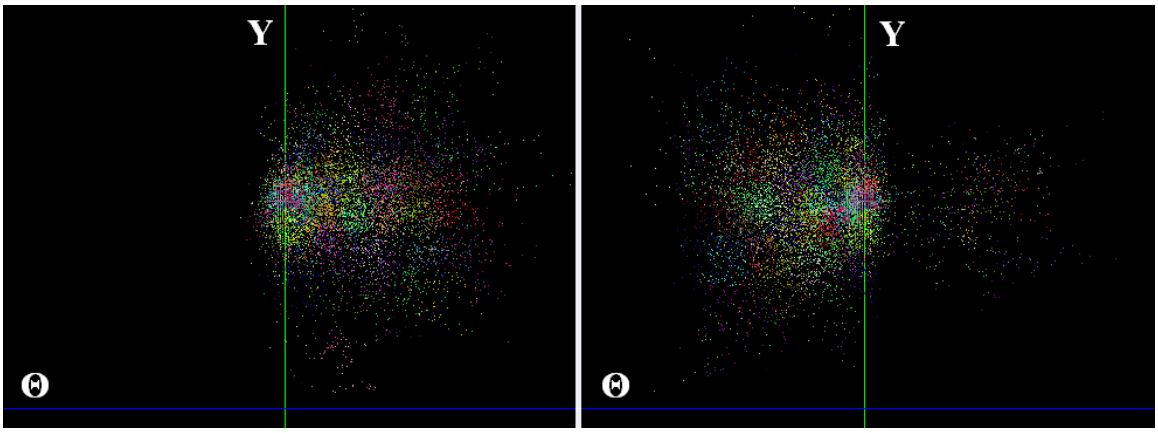

Figure 3.4: The rotational visual sequence processing method, shown from the side for the *left* and *right* primitives' data points respectively.

The data points for the *forward*, *backward*, and *stop* primitives using the rotational data processing method are hard to distinguish. The *stop* primitives are slightly easier to tell apart from the rest, but when used to train HMMs, the classification accuracy for these three primitive types is very low. The reason for this is that the incremental change in θ causes large differences for *left* and *right* primitives as described previously, but almost no differences for *forward*, *backward*, or *stop*. The problem with representing θ with an incremental change for the *forward* and *backward* primitives is that the incremental change will always hover around zero, as both of these primitives have tiny changes in θ over time, and roughly equally in both directions. This makes the incremental value of θ useless when trying to separate *forward* and *backward* primitives. Figure 3.5 illustrates this: the left image shows the

change in an agent's position, but without the absolute orientation, it could represent either a forward or backward motion. If absolute orientation is added (i.e. the actual value of θ), the difference between a forward (middle image) and backward (right image) becomes obvious.

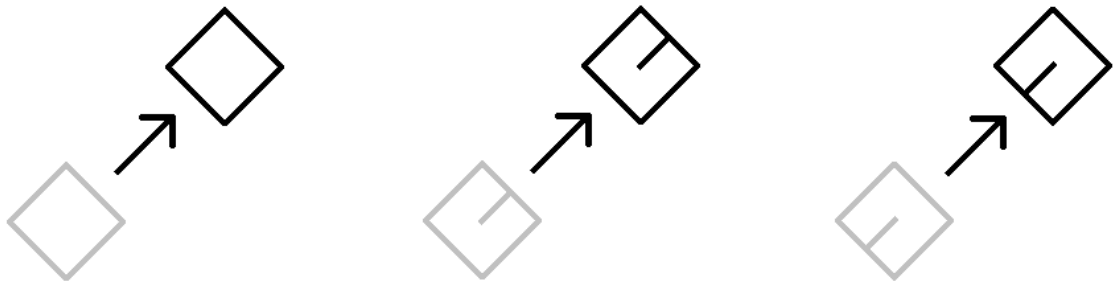

Figure 3.5: Difficulty distinguishing the difference between forward and backward motion. Without absolute θ, the left image could represent either a *forward* (middle) or *backward* (right) primitive.

The incremental changes of x and y are still very useful for separating *forward* and *backward* primitives. The best result for classifying *forward* and *backward* primitives is given by a data processing method that uses incremental changes for x and y, but instead of an incremental change for θ, the actual value of θ is used. The HMMs trained using data that was processed this way resulted in very high classification of the *forward* and *backward* primitives (over 98%), but poor classification for the *left*, *right*, and *stop* primitives. This is essentially the inverse of the results of the rotational processing technique. It should be noted that in both data processing methods, the *stop* primitive was always classified poorly. This was resolved in a different manner, as the vision system was changed to allow for a simple manual filtering method to classify stop commands. This is discussed in Section 3.2.2. Since processing the data using incremental changes for x and y and the actual value of θ

resulted in excellent classification of the *forward* and *backward* primitives, I refer to this processing approach as *positional* processing. Figure 3.6 shows three views (a frontal view, side view, and top view) of all the data points from all of the primitive training data when it is processed using this positional method. The x and y values of each data point represent the incremental change from their sequence's originating data point, while the value of each data point is the actual value of θ. This is also the case in Figures 3.7, 3.8, and 3.9 that follow. The *left*, *right*, and *stop* primitives when separated from this aggregate, can be seen (Figure 3.7) to be quite distinct from the *forward* and *backward* primitives using the positional processing method. Figures 3.8 and 3.9 show the large separation between data points of *forward* primitives and *backward* primitives processed using the positional method.

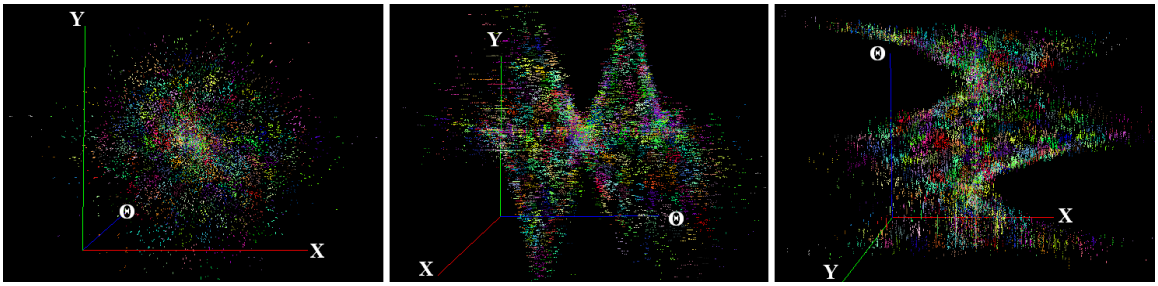

Figure 3.6: The positional visual sequence processing method, shown for all data points from all primitive training sequences. Left to right: front view, side view, top view.

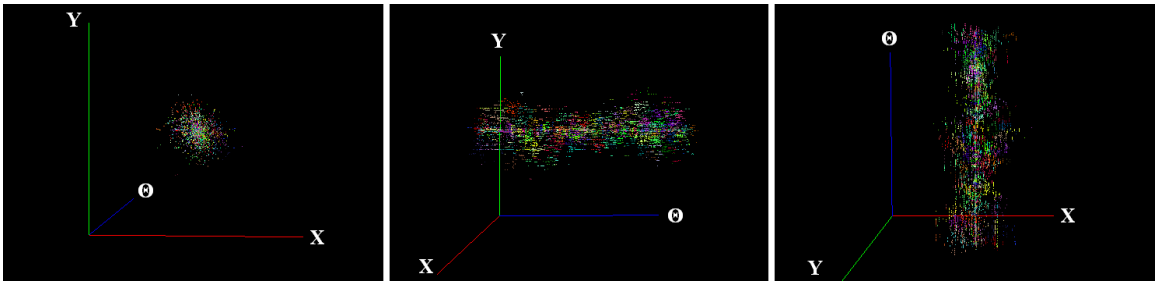

Figure 3.7: The positional visual sequence processing method, shown for only the *left*, *right*, and *stop* primitives' data points. Left to right: front view, side view, top view.

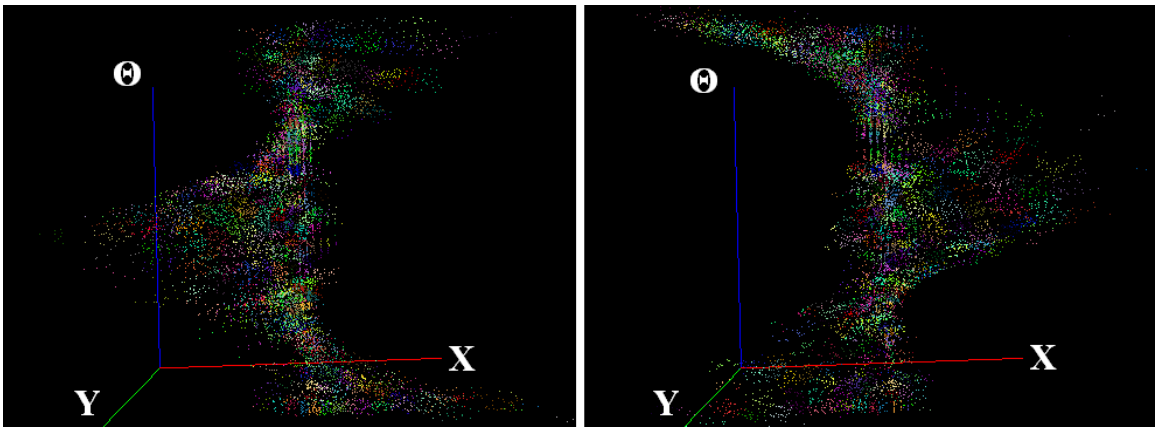

Figure 3.8: The positional visual sequence processing method, shown from the top for the *forward* (left) and *backward* (right) primitives' data points.

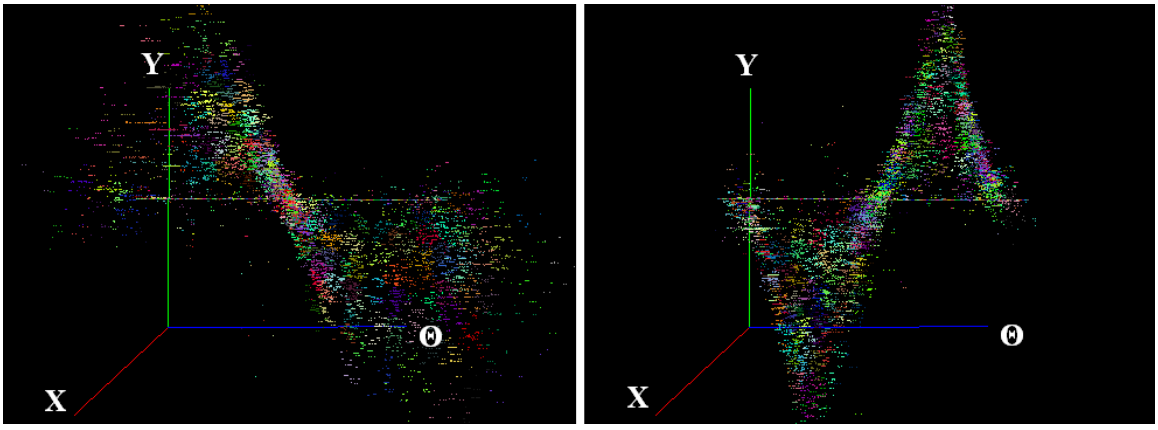

Figure 3.9: The positional visual sequence processing method, shown from the side for the *forward* (left) and *backward* (right) primitives' data points.

In the next section I will detail the process of implementing the HMMs used in my approach.

3.1.3 Programming Discrete Hidden Markov Models

I chose to use HMMs to recognize the visual effects of the imitator's primitive actions in my approach, because they are commonly used to recognize sequences of time-sensitive events [Rabiner and Juang, 1986] such as speech recognition, and handwriting recognition [Rigoll et al., 1996]. Section 2.6 reviews the background literature on HMMs that is necessary to understand my implementation of them.

There are a number of implementations of Hidden Markov Models available, but I wanted to develop them myself, both to better understand them, and to have the HMMs tightly coupled to my thesis implementation. This leaves the possibility for future work running the imitation algorithm in real-time, as opposed to having to incorporate a Matlab or other HMM implementation offline. It also offers an additional contribution from my thesis research, since I found that the literature on HMMs from an implementational standpoint was not as detailed in some respects as it should be. I began with Rabiner [1989]'s introduction to learn HMM theory and develop my code, as this reference is cited by essentially any paper involving HMMs. Rabiner and Juang [1986] wrote an earlier introduction that is also quite useful, though Rabiner's paper has more detail on implementation issues.

I will define the terms used here as Rabiner defines them in his paper, as I will be referring to a number of his equations:

- λ represents the Hidden Markov Model itself.

- $\alpha_t(i)$ is the *forward variable*, which represents the probability of the partial observation sequence until time t ending up in state i, given the model (HMM) [Rabiner, 1989]. The formal definition of $\alpha_t(i)$ is given in equation 3.1.

$$\alpha_t(i) = P(O_1 O_2 \dots O_t, q_t = S_i|\lambda) \tag{3.1}$$

- $\hat{\alpha}_t(i)$ is the scaled version of $\alpha_t(i)$, used to prevent the values being calculated from getting too small to be represented properly.

- a_{ij} represents the probability of a transition from state i to state j.

- $b_j(O_t)$ is the probability of making the observation O_t (the observation symbol from the sequence O at time t), when in state j.

- $\beta_t(i)$ is the *backward variable*, which represents the probability of the partial observation sequence from time $t+1$ to the end of the observation sequence, given that the model was in state S_i at time t. The formal definition of $\beta_t(i)$ is given in equation 3.2.

$$\beta_t(i) = P(O_{t+1} O_{t+2} \dots O_T, q_t = S_i|\lambda) \tag{3.2}$$

- $\hat{\beta}_t(i)$ is the scaled version of $\beta_t(i)$, scaled for the same reasons as $\hat{\alpha}_t(i)$

- c_t is the scaling coefficient at time step t

One major issue with implementing HMMs is the problem of numeric representation. HMMs deal with extremely small probabilities that can go out of the range of most floating point representations [Rabiner, 1989]. Rabiner proposes a scaling

procedure to guard against values becoming too small to be represented when implementing the reestimation of HMM parameters. The reestimation procedure is what iteratively adjusts the HMM's parameters to fit the training data. Using Rabiner's original scaling equations in my work resulted in models that did not train properly. Values of $-\infty$ cropped up, the model's fit to the data did not improve, and state transition probabilities did not sum to 1 as they should have. After double checking my own implementation to ensure it reflected Rabiner's algorithm, I sought errata on the paper. I did find errata on the scaling equations [Rahimi, 2000], the major issue being equation 3.3 (equation *92a* in [Rabiner, 1989]).

$$\alpha_t(i) = \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t) \tag{3.3}$$

One of the errata supplied in Rahimi [2000] points out a subtle error in the original equations, namely the i and j variables should switch places. The correct version of equation 3.3 is given by Rahimi, shown here as equation 3.4. I have modified it slightly to make it more directly comparable to Rabiner's equation 3.3 (simply changing $t+1$ to t, and t to $t-1$).

$$\alpha_t(j) = \sum_{i=1}^{N} \hat{\alpha}_{t-1}(i) a_{ij} b_j(O_t) \tag{3.4}$$

Equation 3.3 is used by Rabiner [1989] as the basis for calculating the scaling coefficient, leading to equation 3.6 (equation *92b* in [Rabiner, 1989]). Since equation 3.6 is based on the incorrect equation 3.3, using equation 3.6 resulted in errors during training of the HMM. An improved version using Rahimi's changes is given in equation 3.7. It would seem that most errors in the original Rabiner paper come from copying

errors. Equation 3.3 is supposed to reflect the induction formula 3.5 (equation *20* in [Rabiner, 1989]). It can be seen clearly that when repeating the formula in his section on scaling, Rabiner accidently transposed the i and j variables (causing the a_{ij} term to represent the incorrect state transition).

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \tag{3.5}$$

$$\hat{\alpha}_t(i) = \frac{\sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ij} b_j(O_t)} \tag{3.6}$$

$$\hat{\alpha}_t(j) = \frac{\sum_{i=1}^{N} \hat{\alpha}_{t-1}(i) a_{ij} b_j(O_t)}{\sum_{j=1}^{N} \sum_{i=1}^{N} \hat{\alpha}_{t-1}(i) a_{ij} b_j(O_t)} \tag{3.7}$$

After using equation 3.7, I still found that training was performed incorrectly, and had to diagnose an additional problem with this equation. The problem was that the transition probabilities did not sum to 1 for each state. This resulted in improper training, as transition probabilities are an essential part of Hidden Markov Models.

It appears that when moving equation 3.5 to his section on scaling, Rabiner [1989] missed one more step, which seems to have been overlooked by Rahimi [2000] as well. The sum in the induction step is clearly defined with brackets to contain the sum of the product of the terms $\alpha_t(i)a_{ij}$. This summation represents the probability of being in state j at time $t+1$, with all previous partial observations accounted for at this point in the observation sequence. Once this sum is obtained, it is then multiplied by $b_j(O_{t+1})$, which represents the probability of the observation O_{t+1} occurring when in state j. The entire equation therefore represents the probability of being in state j, at time $t+1$, and making the observation O_{t+1} [Rabiner, 1989]. As can be seen from

Rabiner's scaling equation (equation 3.6), the product that is summed also includes the $b_j(O_t)$ term, while the original induction step 3.5 that the scaling coefficient was based on does not.

I determined that this incorrect summation was almost certainly the reason for the inaccuracies in the transition probability tables for my HMMs. I did find some verification for this in the form of notes [Stamp, 2004] which gave an introduction to HMMs with pseudocode. Interestingly, the pseudocode for the scaling equation was as I expected, but when giving the formula in mathematical notation, Stamp also made the summation error. The final equation I used for scaling is given in equation 3.8. When using this equation the transition probabilities properly sum to 1.

$$\hat{\alpha}_t(j) = \frac{\left[\sum_{i=1}^{N} \hat{\alpha}_{t-1}(i)a_{ij}\right] b_j(O_t)}{\sum_{j=1}^{N}\left[\sum_{i=1}^{N} \hat{\alpha}_{t-1}(i)a_{ij}\right] b_j(O_t)} \tag{3.8}$$

It should be noted that even when using scaling to prevent representation errors, the *forward algorithm* which calculates the probability of an observation sequence given an HMM (i.e. how well the observation sequence fits that HMM) still gives a result far too small to be represented [Rabiner, 1989]. For this reason, the forward algorithm results in a probability that is actually the log of the real probability, keeping the result within the representation range.

The errata given by Rahimi [2000] also expanded on Rabiner's equations for training HMMs on multiple observation sequences. I supply these errata equations here, as they are used directly in my work and are thus required to reproduce it. I did make one improvement for efficiency reasons. It can be seen that equations 3.9 and 3.10 have identical denominators. To help speed up the training process, I saved the

denominator for each i when $\bar{a_{ij}}$ was calculated, then used those saved denominators for corresponding i values when calculating $\bar{b_i}(l)$. To verify that the result was the same, I ran both versions (regular and with my improvements) and confirmed that they produced identical results.

$$\bar{a_{ij}} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_{k-1}} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{k=1}^{K} \sum_{t=1}^{T_{k-1}} \hat{\alpha}(i) \hat{\beta}(i) \frac{1}{c_t}} \tag{3.9}$$

$$\bar{b_i}(l) = \frac{\sum_{k=1}^{K} \sum_{t=1, s.t. O_t = v_l}^{T_{k-1}} \hat{\alpha}_t(i) \hat{\beta}_t(i) \frac{1}{c_t}}{\sum_{k=1}^{K} \sum_{t=1}^{T_{k-1}} \hat{\alpha}(i) \hat{\beta}(i) \frac{1}{c_t}} \tag{3.10}$$

The algorithm used to calculate the probability of an observation given the HMM is the *forward-backward procedure* [Rabiner, 1989]. The *forward-backward procedure* is used to train the models, but since the desired result here is the probability of a single sequence, only the forward part of the calculation needs to be used [Rabiner, 1989]. The main step of the forward procedure is the induction step (equation 3.5. The training algorithm is known as the *Baum-Welch* algorithm [Rabiner, 1989] and is used to update the state transition probabilities (using equation 3.9), and observation probabilities (using equation 3.10) to better fit the training data. The Baum-Welch algorithm adjusts the HMM parameters using these two equations until no more improvements in accuracy are made [Rabiner and Juang, 1986; Yang and Xu, 1994].

In the next section I will describe the configuration and training process of the HMMs used in my approach, as well as the various attempts which led me to the final configuration.

3.1.4 Training Hidden Markov Models

Designing Hidden Markov Models requires some thought on the topology of the model: that is, the way the states are connected. This is defined by the set of state transition probabilities. If a transition between state i and state j is 0, then there is no way to reach state j from state i (the training of an HMM will never change a state transition from a 0). There are obviously many different ways to design HMM topologies, though there is no set method to determine an optimal topology [Rabiner, 1989]. A very common configuration is the *left-right* topology, where each state leads to itself, with two forward links. These forward links mean that the state can only transition to the state directly to its right, and one more state beyond that. I used this topology because it fits with the sequential nature of the vision data I was using to train the HMMs. Figure 3.10 shows an example of three states connected in this manner. Left-right topologies are also used in the handwriting recognition work of Rigoll et al. [1996], though they only use a single forward link. I tried a number of other configurations, such as having greater or fewer links per state, and connecting all states to all other states, but there were no gains in the HMM classification accuracy.

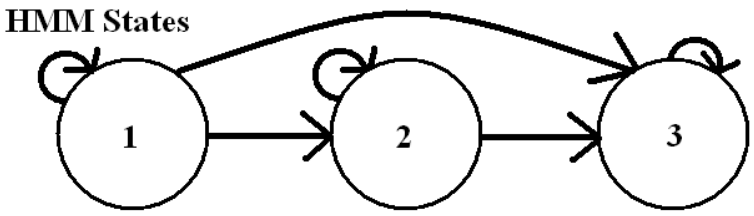

Figure 3.10: An example of an HMM with the *left-right* topology with two forward links.

As mentioned in Section 2.6, I chose to use discrete HMMs for my work, as they are

generally faster to train and use than continuous HMMs [Rigoll et al., 1996; Inamura et al., 2002]. The observation sequences used to train the HMMs in my approach are the sequences of visual frames gathered from the imitator's primitives. Discrete HMMs require a set of discrete symbols to represent all possible observation symbols of a sequence, so the visual frames need to be processed into a codebook of discrete symbols [Rabiner, 1989]. Each symbol in the codebook represents one of the possible categories to which distinct visual elements of the primitives' visual outcomes can be classified.

To convert the visual data of the primitives into discrete symbols, the vision data is clustered using the k-means algorithm [Hartigan and Wong, 1979]. The vectors used in the clustering algorithm are all three-dimensional, relating to the x, y, and θ (orientation) values of the robot. The θ values are in radians ranging from $-\pi$ to π. To ensure that the clustering algorithm would not put too much weight on the proximity of the θ values, I scaled the θ values by a constant (16, a value obtained during experimentation). This brought it more inline with the x and y values, which are in millimeters, allowing for more distinctly spaced clusters. The clusters are calculated using all 5 of the primitive data sets (for a total of 5000 sequences, and approximately 125,000 individual data points). Through experimentation I discovered that 512 clusters (the centroid splitting algorithm that I described in Section 2.7 generates clusters in powers of two [Linde et al., 1980]) is an optimal number of discrete observation symbols for the HMMs in my approach.

I convert each vision frame from the primitive visual data into a symbol from the codebook (one of the 512 centroids that represents a central point in one of the

previously mentioned clusters). This process converts primitive visual data from a sequence of vectors containing x, y, and θ values, into sequences of discrete symbols, the centroids. Each of these sequences of centroids is then divided amongst HMM states, so that each state can make a good estimate of the initial probability of any given observation symbol (centroid) occurring at that particular state.

Training the HMMs with initial observation probabilities that are close to the training data allows the models to train faster [Rabiner, 1989]. To get these initial probabilities, I counted the number of occurrences of each observation symbol (centroid) in a state. I then divided these sums by the total number of observation symbols in that state to get the initial observation probabilities for that state. This is similar to a technique Rabiner [1989] describes when discussing updating the estimate of the observation probabilities using the k-means clustering algorithm [Hartigan and Wong, 1979]. I ensured that all of the observation probabilities had a non-zero initial probability, as Rabiner [1989] suggests. Even though a particular observation symbol may not appear in the training data for a particular state, it does not mean that it will never occur in an unknown observation set [Rabiner, 1989]. The minimum probability of an observation symbol occuring is kept very small (10^{-8}), and I made sure to normalize all probabilities during my calculations to ensure that observation probability distributions summed to 1 at any given time.

To calculate the initial observation probabilities for my HMMs, I first needed to determine which centroids belonged to which state. Primitives are first converted into sequences of centroids, or discrete observation symbols that represent the visual effects of a primitive over time. I chose to have each state represent the passage

of time through the execution of a primitive, to give the HMM states a temporal ordering. The first state represents the beginning of a primitive, while the last state represents the end (the states in-between the first and last states fill out the rest of the primitive). For example, in my approach, the last state of an HMM uses the centroids that occur at the end of all of the individual training sequences for that HMM's primitive type. These centroids are then used to calculate that end state's initial observation symbol probabilities. This is done similarly for the other states (though the first states will take centroids from the beginning of the primitives, middle states from the middle an so on).

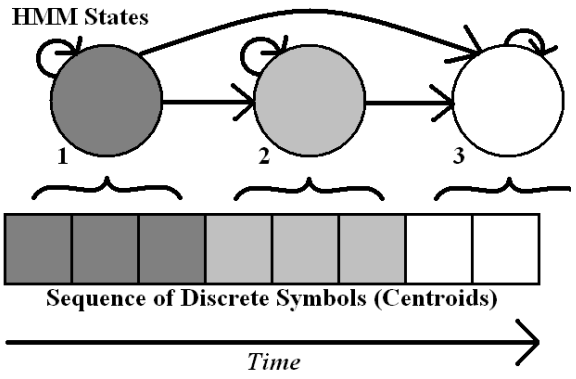

Figure 3.11: Sequences of discrete observation symbols (centroids) from the training data are separated into segments based on when they occur in the sequence.

To divide these appropriately, each sequence of centroids is segmented into evenly sized pieces, representing equal portions of the primitive in terms of the number of centroids. The number of centroids in each segment is determined by dividing the number of centroids in the sequence by the number of states in the HMM, as seen in Figure 3.11. Sequences of discrete observation symbols (centroids) from the training data are separated into segments based on when they occur in the sequence. States

represent specific temporal orderings, and receive the centroids that correspond to their order in the model (first state gets centroids from the first segment, last gets last segment, etc.). The number of segments is determined by dividing the total number of centroids (8 in Figure 3.11) by the number of states in the HMM (3 in Figure 3.11), then rounding up (to 3 segments for this example). It is possible for the last state to have fewer frames as a result, though it will always contain at least the last centroid of the primitive. For example, if a primitive is converted into a sequence of 20 centroids, and the HMM has 3 states, the first state would have centroids 1 to 7, the second state would have centroids 8 to 14, and the third and final state would get the remainder of the sequence, centroids 15 to 20. In Figure 3.11, the first two states each get 3 centroids, but the last state is only assigned the two remaining centroids. Essentially, each state is further in time than the previous state, and contains multiple centroids from a primitive's centroid sequence (not just one centroid per state). I did try topologies with one centroid per state, but these proved to be slow and showed no improvement.

Once I decided how to distribute the observation symbols amongst states, I needed to determine how many states the HMMs in my implementation should have. I tried many different numbers of states and compared the accuracy of the models. My training set had a total of 5000 sequences, 1000 for each of the 5 primitives, *forward, backward, left, right* and *stop*. The test data had a total of 1804 sequences, with over 330 sequences of each type (i.e. 36% as many sequences as all the training data, or over 33% of each individual sequence type). I began with 5 states in my initial experiments, and received fairly promising results on my test data. I then

ran experiments on 6, 7, 8, 9, and 10 states. To my surprise, I noticed very little to no improvement. I continued increasing the number of states, testing 15, and 20 states. There was still very little if any improvement over the 5 state model, and the 5 state model was much faster when training, and calculating probabilities of unknown sequences.

I became suspicious and decided to test how much of the work was actually being done by the HMMs. I tried training an HMM with a single-state, and discovered that it was just as accurate at classifying the test data as all the other previously trained models, and far faster than any of them. The training didn't do very much at all: since there was only one state, there was only one link (to itself) and therefore the training never changed the state transition probabilities. The observation probabilities were set from the training data itself, and since it was set for only a single-state, these never changed either. The success of the single-state HMMs led me to believe that the data processing methods discussed in the Section 3.1.2 were actually responsible for much of the classification work.

After settling on the rotational and positional data processing techniques described in Section 3.1.2, I realized through observing the data points just how much work the data processing was doing. I believe the incremental x and y values are doing most of the separation. When converting the imitator primitives to incremental frames, the greatest differences from the origin are at the end of the primitive. This is due to the fact that the end of the primitive is the greatest state change (furthest travelled in position, or greatest amount θ has changed when turning), since change is measured from the origin frame. The primitives in my work make incremental changes to the

robot's originating position, causing the incremental frame change vectors to move further away from the primitive's originating vector with each subsequent frame. A primitive's motion does not oscillate frames. That is, a *forward* command will only move the robot forward, it will not have any frames where the robot moved backward slightly: if it did, that would start a new *backward* primitive. Similarly, the *left* and *right* primitives are always increasing or decreasing in θ for each frame, because they never turn further than 360 degrees, and if the rotational direction changed, that would define a new primitive. Figure 3.1 illustrates this using incremental x and y changes: of the four frames the last has the greatest change, because this summarizes the change over all frames. This representation causes the end points to distinguish the different primitive sequences so much that the number of states in the HMMs end up being mostly irrelevant given my data. The state that contains these largest incremental changes (the last few frames of a primitive sequence) becomes the deciding factor. This means that in multiple-state HMMs, it is essentially the last state that is doing the classification. This makes the preceding states superfluous in my implementation, allowing a single-state HMM to classify sequences with the same accuracy as an HMM with many states, albeit much faster.

Since I use two data processing methods, I also have two possible ways to interpret visual data when determining the primitive type of an unknown visual sequence. The HMMs trained to recognize *forward* and *backward* primitives require that the data they try to classify be processed using the positional processing method. Similarly, the HMMs trained to recognize *left* and *right* primitives require that their data be processed using the rotational processing method. The clustering algorithm is run

after the data processing, so it is imperative that an HMM processes unknown visual sequences using the same data processing method it used during training. This is because the discrete symbols (the 512 centroids generated by the clustering algorithm) used by an HMM are from clusters generated by the data processing technique that was used to train that HMM. The 512 centroids created from data points processed using the rotational method will be very different from the centroids created from data points processed using the positional method. In my HMM implementation, I save the data processing method that was used to train an HMM, so that an HMM can easily convert visual data using its required data processing method. This way, when converting visual data to sequences of centroids, the HMM will have the proper visual data representation, to ensure that the clusters match the ones that the HMM was trained with. The centroids themselves are also saved by each HMM during the training process.

Implementing the HMMs was a useful experience, as they provide perfect structures to represent and calculate the probabilities of unknown observation sequences. It was also an excellent learning experience, which led to a simple but effective set of single-state HMMs for classifying primitives from segments of the visual stream. While this was an unanticipated outcome of my research, it is important to put it in perspective: the use of HMMs to recognize primitives from visual streams is one small part of my overall imitation learning architecture. The behaviour learning process of my work (Section 3.3) requires a sequence of primitive symbols, but the method of acquiring these can vary. Any process that can recognize primitives from visual data and create sequences of primitives symbols could be used as an alternative. I

chose HMMs simply because I thought that their ability to model time-sensitive data would work well with the sequential frames collected from the vision data by Ergo. In the next section I will discuss the development of my system to convert the visual sequences of demonstrations into sequences of primitive symbols.

3.2 Converting Demonstration Visual Streams into Sequences of Primitive Symbols

In this section I will first discuss the process of gathering the visual data from demonstrations and converting them to sequences of primitive symbols. I will detail the robots used for demonstrators as well as their control programs in Section 3.2.1. In Section 3.2.2 will discuss the attempts that led to a final conversion method using the HMMs trained using the techniques described in Section 3.1.

3.2.1 Collecting Demonstration Visual Streams

In my work I used three physically distinct robots, as shown in Fig. 1.1. The robot imitator is a two-wheeled robot built from a Lego Mindstorms kit. One of the four demonstrator types is a Bioloid humanoid robot which uses a cellphone for vision and processing. The control program for the Bioloid robot was created by Jonathan Bagot for competition in RoboCup in 2008 [Baltes et al., 2008]. One of the demonstrators is physically identical to the imitator, but uses a minimally modified control program obtained from the University of Manitoba Autonomous Agents laboratory that was used in a RoboCup competition in 2004 [Anderson et al.,

2004a]. Since it is a competition program, it plays extremely well. There are two more demonstrators that are physically identical to the imitator. One is programmed to simply drive at the ball with no other behaviours, and represents a poorly skilled (since this behaviour does not properly position the robot for good aim, and will be less likely to score). I also use an imitator trained with the system presented in this thesis as a demonstrator. This demonstrator is physically identical to the imitator, and represents a demonstrator of average skill, in contrast to the poorly skilled demonstrator that drives at the ball, and the more highly skilled fine-tuned RoboCup 2004 demonstrator. The imitator-based demonstrator is also used to verify that my imitation learning architecture can pass knowledge acquired by one imitator to another through demonstration. The last demonstrator is a Citizen Eco-Be (version I) robot. This is physically similar to the imitator in that it is also a two-wheeled robot, but it is about 1/10 the size of the imitator.

The demonstrator visual data is gathered in much the same way as the imitator primitive data. The demonstrators are recorded taking shots on goal with the same Ergo global vision system [Baltes and Anderson, 2007]. I also record the command that each demonstrator sends during each frame recorded. This allows me to evaluate the process of converting visual demonstrator data into sequences of primitive symbols, since I can compare the output of my approach to the actual actions the demonstrator executed. When recording demonstrations, the demonstration is considered invalid if the robot breaks down or the vision system stops tracking the ball or robot. A demonstration is considered to be complete when the ball or robot leaves the field. Demonstrations are recorded with only the demonstrating robot and the ball

on the field. For each demonstrator type, 50 demonstrations in 2 field configurations (described in Chapter 4) with 25 demonstrations each were selected randomly from all valid demonstrations. In the next section I will detail the process of converting visual data to sequences of primitives.

3.2.2 Matching Primitive Symbols to Visual Data

When first attempting to convert demonstration visual data to a sequence of primitives, I used two stages of conversions, inspired by the separation of data points in the rotational and positional data processing techniques described in Section 3.1.2. The data processing techniques were very good at separating *forward*, *backward*, and *stop* primitives from *left*, *right*, and *stop* primitives, but were not effective at isolating the *stop* primitive. I trained two HMMs, one to recognize *forward*, *backward*, and *stop* primitives, and another to recognize *left*, *right*, and *stop* primitives. Once visual data was separated into two sets using these HMMs, I then tried to use HMMs that were trained on the individual primitives to properly classify the data as a specific primitive type. For example, a *forward* primitive would first be classified as belonging to the group of *forward, backward, stop*. Once this was determined, three HMMs were used (one for *forward* primitives, one for *backward*, and one for *stop*) to classify the *forward* primitive between the remaining three possible types of primitives. After many attempts at this approach with little success, independent work on the Ergo vision system provided a solution.

Once of the main reasons that I initially tried to use HMMs to classify the *stop* primitive was because the jitter inherent in Ergo's vision data [Baltes and Anderson,

2007] made it impossible to tell if the robot had actually remained stationary during a short time period. Improvements were made in Ergo that significantly decreased this jitter in the x and y coordinates, which allowed me to define a suitable classification for *stop* primitives. A *stop* primitive is recognized if the robot's position has moved less than 1 millimeter during a segment of a visual stream, and the rotation has moved less than 20 degrees (the rotational jitter still remains). These minimal motion values were determined through preliminary experimentation. I also check to see if the previously classified primitive is a *stop* primitive and do not append *stop* segments to other *stop* segments, to avoid having long chains of *stop* primitives. This is possible because consecutive *stop* primitives are always equivalent to a single *stop* primitive (no matter how many *stop* primitives might occur together, the net effect is that the robot has not moved). With this stop filtering method in place, the HMMs trained using the rotational and positional data processing techniques can separate the remaining primitives with over 90% accuracy.

In my implementation, the initial step in converting a demonstration to a sequence of primitives is to segment the demonstration. A demonstration is segmented so that each segment is an atomic piece of a demonstration, in that it can be described by a single imitator primitive. This is achieved by creating a segment whenever the demonstrator has moved (or turned) too far for any of the imitator's primitives to possibly match the motion. I also used a minimum number of frames for segments, in order to give the HMMs performing the classification a reasonable amount of motion data to work with. The HMMs can try to classify a single frame as a primitive, though this is futile, as the motion data contained in one frame is so small that there

is no way to differentiate between primitives. I use a minimum of 10 frames for each segment, which was determined during preliminary experimentation. I do not use a maximum value, as the segments are only cut off when the motion exceeds the amount of movement possible by any of the imitator's primitives. I determine *stop* primitives using a similar process: if the minimum number of frames has already occurred, but the motion contained in them is very low (the position of the robot has changed less than 1 millimeter, and the orientation less than 20 degrees), the segment is classified as a *stop* primitive.

If it is determined that a segment is not to be classified as a *stop* primitive, the segment is analyzed by the *forward*, *backward*, *left*, and *right* HMMs. The HMM that provides the highest probability of a match to the segment is chosen, and its primitive is used as the symbol to classify the segment. If two or more HMMs produce probabilities that are too close to each other (within 2%, a value obtained during experimentation) for a given segment, there is no clear frontrunner to classify that segment confidently. In this case it is considered too close to call, and the segment is instead classified as a *gap*. A gap in a demonstration represents a part of the demonstration where no primitives are capable of achieving the same state change as the segment, and so a higher level abstraction over individual primitives is needed to cover these gaps. For example, if a humanoid demonstrator performs a side-step, and a wheeled robot cannot achieve that same motion with any of its primitives, such a gap would occur. This is to be expected when using demonstrators that are heterogeneous.

Once a demonstration broken up into segments labelled with primitive symbols,

these sequences of primitive symbols can be used to reason about the demonstration. By analyzing these sequences of primitives, new behaviours can be learned that encompass various primitive sequences. The next section details the process of creating behaviours in my implementation.

3.3 Creating and Managing Behaviours

Having explained how primitives are recognized, I now turn to explaining how these are built up into more elaborate behaviours that can be used to abstract the activities of a demonstrator. In the first section I will detail the various components that make up a behaviour in my implemented system. In Section 3.3.2 I will discuss the behaviour creation and deletion processes in detail. Following that, in Section 3.3.3 I will discuss the predictive abilities of the behaviours in my system.

3.3.1 Behaviours and their Attributes

A behaviour in my system is a combination of primitives and other behaviours that occur frequently. Since agents begin with only primitives, this means the first behaviour that will be created is a combination of two primitives that are often recognized as useful when used sequentially. Figure 3.12 illustrates the creation of a behaviour composed of two *left* primitives in sequence. The primitive sequence that the behaviour encompasses becomes saved inside the behaviour as a list of primitives. Note that there is a frequency attribute as well: the nature of this value will be explained in the next section, where the behaviour creation and deletion process is discussed.

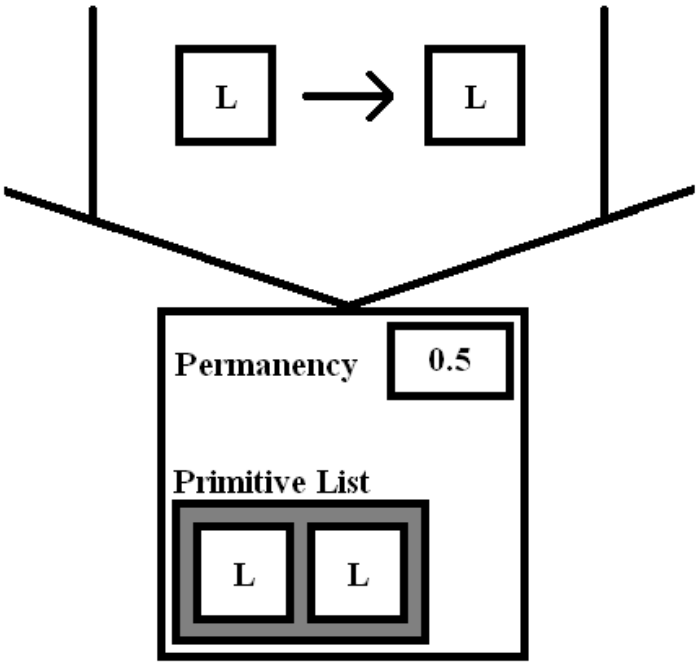

Figure 3.12: A new behaviour is created to encompass two *left* primitives occurring in sequence.

The traditional use of behaviours in AI is for control: a behaviour dictates control values (e.g. in [Arkin, 1998; Brooks, 1986]), or based on a precondition is applied or activated much like an action in a planner. These are used in my implementation in this fashion, since the developed collection of behaviours will ultimately be controlling the imitating robot when I evaluate its own performance. The main use of behaviours in my implementation, however, is in forming abstractions above the primitive level: once the learner can reason about two primitives together, as in Figure 3.12, and it finds that this is commonly used in conjunction with another primitive, or another behaviour, it can put this together into a more complex, more abstract behaviour, as shown in Figure 3.13. Behaviours in my system will always be formed in this fashion (the implementation of this is described in the next section): two primitives will be

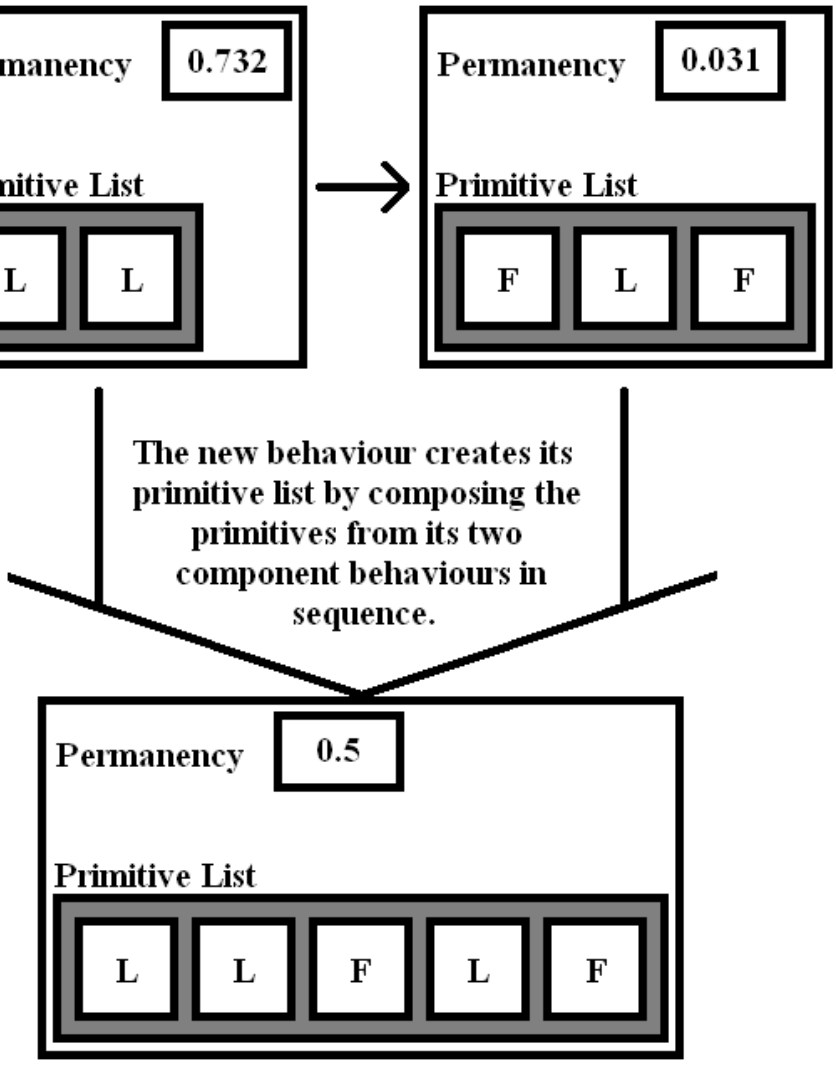

Figure 3.13: The creation of the *LLFLF* behaviour from the *LL* and *FLF* behaviours.

combined, primitives will be added to existing behaviours, or two behaviors will be abstracted together.

Once a behavior exists, not only can it be used for control, and for reasoning at a level where individual primitives don't have to be considered: we can also predict what the world will look like if the behaviour were to be carried out. For example, if the ball is directly in front of the robot, a behaviour that causes the robot to move forward

and turn left can predict not only that the robot's position will change, but also that when moving forward and left, the robot will cause the position of the ball to change accordingly. It is important that the behaviours can make accurate predictions about their effects, so that the best behaviours can be chosen to represent segments of a demonstration. This allows the imitator to translate demonstrations into higher level sequences of behaviours, which can then be used to build more complex behaviours.

Just like you can see how well a given primitive fits the current part of a demonstration you're observing, you can see how well a behaviour fits too: this involves partly matching the parts of the behaviour that have already happened (e.g. the forward in a forward left) and predicting the remainder. If the remainder turns out to be correct, that behaviour is a good explanation for the series of actions that just unfolded, and in turn might also be added to by actions that are about to happen, to create new behaviours. The more often a behaviour works in this fashion, the more useful it is to an imitator. Similarly, a behaviour might be proposed and then never be used again. There will clearly be many of the latter case if we are always trying to add new actions to behaviours, so we need a mechanism to ensure we don't have an inordinate amount of useless behaviours.

3.3.2 Behaviour Creation and Deletion

In my implementation, behaviours are created by composing sequences of primitives (and/or other behaviours). The method for selecting which behaviours or primitives are combined are explained in the next section when I discuss the predictive abilities of behaviours, and in detail in section 3.4. When the imitator first starts

learning, it begins with only its primitives: no behaviours exist. The first behaviour created by my implementation will always be a pair of primitives (as seen in Figure 3.12). More complex behaviours can then be created by composing other behaviours (or a behaviour with a primitive) into one behaviour that encompasses all primitives from all of its components. This leads to behaviours that contain more than two primitives, such as the *left, left, forward, left, forward* behaviour shown in Figure 3.13. Table B.1 of Appendix B shows the most commonly created behaviours using my approach.

A method must be used to determine which behaviours and primitives are combined to form new behaviours to avoid creating behaviours that are useless. My implementation is designed to determine the frequency with which primitives and behaviours recognized from the demonstrations follow each other in sequence. When two behaviours or primitives are found to be occurring sequentially frequently enough, a new behaviour is created to encompass them both. There is still the matter of determining which behaviours or primitives occur sequentially. This is where the behaviour's predictive abilities come into play. All behaviour's predictions about how they will affect the current state of the field are compared to the next state, and the behaviour that has a prediction that best approximates the actual outcome of the demonstration is selected to occur next. The details of how a behaviour generates its prediction are given shortly. Through this process behaviours are chosen to follow one another based on how well they predict the outcome of the demonstrations.

If behaviour A follows behaviour B in sequence, it means that behaviour A was chosen as the best prediction, and then behaviour B was chosen as the best prediction

afterwards. When a behaviour follows another in sequence, the element in the frequency matrix representing the frequency of the two behaviours occurring in sequence is increased. In Figure 3.14, the L primitive has just been predicted to follow the L primitive, and so the frequency is updated (to 0.3 in the figure). If that frequency surpasses the behaviour creation threshold of 0.3, a new behaviour is created that encompasses both of the primitives. Figure 3.12 shows the internals of two L primitives being used to create a new behaviour LL. Note that the new LL behaviour has its permanency set to the default value of 0.5 when it is created. If a behaviour is created that already exists, a second copy is not made. Instead the existing behaviour has its permanency increased, because it is useful enough to have been created more than once independently.

To keep track of how often behaviours follow each other in sequence, a matrix of behaviour frequencies is maintained (the matrix labelled "Frequencies" in Figure 3.14. All behaviours have their own unique row and column in the frequency matrix. Each column in a given behaviour's row represents the frequency that a behaviour will follow another. For example, in Figure 3.14, the L behaviour has not followed the F behaviour, and as a result the frequency at row F, column L is 0.

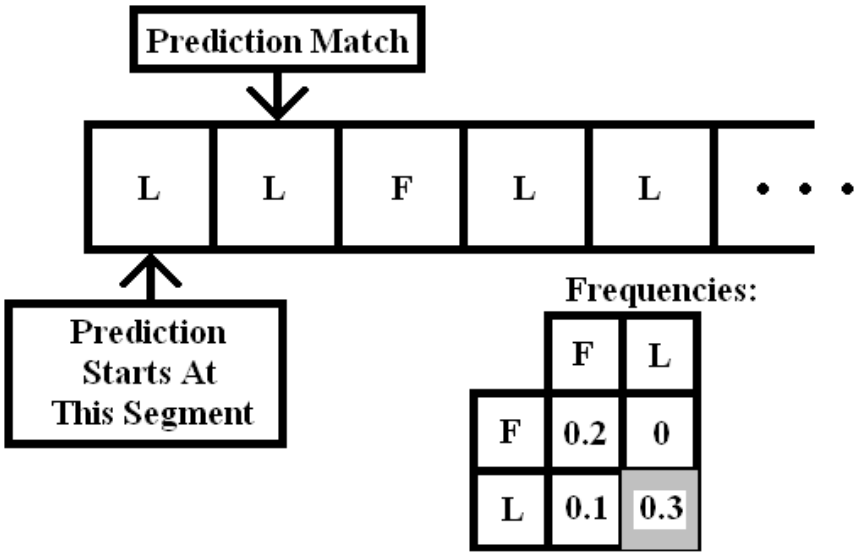

Figure 3.14: A *left* primitive is predicted to follow another *left* primitive, causing the frequency at row L and column L to be increased.

Since behaviours can be created from other behaviours, the total number of behaviours needs to be kept under control. One of the ways this is done is to represent the behaviours as their component primitives. Instead of representing composite behaviours as links to other behaviours, which may in turn have links themselves, I represent all behaviours as the sequences of primitives that were used to create them, and copy primitives when behaviours are combined. This is possible because my implementation only focuses on behaviours that occur in sequence. I did not explore compositions of behaviours that occur simultaneously, in order to keep the scope of this research reasonable, and due to my implementation platform. My imitator is a wheeled robot with no other actuators such as arms, and so there are no actions that can be carried out simultaneously.

The problem with representing behaviours as components that then link physically

into composite behaviours (which is often done when using behaviours for control (e.g. [Arkin, 1998]) is that if a behaviour is deleted because its individual contribution is not very useful on its own, it still must be maintained or the representation of larger behaviours would be lost. By representing a behaviour as the primitive sequence it entails, the components of a behaviour that later go into creating more complex behaviours remain even if that behaviour is deleted. If two primitives are used to create a new behaviour, then that new behaviour contains those two primitives. If one or more components is itself a behaviour, then the primitives used to create that behaviour are copied to create the new behaviour. For example, if the behaviour *left, left* and the behaviour *forward, left, forward* are observed occurring often enough in sequence, then a new behaviour is created that contains the list of primitives: *left, left, forward, left, forward*, as shown in Figure 3.13. If the behaviour *forward, left, forward* is later deleted, it no longer needs to be uniquely represented by the system just to support later behaviors that use it as a component. The *left, left, forward, left, forward* behaviour's primitives will remain intact even if the behaviours that the primitive list was originally built from are deleted.

As can be seen in Figure 3.12 and 3.13, behaviours in my implementation have a *permanency* attribute, which is also used to keep the total number of behaviours in check. The permanency value of a behaviour is a rough measure of the usefulness of that behaviour, and ranges from 0 (not useful) to 1 (very useful). The permanency is adjusted over time depending on how much a behaviour gets used. This permanency attribute helps to retain useful behaviours while discarding the rest. Each behaviour starts with a default permanency value (0.5). The permanency of all behaviours

decreases slowly, or *decay* over time (more detail is given in Section 3.4). In my work, the rate that the permanency of behaviours decreases is referred to as the *decay rate*.

A behaviour needs to have a mechanism to keep increasing its permanency or it risks getting deleted. This is explored in more detail in Section 3.4. Basically if a behaviour makes the best prediction about a demonstration compared to all other behaviours making predictions, that best behaviour has its permanency increased slightly. This way a behaviour that is used constantly to predict what a demonstrator will do has its permanency increased rapidly enough that it will not be deleted. Behaviours that make poor predictions will not have their permanencies increased, and therefore will eventually be deleted when their permanency reaches zero. If the permanency of a behaviour reaches an upper threshold, the behaviour is made permanent, and is no longer evaluated for permanency. Behaviours that are permanent cannot be deleted. The threshold for permanency is 0.8 which was determined during preliminary experimentation. This and other values relating to the creation and management of behaviours can be seen in Table 3.2.

Attribute Name	**Range**	**Value**	**Description**
Creation Threshold	0 to 1	0.3	The frequency of two behaviours occurring in sequence must pass this value before a new behaviour is created.
Default Frequency	0 to 1	0	The starting frequency of two behaviours occurring in sequence. After behaviour creation, frequencies that had surpassed the Creation Threshold are reset to the Default Frequency value.
Permanency Threshold	0 to 1	0.8	When the permanency of a behaviour passes this value, that behaviour becomes permanent to the forward model that it was created in.
Default Permanency	0 to 1	0.5	The default permanency that all new behaviours receive upon creation.
Deletion Threshold	0 to 1	0	When the permanency of a behaviour reaches this value, the behaviour is deleted from the forward model that created it.

Table 3.2: The different attributes used when creating and managing behaviours.

3.3.3 Using Behaviours to Approximate Demonstrations

Choosing which behaviours are used to approximate a demonstration's segments through this predictive process is important to allow for higher level abstraction over the primitives. When a behaviour makes a prediction, its effect is not just compared to the very next demonstrator segment, it can also be compared further into the demonstration (more detail is given in Section 3.4). Generally, the demonstrator segments are compared further and further away from the segment that the predictions started from (the first L in Figure 3.14), as long as at least one of the predictions matched the previous segment. This means that a behaviour can span more demon-

stration segments than the number of primitives in that behaviour's primitive list. For example, a *forward, forward* behaviour might generate a prediction that would match the last segment in the sequence of demonstration segments: *stop, backward, forward, forward, forward*, since they would both have roughly the same effect on the imitator's actions. Comparing predictions of behaviours this way is essential for the imitator to approximate the actions of the demonstrators by reasoning about the demonstrations and attempting to use behaviours that achieve the demonstrator's actions more quickly.

To actually create a prediction of how its actions will affect the imitator, a behaviour uses the previously calculated primitive statistics (Table 3.1) to approximate how each of the behaviour's primitives in its primitive list will change the position and orientation of the robot. The mean and standard deviation are used to generate normally distributed random values for both the distance each primitive travels, and the degree each primitive turns the robot. When generating normally distributed random values using the standard deviation of the primitives' motion data, the behaviours often made predictions where the position of the robot changed too far to be reasonable. I believe this is due to the wheels sticking or slipping during primitive data collection, throwing off the standard deviation. For example, in Table 3.1 it can be seen that the *backward* primitive has a very large standard deviation in the amount that it affects the imitator's orientation (12.6 degrees), which could be the result of one wheel constantly sticking slightly, and causing a slight turns during the collection of the imitator's primitive data. In practice, using 1/3 of the standard deviation produced better results, as this kept the predictions within one standard

deviation of the mean, keeping the predicted robot positions closer to the average primitive motions.

A behaviour starts making predictions with its primitives by taking the position of the robot from the current demonstrator segment, and predicting the state change that the behaviour's first primitive in its primitive list would cause if the robot executed that primitive. These state changes are calculated using the randomly generated values described above. These predicted state changes can then be compared to the actual state change in the demonstration (this process is detailed in the next section). Behaviours need to incorporate the predicted state change of all their primitives into the behaviour's overall prediction. Once the first primitive in a behaviour's primitive list predicts its state change, the behaviour's second primitive treats the predicted state of the first primitive as the robot's current position. This continues down the primitive list, with each primitive starting its prediction from the predicted state of the primitive that preceded it. The end result is an overall prediction for a behaviour, which is generated by chaining all of the predictions from that behaviour's primitive list together. Through this method a behaviour can make a prediction of the position the robot will be in if it were to execute each of the primitives contained in the behaviour's primitive list sequentially.

Behaviours can also make additional predictions about the position of the ball. To gather data about a behaviour's effect on the ball, a behaviour saves the distance the ball travelled between the start and end frames of that behaviour, as well as the angle of the ball relative to the robot at the end of the behaviour. These ball statistics are only gathered when a behaviour is chosen as the best match to a demonstration

segment. I use these statistics to calculate rough predictions of how a behaviour will affect the ball's position relative to the robot. Behaviours only predict the ball position if they have gathered a minimum number of samples of ball movement data, and if the standard deviation of the angle of the ball relative to the robot is smaller than a threshold. The standard deviation of the ball's angle is compared to the threshold to ensure that the movement of the ball is reproducible that is, it is always generally the same motion, and does not deviate drastically.

The behaviours I designed also gather statistics relating to the state of the robot and ball on the field throughout the imitator's learning process. These statistics are used to develop basic preconditions that can be used to determine when a particular behaviour is applicable or not. These preconditions are only used in my system when controlling the imitator with behaviours, not during the training process.

When behaviours are used to control the imitator's actions which behaviour to execute must be determined. My behaviours use roughly calculated preconditions to determine if they are applicable to the current state of the field. To determine possible preconditions for the behaviours in my work, each behaviour saves statistics about the state of the field when that behaviour was chosen as the best prediction. These statistics represent the collective states of the field when the behaviour was selected as the best course of action (because its prediction was the best of all the other behaviours), essentially all of the states the behaviour was predicted to start from. The statistics that behaviours save about their starting states are the locations of the robot relative to the center of each of the goal posts, as well as their angle relative to the robot. The ball's location and angle relative to the robot were also saved for each

starting state of the behaviours. Each behaviour calculates the mean and standard deviation for each of these statistics, to be used as rough precondition calculations. When determining which behaviour should be used to control the imitator, the current state of the field is compared to see if all of the statistics fall within one standard deviation of their respective means (i.e. is this state statistically similar to the others encountered when this behaviour was predicted?). If this is the case, that behaviour is considered to be applicable to the current state, otherwise it is not. These statistics were only used to determine which behaviours to use when controlling a robot using a forward model, discussed in the next section.

The behaviours are contained in and managed by *forward models* (discussed in Section 2.4). Having explained the components that make up the behaviours in my implementation, I will now discuss how the behaviours are learned and managed by the forward models in the next section.

3.4 Training Forward Models

Most imitation learning systems are designed to generalize over multiple demonstrations. This is necessary, because a single demonstration does not provide enough information to learn all the nuances involved in optimally performing a task [Nicolescu and Matarić, 2003]. What most research has omitted is the need for imitating from many different demonstrators and different kinds of demonstrators. Expecting to have highly-skilled demonstrators readily available for any given task to be learned is unreasonable. Children certainly do not need direct tutoring from professional athletes in order to learn how to play a sport, for example. What imitation learning

approaches need is a way to learn from many different demonstrators, and allow them to influence the imitator's learning according to their relative skill levels.

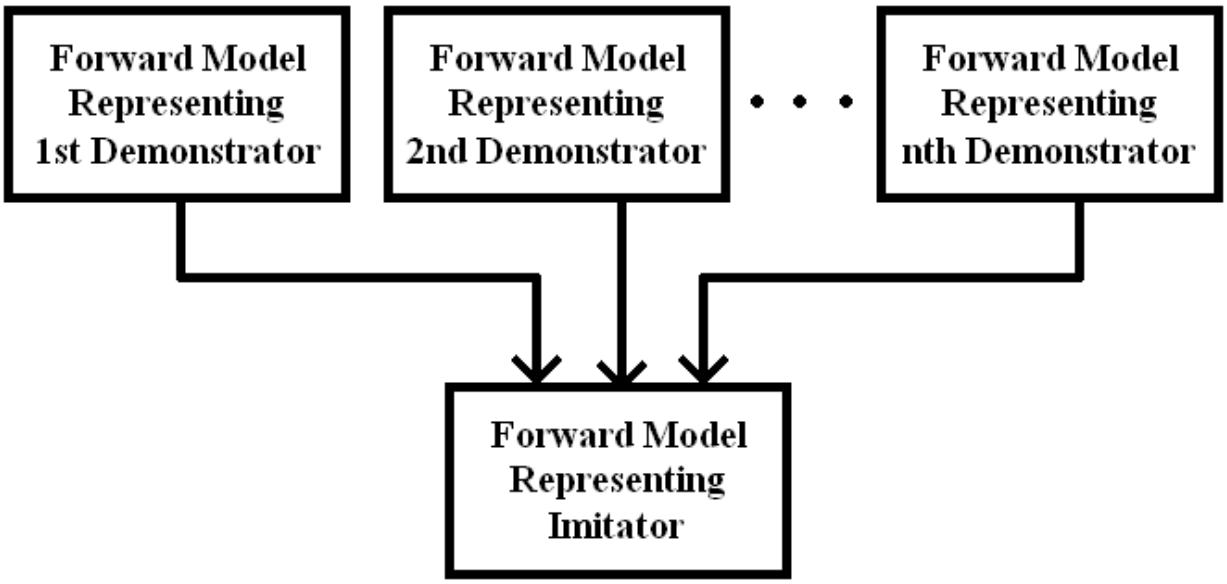

Figure 3.15: The various forward models used to represent all the different demonstrators to which the imitator is exposed, feeding behaviours to the final model being built to represent the imitator itself.

My approach is designed to imitate multiple heterogeneous demonstrators that vary in skill levels. In my approach, each demonstrator that the imitator is exposed to is assigned a unique forward model as seen in Figure 3.15. Forward models are designed to take in the current state of the environment, and make predictions about future states [Demiris and Hayes, 2002]. In my implementation, forward models are used to manage and create behaviours from the imitator's primitives and existing behaviours, as seen in Figure 3.16. There is also a distinct forward model that represents the behaviours that the imitator itself has learned and acquired. The forward model representing the imitator is the final product of the entire imitation learning

process in my approach, and once learning is complete, that forward model can be used to control an imitating robot to achieve the same tasks that it learned from the demonstrators. The forward model representing the imitator can learn its own behaviours like the demonstrator models, but it is also given frequently-used demonstrator behaviours to aid in its learning process. These additional demonstrator behaviours give the imitator a general model of all the useful activity obtained from the demonstrators. A demonstrator forward model learns the various behaviours exhibited by a specific demonstrator, and can be used to predict what that demonstrator might do in any given situation. I use separate forward models for each demonstrator so that the relative skill levels of demonstrators can be modelled and compared.

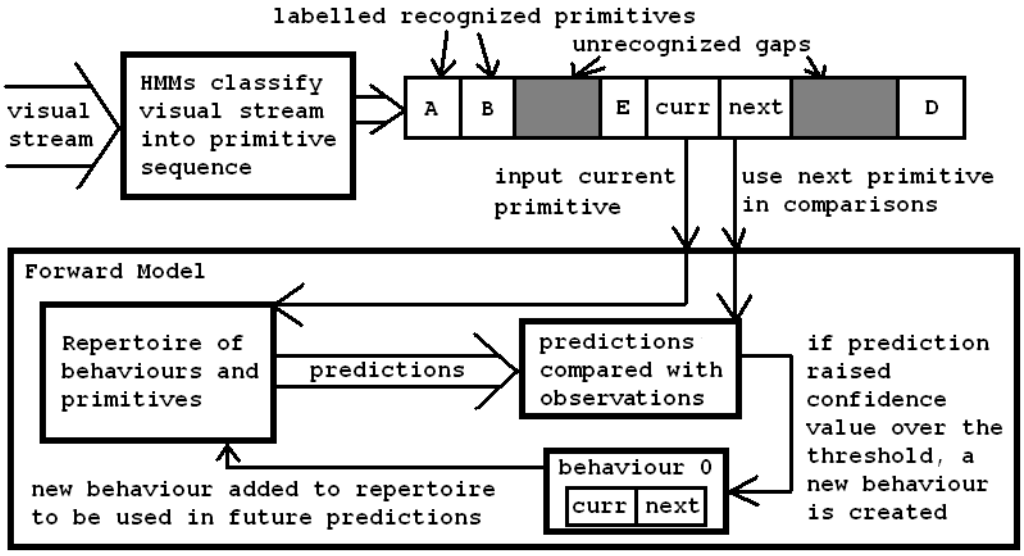

Figure 3.16: Imitation Learning Architecture

To model the relative skill levels of the demonstrators in my system, each of the demonstrator forward models maintain a demonstrator specific learning rate, which I

call the *learning preference* (LP). The learning preference is analogous to how people favour certain teachers, and tend to learn more from these preferred teachers. A higher LP indicates that a demonstrator is more skilled than its peers, so behaviours should be learned from it at a faster rate than a demonstrator with a lower LP. The LP is used as a weight when updating the frequency of two behaviours or primitives occurring in sequence (as discussed in Section 3.3). The LP of a demonstrator begins at the half way point between the minimum (0) and maximum (1) values (the starting LP value for a new demonstrator is 0.5). When updating the frequencies of sequentially occurring behaviours, a minimum increase in frequency (referred to as *minFrequency* in equation 3.11) is preserved (a value of 0.05, obtained during experimentation), to ensure that a forward model for a demonstrator that has an LP of 0 does not stagnate. The forward model for a given demonstrator would still update frequencies, albeit more slowly than if its LP was above 0. Equation 3.12 shows the decay step, where the *decay rate* is equal to $1 - LP$ and the *decayStep* is a constant (0.007).

$$originalFrequency = originalFrequency + minFrequency + minFrequency \times LP \tag{3.11}$$

$$originalPermanency = originalPermanency - decayRate \times decayStep \tag{3.12}$$

Demonstrator Learning Preference Changes	
LP Increased	**LP Decreased**
Goal scored	Goal scored on own goal
Robot moves ball closer to goal	Robot moves ball further from goal
Robot moves closer to ball	Robot moves further from ball
A behaviour is made permanent to the model representing the imitator	Behaviour deleted from the model representing the imitator
A behaviour makes an accurate ball prediction	

Table 3.3: The various actions that cause the learning preference of a demonstrator to be increased or decreased.

$$originalLearningPreference = originalLearningPreference \pm lpShapeAmount \tag{3.13}$$

Throughout the learning process the LP of a demonstrator is altered for varying reasons, which are summarized in Table 3.3. The LP of a demonstrator is increased if one of its behaviours results in the demonstrator (ordered from highest LP increase to lowest): scoring a goal, moving the ball closer to the goal, or moving closer to the ball. The LP of a demonstrator is decreased if the opposite of these criteria results from one of the demonstrator's behaviours (the other factors seen in Table 3.3 will be discussed later in this section). Equation 3.13 shows the update step, where *lpShapeAmount* is either a constant (0.001) if the LP is adjusted by the non-criteria factors, or plus or minus 0.01 for a behaviour that results in scoring a correct/incorrect goal, 0.005 for moving the ball closer to the goal, or 0.002 for moving the robot closer to the ball. In my system I did not factor in the distance of the robot to the ball or the ball to the goal if the result was a goal being scored, because a behaviour resulting in a goal

being scored (even a wrong goal) is so important that everything else is comparatively meaningless. These criteria are obviously domain-specific, and are used to shape the learning (a technique that has been shown to be effective in other domains [Matarić, 1997]) in my system to speed up the imitator's learning. Though this may seem like pure reinforcement learning, these criteria do not directly influence which behaviours are saved, and which behaviours are deleted. The criteria merely influence the LP of a demonstrator, affecting how much the imitator will learn from that particular demonstrator.

The LP of a forward model for a given demonstrator influences the rate that the permanency of the behaviours learned by the forward model for that demonstrator are increased. A forward model for a demonstrator with a higher LP has more behaviours become permanent than a forward model representing a demonstrator with a lower LP. If a behaviour in a model representing a given demonstrator occurs frequently enough to become permanent, a copy of that behaviour is added to the model representing the imitator. A behaviour added to the model representing the imitator this way has its permanency reset so that the behaviour has to earn permanency in the model representing the imitator. Just because a behaviour becomes permanent to a model representing a given demonstrator does not mean that it will be useful enough to become permanent to the model representing the imitator: this should be intuitive in a heterogeneous environment, as specific behaviours that benefit one physiology may not be useful at all for others. Essentially a forward model for a given demonstrator nominates *candidate behaviours*, which are any of the behaviours that become permanent to the forward model representing that demonstrator. These can-

didate behaviours are behaviours that the forward model representing the imitator might be able to use when building a repertoire of useful behaviours for the imitator. If a forward model for a given demonstrator has made a behaviour permanent, that permanency is an indication that the behaviour is useful (or at least it is useful to model some of the actions of that particular demonstrator). These useful candidate behaviours are nominated for consideration as final behaviours for the imitator to use. If a candidate behaviour is added that already exists in the forward model representing the imitator (possibly created by another forward model representing a demonstrator, or the forward model representing the imitator), that behaviour has its permanency increased in the model representing the imitator. This is done because if a behaviour is useful enough to be added to the forward model to the imitator once, then it must be very useful if other forward models nominate the same behaviour, and so its permanency should be increased. If a model representing a given demonstrator has one of its behaviours added to the model representing the imitator, and then later that behaviour becomes permanent *to the model representing the imitator*, the LP for that model representing a given demonstrator is increased, since one of its suggested behaviours is useful enough for the imitator to keep. If the forward model representing the imitator deletes a behaviour that was added by one of the forward models for a given demonstrator, that forward model for the demonstrator has its LP decreased, since its suggested behaviour was not useful from the perspective of the forward model of the imitator.

Each forward model also maintains a decay rate, which is the inverse of the LP for a forward model for a given demonstrator. Poorly skilled demonstrators will have

the behaviours in the forward models that represent the demonstrators decay faster, leading to their deletion. Forward models apply the decay to all their behaviours at each prediction step. This means that whenever a new behaviour is chosen to match the current segment of the demonstration, all behaviours decay. The chosen behaviour also decays, though it will have its permanency increased enough to overcome the decay at this particular prediction step. Behaviours therefore need to be predicted fairly often and accurately to remain in a forward model without being deleted. Useful behaviours will acquire enough permanency to stay ahead of the decay rate, and eventually become permanent to the forward model.

All forward models begin with the imitator's primitives as the base behaviours to create more abstract behaviours. To make predictions about a demonstration, a forward model makes predictions about how each of its behaviours and primitives will change the state (the robot's orientation, as well as the robot's and ball's positions) of the current segment. A demonstration segment labelled as a gap represents a segment that no primitive was able to classify (as described in Section 3.2). When a gap is encountered, no predictions are generated by the primitives, since a gap indicates that no primitives can adequately match the segment's state change. Only predictions from behaviours are used to approximate the state change of segments classified as gaps. When a forward model starts to process a demonstration, the first primitive segment is chosen as the current segment, as well as the current behaviour. The current behaviour is the behaviour whose row in the frequency matrix will be updated once the behaviour that follows it is determined through the predictive process. A forward model compares the predictions to the actual next segment. At first the

next segment considered is the one immediately adjacent to the current segment. If the demonstration has just begun to be processed, the current segment is the first primitive, and the next segment is the second primitive (in Figure 3.14, the first primitive is the first L, and the second primitive is the second L).

The imitator should learn how to approximate the result of the demonstrations as efficiently as possible. I have designed the predictive process to ensure that all forward models attempt to span as many segments of the demonstration that they can with their available behaviours. As the predictions are made, some behaviours can make accurate predictions about segments that are further along the demonstration than others. For example, if the demonstration contains a sequence of repeating *forward* primitives (*forward, forward, forward, forward, forward, ...*), and a forward model has a behaviour that is composed of three forward primitives in sequence (*forward, forward, forward*), that behaviour can make a prediction that matches the third segment. In my system, as long as the next segment has at least one prediction that matches, the position of the next segment is incremented and all the predictions are compared to this segment. In Figure 3.17 at stage a) all behaviours have generated predictions about how they will change the state of the first primitive (F). All of these predictions are compared to the next segment, in this case a *stop* primitive (the S). Since at least one of the predictions is considered a match in stage a), all predictions are compared to the segment following, the second F in this example.

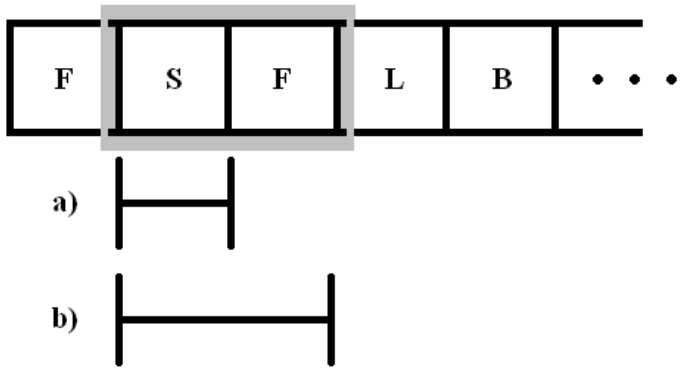

Figure 3.17: Predictions are compared incrementally down the demonstration's primitive segments. Stage a) represents a match up to one segment past the current behaviour, while stage b) represents a match (the gray box) up to two segments past the current behaviour.

In Figure 3.17 at stage b), at least one prediction was a match, and so the segment being compared moves down another primitive. This process continues as long as at least one of the behaviour's predictions matches the next segment. In the example given in Figure 3.17, no predictions match segments past those matched at stage b), and so stage b) is as far as the current round of predictions can reach in the demonstration. The primitives covered by the predictions in b) are shown outlined in gray. Once no more predictions can match the next segment, the maximum span that the forward model's existing behaviours can approximate has been reached. All behaviours that had a prediction that matched that segment are then evaluated and one is selected as the next behaviour. The furthest segment that was matched becomes the new current segment, and in the next round of predictions, the behaviours will predict their effects on this new current segment. The selected behaviour has the frequency with which it follows the current behaviour increased. The selected behaviour becomes the current behaviour, and in the next round of predictions it

will update the frequency of whatever behaviour follows it. If no predictions match even the very first segment compared, then the current behaviour is set to whatever primitive is in the next segment, and the prediction process starts over as if the next segment was the beginning of a demonstration.

The accuracy of a prediction is determined by how well the predicted position and rotation of the robot matches with the actual outcome (the next segment of the demonstration). All predictions are evaluated to ensure that their rotational accuracy as well as their positional accuracy are within a minimum range. Any prediction that does not meet this minimal accuracy criteria is removed and considered a failed prediction. In my experiments I found that a minimum positional accuracy of 40 millimeters and a minimum rotational accuracy of 45 degrees works best. The reasoning behind making the range of acceptable rotation accuracies fairly large relative to the range of positional accuracies is due to the fact that in a two-wheeled robot, an accurate position is harder to achieve than an accurate rotation. Since the robot can rotate on the spot, it is easier for it to turn to adjust an incorrect orientation than it would be for it to drive to the correct position.

Once all predictions have been pruned down to the ones that meet minimal positional and rotational accuracies, the remaining predictions still need to be compared so that the behaviour that generated the best prediction can be chosen. One major deciding factor is if a behaviour predicted the ball's movement or not. The ball's movement is predicted by a normally distributed random value for distance, and the mean of the previously observed ball angles relative to the robot. Figure 3.18 shows a screen shot of a prediction generated by a behaviour that predicts the ball's posi-

tion. The ball is represented as a circle, while the robot is a square, with a line to indicate its orientation. The actual positions of the ball and robot are solid, while the predictions of the ball and robot are represented as semi-transparent.

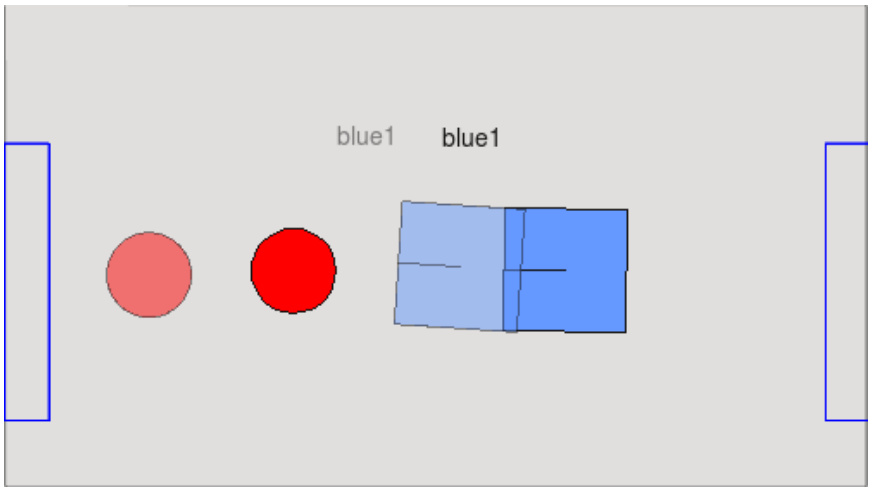

Figure 3.18: A behaviour that predicts the ball's movement. The square represents the robot, the circle is the ball. The predicted positions of both the ball and the robot are faded.

In my experiments it was rare for a behaviour to make predictions about the ball, and even more rare for those predictions to be accurate. Due to the rarity of accurate ball predictions, any behaviour that accurately predicts the ball has its frequency of occurring in sequence with the current behaviour increased, even if it is not chosen as the final behaviour for this round of predictions. This is done to weight the learning towards acquiring behaviours that can predict their effect on the ball. If the imitator can learn more behaviours that make accurate ball predictions, it can choose its actions much more wisely. The LP of the demonstrator is also increased when an accurate ball prediction is made. Only the accuracy of the angle of the ball relative to the robot is evaluated for ball predictions, with a minimal accuracy

of 22.5 degrees (which was determined during preliminary experimentation). If any behaviours make accurate predictions about the angle that the robot moves the ball relative to itself, these behaviours move on to the final phase of prediction selection, and all other predictions are pruned. If no behaviours make accurate ball predictions, none are pruned and the predictions remaining (the predictions that meet minimal positional and rotational accuracies) go on to the last phase of prediction selection.

At the last phase of prediction selection, only the most accurate predictions remain. At this point all predictions satisfy the minimum requirements of an accurate prediction, meaning that their behaviours will approximate the state change up to the next demonstration segment. My implementation is designed to learn the shortest behaviours available to achieve desired environmental changes. To achieve this, the behaviour with the smallest number of primitives is chosen from the remaining accurately predicted behaviours. In the case of two or more behaviours tying for the fewest primitives, the behaviour that has been in the forward model the longest (the oldest behaviour) is taken as the best match. This is done to prevent newer behaviours from being preserved if an existing behaviour achieves the same effects.

To train the individual forward models in my system, I begin by training the forward models representing the demonstrators separately, using the demonstration data collected specifically from each demonstrator respectively. These demonstrations are given to their respective models in random order. The demonstrator forward models start with a repertoire of only the imitator's primitives, and an LP of 0.5. As the demonstrator forward models are being trained, behaviours that become permanent to these models are constantly being forwarded to the forward model representing

the imitator. The imitator therefore starts its training with its primitives (just as the demonstrator forward models started with the imitator's primitives), but the imitator is also primed with the added demonstrator behaviours.

After this phase of training is complete, the imitator has specific forward models for each unique demonstrator. When training, the imitator groups all of the demonstrations from all of its demonstrators together. Each demonstration is selected randomly, and is first passed to each of the demonstrator forward models to see if any new candidate behaviours based on the current demonstration are nominated by any of the forward models for the demonstrators.

The forward model representing the imitator makes predictions and creates behaviours in exactly the same way as forward models that represent the different demonstrators. The forward model representing the imitator processes a demonstration after all of the forward models representing the demonstrators have had time to add any last minute candidate behaviours to the model representing the imitator. The forward model representing the imitator never changes its LP, which stays at a value of 0.9. The training process is complete when all demonstrations have been processed by the forward model representing the imitator. The last step is to remove any non-permanent behaviours from the model representing the imitator so that it can be used to control the imitator in the future.

After the forward model representing the imitator has been trained, the learning process is complete. The primitives and permanent behaviours remaining in the forward model representing the imitator should be able to guide the robot through the task it is to imitate. The forward model representing the imitator uses the precon-

ditions for each behaviour (these statistics are gathered during the training process) and compares them to the current state. When determining if a behaviour is applicable to the current state of the environment, the distances and angles of the two goal centers and the ball are compared to their respective means of all the precondition statistics collected for that behaviour so far. If the distance from all of these statistics from their respective means falls within one standard deviation, then the behaviour is considered applicable to the current state (position of the robot and ball). Only behaviours whose preconditions match the current state are selected, though if no behaviours have matching preconditions, then all behaviours are evaluated to see which one should be chosen to be executed. The same criteria that are used to shape the LP of a demonstrator are used to compare the predictions of all the behaviours in the forward model. The behaviour whose prediction has the best rating relative to these criteria (e.g. does the behaviour predict that its execution will result in the imitator scoring a goal?) is chosen to be executed. In practice it was necessary to stop the execution of a behaviour if the robot was about to drive off the field. This is due to the fact that my implementation did not focus on behaviours that kept the robot on the field, only those that involved scoring a goal with the ball. I kept the criteria limited to ball manipulation in order to keep the scope of the behaviours learned reasonable, and with additional work in capturing behaviours for staying on the field, this framework could just as easily cover this situation as well.

3.5 Summary

In this chapter I have described my approach and provided the implementation details required to reproduce my work. I have detailed the process of gathering visual data of the imitator's primitives as well as demonstrations to for the imitator to learn from. I have elaborated on the techniques that I use to process the visual data into discrete symbols to train Hidden Markov Models. I discussed implementation details of HMMs, as well as specific design decisions I use in my system to implement them. I discussed converting a demonstration into a sequence of primitive symbols so that behaviours can be created to reason about the demonstration. I detailed the creation and maintenance of behaviours in my system, including their permanency attribute and how they take additional information about the state of the environment, namely the ball's motion, into consideration. Finally I have given implementation details on the forward models I have designed for my imitation learning system, including how the forward models use behaviour's predictions to match behaviours to demonstration segments, and how this process increases frequencies of behaviours occurring in sequence. I discussed how new behaviours are incorporated into the forward models, and how demonstrator specific forward models benefit the model representing the imitator by suggesting their permanent behaviours to the imitator. I then explained how the imitator processes all of the demonstrations and ultimately ends up with a final set of permanent behaviours. Finally I briefly discussed the way that I use my imitator forward models to control the imitator after all learning has been completed. In Chapter 4 I will discuss the results of my experiments and provide a detailed evaluation of my work.

Chapter 4

Evaluation

In this chapter I will discuss the setting in which I evaluate my implementation, and the results of that evaluation. I will first elaborate on the various robots used as demonstrators in my work, as well as the specifics of setting up the demonstrations. The first results I will discuss are the primitive classification results given by the HMMs in my work (Section 3.1.4). I will then show the results of using the HMMs in the process of converting visual data to sequences of primitives, as detailed in Section 3.2.2. In Section 4.4 I will present the results for my complete imitation learning architecture. This will involve all the components evaluated in the preceding sections. I will break this overall system evaluation into two separate evaluations. First I will present results of my system being used to train an imitator from demonstrators of varying physiologies in Section 4.4.1. Following that, I will present results of my system training an imitator with identical physiologies, but varying skill levels in Section 4.4.2. The last section of this chapter will provide a summary of the evaluation of my work.

4.1 Experimental Setting

To evaluate this approach in a heterogeneous setting, I employed the Lego Mindstorms, Citizen, and Bioloid robots previously shown in Figure 1.1 to gather demonstrations. These are shown here in Figures 4.1, 4.2, and 4.3, respectively, to show a more detailed view of these robots. The control programs for the robots shown in Figures 4.1 and 4.2 run on an external computer that transmits robot commands via infrared (IR), while the Bioloid shown in Figure 4.3 runs its control program on the cellular telephone mounted on top of the robot. The telephone transmits robot commands via Bluetooth. More detail on gathering demonstration visual data can be found in Section 3.2.1. Each of the robots used in my experiments is controlled using its own behaviour-based control system, which was the same system used when the particular robot was previously employed in RoboCup competition (that is, the demonstrator code was not specifically developed for this work). Alterations did have to be made with regard to the expectation of field size, since my implementation uses small fields to ensure multiple penalty kick attempts can be recorded in reasonable time. The field size was adjusted more significantly for the Citizen robot because it could not complete demonstrations on a large-size field given its low available battery charge and extremely small size. To put this into perspective, because the Citizen is a tenth the size of the imitator, having the Citizen demonstration on the same field size would be the scale-equivalent of a human demonstrating goal kicks across a 1000-yard field. The Bioloid and Lego Mindstorms robots were demonstrated on a 102 x 81 cm field, while the Citizen was demonstrated on a 56 x 34.5 cm field. The ball used by the Bioloid and Lego Mindstorms robots were made of foam and was 10 centimeters

in diameter, while the ball used by the Citizen robot was made of styrofoam (even a ping pong ball was too heavy for the Citizen to move), and was 2.5 centimeters in diameter. The body of the Lego Mindstorms demonstrator was changed slightly from its original design (the bumpers used to move the ball were reduced in size) which was also a two-wheeled robot design. All applications (screenshots can be seen in Appendix A) were developed using the *Qt* platform in *Ubuntu* Linux.

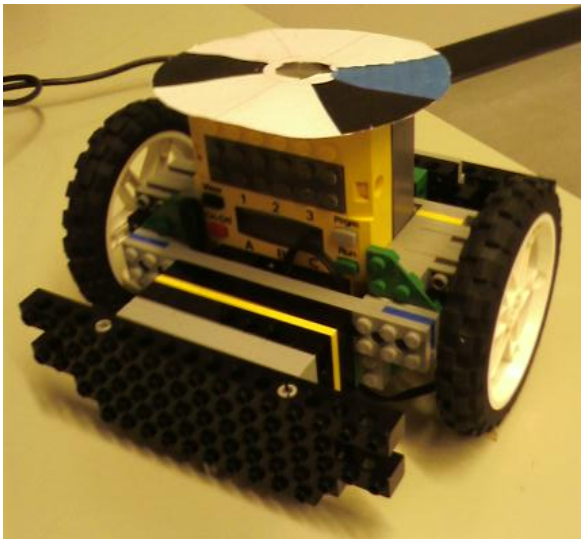

Figure 4.1: The Lego Mindstorms robot used as the imitator and some of the demonstrators in my work.

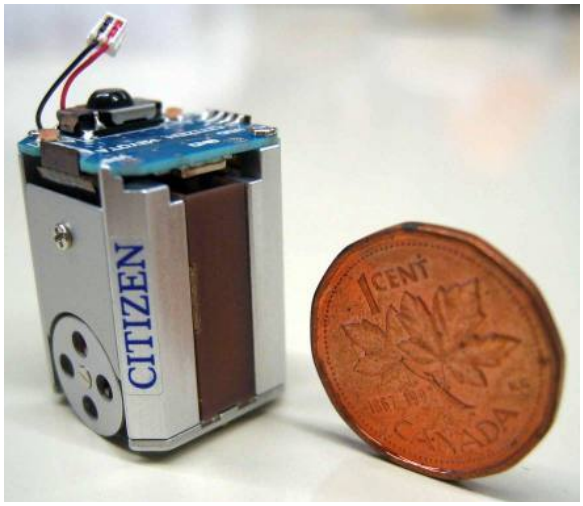

Figure 4.2: The Citizen EcoBe (version 1) robot used as a demonstrator in my work.

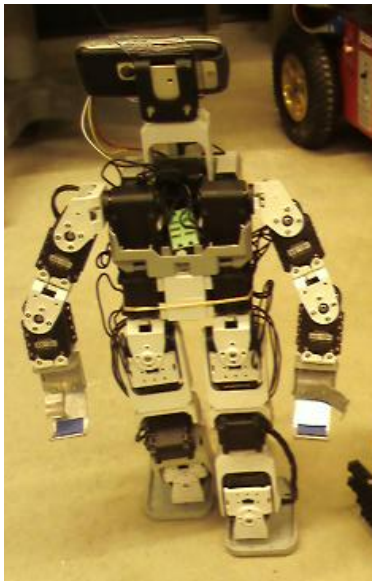

Figure 4.3: The Bioloid robot used as a demonstrator in my work.

There are a total of four control programs used in my work to control the Lego Mindstorms robot. The name I use for the Lego Mindstorms demonstrator controlled

using the RoboCup code is *RC2004*. In Section 4.4.2, I evaluate my system's ability to learn from demonstrators that vary in skill levels. All demonstrators in those experiments control the same physical Lego Mindstorms robot. I have named them according to their skill levels for this experiment. I call the expert demonstrator ExpertDemonstrator which is just an alias for the RC2004 demonstrator when used in the varying skill experiments. The demonstrator of average skill is called AverageDemonstrator, which is controlled by a randomly selected forward model trained during the experiments described in Section 4.4.1. The poorly skilled demonstrator is called PoorDemonstrator and it is programmed simply to drive at the ball. The fourth robot that uses the Lego Mindstorms is not used as a demonstrator. This robot was trained using my implementation to evaluate the ability of my system to learn from demonstrators of varying skill, and so I call it VaryingSkillTrained. VaryingSkillTrained learned from PoorDemonstrator, AverageDemonstrator, and ExpertDemonstrator in the experiments described in Section 4.4.2. After VaryingSkillTrained's training process was complete, I gathered visual data of its actions just as I would for a demonstrator, so that its resulting skill level could be directly compared to the skill level of the demonstrators it learned from.

In terms of the placement of the robot and ball on the demonstration fields, some consistency was required for proper comparison of results. Rather than placing the ball and robot anywhere, I limited the positions to the two field configurations shown in Figure 4.4. In the configuration on the left, the demonstrator is positioned for a direct approach to the ball. As a more challenging scenario, I also used a more degenerate configuration (on the left), where the demonstrator is positioned for a

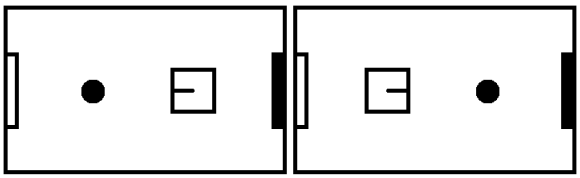

Figure 4.4: Field configurations. The demonstrator is represented by a square with a line that indicates the robot's orientation. The target goal is indicated by a black rectangle, the demonstrator's own goal is white.

direct approach to the ball, but the ball is lined up to its own goal – risking putting the ball in ones own net while manoeuvering, and requiring a greater field distance to traverse with the ball. The goals are half the height of the field, and centered at each end. The placement of the robot and ball are in the center of the height of the field, and 1/3 or 2/3 of the length of the field depending on which configuration is being used. While these do not cover every possible place a goal kick might occur from by any means, they do cover both the typical scenario that would be used in a penalty kick, and a significantly more challenging configuration as well. Moreover, the second configuration is a very different set of operations from the first, and the learner must be able to successfully differentiate these in testing for good performance.

The individual demonstrators were recorded by the Ergo global vision system [Baltes and Anderson, 2007] while they performed 25 goal kicks for each of the two field configurations. The global vision system continually captures the x and y motion and orientation of the demonstrating robot and the ball. The demonstrations were filtered manually for simple vision problems such as when the vision server was unable to track the robot, or when the robot broke down (falls/loses power). The individual demonstrations were considered complete when the ball or robot left the

field. Since all demonstrator types used IR control for hardware, it was also possible to record the command streams for reference, to use when evaluating the conversion of demonstrations to sequences primitive symbols (Section 3.2).

One learning trial consists of each forward model representing a given demonstrator training on the full set of kick demonstrations for that particular demonstrator, presented in random order (see Section 3.4 for more detail on the forward model training process). Once the forward models representing each demonstrator are trained, the forward model representing the imitator begins training. At this point all the forward models for the demonstrators have been trained for their own data, and have provided the forward model representing the imitator with candidate behaviours. The forward model for the imitator then processes all the demonstrations for each of the two field configurations (a total of 150 attempted goal kicks) in random order. All of the forward models for each demonstrator predict and update their models at this time, one step ahead of the forward model for the imitator. This is done to allow each forward model a chance to nominate additional candidate behaviours relevant to the current demonstration instance, to the forward model for the imitator.

Behaviour Metrics Gathered for All Individual Forward Models
Number of created behaviours
Number of deleted behaviours
Number of permanent behaviours
Number of candidate behaviours created
Number of candidate behaviours made permanent in forward model representing the imitator
Number of duplicate candidate behaviours

Table 4.1: Individual Forward Model Behaviour Metrics

During the forward model training process, I gather a number of different metrics. Table 4.1 lists the various metrics dealing with behaviours that were gathered for each of the forward models representing the individual demonstrators. Table 4.2 lists the metrics gathered for each individual forward model that are involved with the prediction process. The way that the LP changes over time is affected by the accuracy of a model's predictions, though the LP can also change in other ways (see Section 3.4).

Prediction Metrics Gathered for All Individual Forward Models
LP change over time
Number of prediction rounds
Number of times prediction match found

Table 4.2: Individual Forward Model Prediction Metrics

4.2 Using Hidden Markov Models for Classifying Primitive Sequences

To evaluate the HMM training process, I gathered vision data of the imitator executing random primitives, while recording the stream of commands the imitator was sending, and recorded which vision data belonged to which primitive types. The positional (for the *forward* and *backward* primitive data) and rotational (for *left* and *right*) vision data techniques explained in Section 3.1.2 were used to obtain discrete symbols to train the HMMs. I used four HMMs in my work, one to recognize each of the *forward*, *backward*, *left*, and *right* primitives. I averaged the probability that

each HMM matched a primitive sequence over each of the primitive test data sets (details about how this data was obtained is given in Section 3.1.1). The averages of these probabilities are shown in Figure 4.5, and are separated by HMM type. Each HMM also indicated which vision data processing method was used to train it. All the probabilities are given as the log of the actual probability as described in Section 3.1.4, as the actual probabilities are extremely small values (some are less than 10^{-300}).

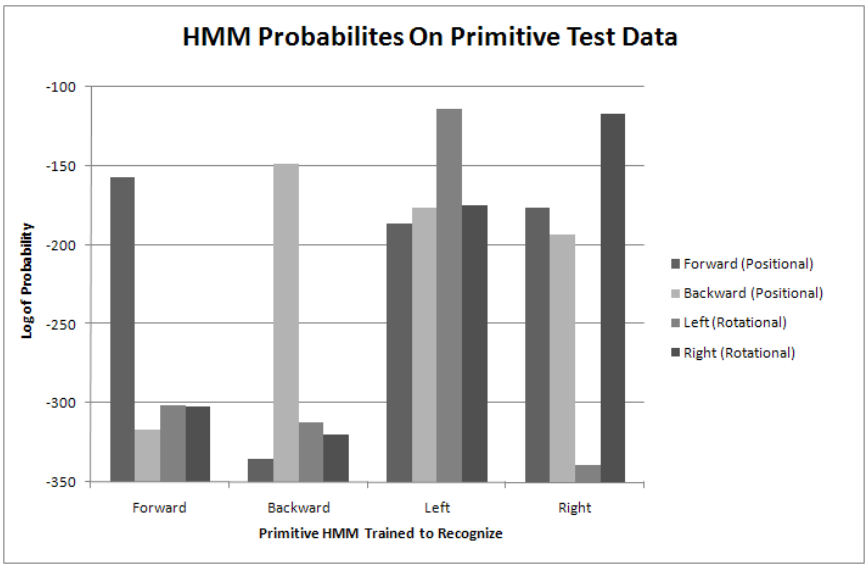

Figure 4.5: The average probabilities for each HMM type on the primitive test data.

It can be seen in Figure 4.5 that each HMM has a higher probability on average for the primitive it is trained to recognize than the other HMMs. The HMMs trained to recognize the *forward* and *backward* primitives have substantial distances between their probabilities and the other HMMs for their primitive types. The HMMs trained to recognize the *left* and *right* primitives are not as distinct, but they are still the highest probabilities over the remaining HMMs for their primitive types. The minimum difference is 60, between the probabilities for the *left* primitive. This minimum

distance is what is the basis for classifying gaps (when two or more HMMs have probabilities too close to choose one to classify a demonstration segment) in the conversion process described in Section 3.2.2. During the conversion process I found that a much smaller minimum distance between HMMs was best to use to determine a gap in demonstration classification, and the final minimum distance used is 2. This led to the classification results discussed in the next section.

HMM Type	**Accuracy**
Forward	98.00%
Backward	98.69%
Left	96.84%
Right	91.24%

Table 4.3: Individual HMM Accuracy.

When gathering the primitive visual data used to train the HMMs, I also collected primitive visual data used as testing data (Section 3.1.1) to evaluate the use of the HMMs to classify primitive segments. This primitive visual testing data was gathered in exactly the same manner as the HMM primitive visual training data, but the test data was never used by any HMM during the training process. To measured each HMM's individual accuracy, I calculated the probability that each HMM classified each individual primitive visual sequence in the test data. For a given sequence, the HMM that gave the highest probability was selected as a match, and the primitive that the HMM was trained to recognized was used to classify that primitive sequence. The actual primitive that was used to create that sequence was then compared. A separate count for the number of correctly classified sequences was maintained for each of the individual primitives. These counts were used to determine the percentage of

primitive test sequences that were classified accurately. These results are shown in Table 4.3. The results for the HMMs classifying the imitator's own movements are very good, especially for the *forward* and *backward* primitives.

Each individual primitive sequence used for training and test data contained only the visual effects of a single primitive being executed (Section 3.1.1). Each primitive training sequence needs to be separate from the other training sequences so a HMM can be trained to recognize each primitive when it is observed on its own. Keeping individual primitive training sequences separate prevents the HMM from getting confused by data from other primitives (especially those that the HMM is not being trained to recognize). The test data was also made up of individual primitives in each sequence, to determine how well the HMMs classify data similar to that which they were trained to recognize.

Recognizing sequences using the HMMs works very well when each sequence being recognized is guaranteed to contain the visual effects of exactly one primitive. A more difficult problem is classifying primitives from the visual data of others, as the motions of others are not guaranteed to match with any of the primitives known to the imitator. More difficult still is when this visual data is not separated into sequences of only one primitive. All the demonstrations were converted into sequences of primitive symbols using only the visual effects of the demonstrators, meaning that my system needs to both separate each demonstration into smaller pieces, and then classify those smaller pieces as primitives using the HMMs. In the next section I will discuss the evaluation of my system's ability to first segment a demonstration into roughly primitive-sized (in terms of the visual effects of a demonstrator's actions) sequences,

and then classify each of these unknown segments as a primitive.

4.3 Primitive Recognition from Demonstration Visual Data

The segmentation and classification of the demonstration sequences is what enables my framework to recognize the visual effects of the actions of others in terms of the imitator's own primitives. I needed a method of evaluating the accuracy of the conversion of the demonstrations into primitive sequences, but comparing each classified demonstration segment was out of the question. This is because it would take an inordinate amount of time, since some demonstrations are segmented into hundreds of primitive symbols. The demonstrations were captured by the Ergo global vision system as contiguous visual frames, with the x, y, and θ of the robot in each visual frame recorded, as well as the command that the demonstrator was sending at the time the visual frame was recorded (Section 3.2.1). I used the recorded demonstrator commands to evaluate the final result of the process of converting demonstrations to sequences of primitive symbols. The commands stored in each vision frame were not used in any way to classify or segment the demonstrations.

To record control program commands during a demonstration, I take advantage of the fact that all of the demonstrators were controlled by programs residing on external computers (the Bioloid's external computer was the cellular telephone), that transmit their commands to the robots via wireless technology. I created a recorded set of commands as demonstrations were performed by causing the control programs

to not only transmit each command to the robot, but create a separate record of it. The recording program that I wrote accepts these commands and saves them to the current visual frame being recorded. This then serves as the baseline for comparing the recognition of the actions of others by my system.

For a completely accurate evaluation of primitive recognition in my system, these recorded commands need to perfectly match those that were actually executed by the demonstrators, so that each primitive my system recognizes can then be labelled as correct or incorrect. As is common when working with mobile robots, however, there are elements of the real world that complicate this.

The accuracy of recorded commands is reduced by the fact that the control programs transmit each command remotely. Infrared (IR) can be missed or delayed, resulting in differences between the command sent (and recorded) from a control program and the command currently being executed (i.e. generating visual effects) by the robot. There are also delays between the time a command is received and when it is executed, so the visual effects of the previous command can appear in the visual data even though the command being recorded has already changed.

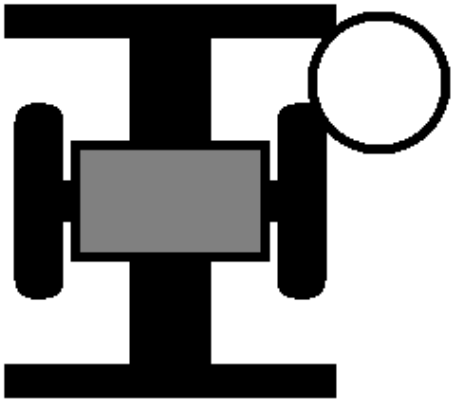

Figure 4.6: A scenario where the ball becomes stuck between the Lego Mindstorms robot's wheel and bumper.

Beyond issues in command processing, there was one further problem that made gathering the actual executed commands a challenge. This was the design of the Lego Mindstorms Robot. An example of this problem is depicted in Figure 4.6: the ball can become wedged between the robot's wheel and its front or back bumpers (the gap that allows this can be seen in Figure 4.1), which causes the robot to remain stationary even if it is sending commands to move itself. It will eventually break free, though it can take a few seconds. The primitive motions are quite small commands, however, and on rare occasions do not move the robot the first few times the command is sent because the motor has a hard time overcoming its inertia. The robot is still properly trying to generate the action, and normally my system would be able to match the visual effects of its action correctly. In this case however, the recorded command will not match the visual outcome, as all commands executed in this scenario will result in visual effects equivalent to the *stop* command, since the robot remains stationary while stuck. When my primitive recognition phase recognizes this as a *stop* command

(which is correct), it will be marked as incorrect because its output does not match the recorded command.

In addition to less-than-perfect accuracy in the recorded demonstrator commands, there is also the problem of comparing the actual commands used by the demonstrators (i.e. the symbols recorded from their control programs) to primitives that are recognized from the visual stream by my approach. The symbols used as commands for some of the demonstrators are essentially components of a different language than the imitator uses, since the people developing these control programs in the past used names for their actions and behaviours that are specific to their architecture. A few robots have the same primitives as the imitator, such as VaryingSkillTrained, AverageDemonstrator, PoorDemonstrator, and the Citizen demonstrator. The Bioloid and RC2004 demonstrators use very different command languages than the imitator. For these two demonstrators, there is rarely a direct comparison between the commands recorded and the primitive symbols used to classify each segment. I hand-compiled a reference list of rough equivalencies for these demonstrator's commands by observing the visual effects of their actions and manually selecting the imitator's primitive that best matched the demonstrator's motion. There are 130 unique commands that do not match the name of one of the imitator's primitives used by all demonstrators combined. The vast majority of these commands are from the RC2004 demonstrator. The RC2004 demonstrator was designed to play soccer on a field approximately ten times the size of the field used in my implementation. The body of the RC2004 demonstrator was also changed slightly from its original design, though it remained two-wheeled. There were many variations on turning commands, some of which were

very subtle in the smaller space.

The issues in the Bioloid were not as much the broader range of commands as they were commands that affected only specific body parts. The command equivalencies of the commands used by the Bioloid demonstrator that do not share names with any imitator primitives, along with my mapping to an appropriate visually equivalent primitive are shown in Table 4.4. When the Bioloid is scanning for the ball, for example, which involves tilting the cell phone to move its integrated camera, this behaviour appears as a *stop* primitive in the visual stream (since the robot remains stationary when viewed from above). When the Bioloid is kicking the ball, it is balancing on one foot and angling its body slightly backward to compensate for the raised leg, so the kick command appears as a backward motion. This motion needs to be categorized as a *backward* primitive in order to properly evaluate the conversion accuracy when a demonstration segment encompasses the visual effects of the Bioloid kicking the ball. If this demonstration segment is classified as a *backward* primitive by my primitive recognizer, this should be considered correct for evaluation purposes, as it is the imitator primitive that best matches the visual effects of the demonstration segment. It must be stressed that the manual selection of command equivalencies was only done to get a measure of the classification accuracy of my system. The hand-compiled reference list was not used in any way to perform actual classification.

Comparing recognized primitives to actual recorded commands started out being an obvious way of evaluating primitive recognition in my approach. However, given the above factors, which were noted as this research progressed, I came to expect this method of evaluation to be pessimistic in nature. There are many more opportunities

Bioloid Command Equivalencies	
Actual Command Recorded	**Primitive Selected as Match**
Wait	Stop
Go	Stop
Scan Forward	Stop
Scan Down	Stop
Lost	Stop
Kick Right	Backward
Kick Left	Backward
Scan Left	Left
Shuffle Left	Left
Scan Right	Right
Shuffle Right	Right

Table 4.4: Some examples of commands recorded from the Bioloid humanoid demonstrator and their selected primitive equivalents.

to have correct or roughly-approximate primitives marked as incorrect under these conditions than one would think at first.

To evaluate the ability of my system for converting demonstrations into sequences of primitive symbols, I compared each segment's classification to the actual underlying commands that were sent by the demonstrator during that time, after converting the latter to equivalent imitator primitives using the manual mappings described above. A particular demonstration segment can have more than one type of demonstrator command among its vision frames, since segments are determined by breaks in the visual effects of a demonstration, not the commands used to create them. This can lead to adjacent segments bleeding some of their first or last commands into their neighbouring segments. For example, if a demonstrator executes a *forward* command, followed by a *right* command, the segment that is created by my system that encompasses the visual effects of the *forward* command may also contain a few

Demonstrator Type	**Commands Matched**	**Gaps (No match found)**
Citizen	31.4%	2.7%
RC2004	35.5%	1.3%
Bioloid	11.6%	2.8%
PoorDemonstrator	50.1%	1.2%
AverageDemonstrator	36.2%	2.6%
VaryingSkillTrained	62.6%	2.0%

Table 4.5: Demonstrator Segment Command Match and Gap Percentages

of the *right* commands at the very end. One cause of this is due to the delay between commands being sent and the time they are executed. To deal with this issue, I calculate how many of the demonstration segment's frames contain each primitive type. If a primitive takes up at least 75% of a demonstration segment (this value was determined during preliminary experimentation), that segment is considered to contain demonstrator commands that create visual effects similar to that primitive, and my system should match this segment to the same primitive for the match to be considered correct. If no primitive takes up 75% of the segment, the segment is not considered classifiable for the purposes of this evaluation. This primitive is then compared to the classification given it by the HMM that gave the highest probability of a match. This comparison provides a rough idea of how well the segmentation and classification of demonstrations is performed by my system. I also recorded the number of segments that were classified as *gaps*: that is, the HMMs probabilities of matching the segment were too close for a clear classification to be chosen. These results are shown in Table 4.5. All percentages in the table were calculated from converting 50 demonstrations for each robot (25 of each of the two field configurations shown in Figure 4.4).

The VaryingSkillTrained robot has the highest overall classification accuracy. I believe this is due to the fact that it has the same physiology as the imitator, and also the same primitive motions, so it is easier for my system to classify its motions as primitives, and avoids issues in attempting to translate actual commands to the primitives used by my system. Also, VaryingSkillTrained was trained on physiologically identical demonstrators, so its learned behaviours are directly related to the motions possible for a two-wheeled robot, making it easier for the HMMs trained on the imitator primitives to classify its motions. The PoorDemonstrator also has a high classification accuracy, and this is likely due to the fact that it is physically identical to the imitator and uses the same basic primitives as the imitator, making it easier to classify than the rest. The AverageDemonstrator is controlled by a forward model from my implementation that was trained on three demonstrators with different physiologies. Even though it also is the same physical robot as the imitator, and shares the same primitive commands, the conversion accuracy is a fair bit lower than the VaryingSkillTrained and PoorDemonstrator robots. This could be due to the fact that the PoorDemonstrator has many repeated movements (many *forward* commands in a row for example), and that AverageDemonstrator learned behaviours from physiologically heterogeneous demonstrators, as opposed to VaryingSkillTrained which learned all of its behaviours from a two-wheeled robot.

The percentage of demonstration segments classified as gaps is very low. This means that the number of times that two or more HMMs had probabilities too close to choose a primitive classification was very low. This indicates that for the most part, one of the four HMMs used in my system had probabilities clearly higher than

the others for the majority of the demonstrator segments. This is a promising result.

The actual classification results overall, however, appear poor in absolute terms at first glance, especially for the humanoid. As already stated, however, there are numerous reasons to expect this to be a pessimistic measure. The command range and types for a humanoid, in particular, are far different than a simple wheeled robot: the humanoid is a real extreme in heterogeneity relative to a wheeled robot. These results simply indicate how well the actual commands used match with the visual classifications. The actual visual output may match the primitive that a segment was classified as (e.g. the Bioloid's *kick* command may match perfectly with the imitator's *backward* behaviour). Even if the underlying commands do not match perfectly, or even well, the visual outcomes can still be learned by a physically different robot using my approach, allowing an imitator to learn from demonstrators of varying physiologies: a very positive result.

4.4 Evaluation of Forward Models

Having evaluated the low-level components of my approach, we can now turn to examining the performance of my imitation learning approach as a whole. To evaluate the ability of my system to learn from demonstrators that vary both in physiologies and skill levels, I used two separate training scenarios. The first scenario is where the imitator learns from three demonstrators: one that is physically identical, and two that are physiologically distinct from the imitator. The results of this evaluation are presented in Section 4.4.1. The second scenario is where the imitator learns from three demonstrators that are physiologically identical to the imitator, but all three

have different skill levels. The results of the scenario with demonstrators of varying skill are presented in Section 4.4.2.

In both scenarios, there are two phases to training a forward model to represent an imitator. These are described in detail in Section 3.4, but I briefly remind the reader of the phases of this approach here, since elements of this are important in the analysis of my evaluation. The first phase is where the forward models representing the demonstrators get trained (using demonstrations by the respective demonstrator only), while the forward model representing the imitator merely acquires candidate behaviours. Once this phase is complete, each forward model for a demonstrator has only been exposed to a single demonstrator: itself. The next phase trains a forward model to represent an imitator. This is the phase of the training process where the forward model for the imitator processes the demonstrations, but first passes each demonstration to all of the forward models representing the demonstrators. When this is complete, the forward model for each demonstrator will have been exposed to four sets of demonstrations: the demonstrations for the demonstrator it represents twice (once in the first phase, and then again when the imitator passes those demonstrations to it), and once each for the other two demonstrations. At the end of this second phase, training is complete, and the forward model representing the imitator deletes any behaviours that are not permanent. This leaves a single forward model that represents the most useful behaviours the imitator was exposed to during the entire training process. This forward model can then be used to control the Lego Mindstorms robot (the imitator).

The following sections use this training process to explore the two scenarios de-

scribed previously. Both of these scenarios use the exact same training procedure, though there are minute differences in the order that demonstrations are exposed to the various forward models. I will elaborate on this further in each section separately.

4.4.1 Learning from Physically Heterogeneous Demonstrators

To evaluate the ability of the imitator to learn from various demonstrators, I used the Bioloid and Citizen robots, as they are both very physiologically distinct from the Lego Mindstorms robot that I use as the imitator. They are also distinct in different ways: in the manner of locomotion for the humanoid Bioloid, and in size for the Citizen. I also used a Lego Mindstorms robot as a baseline demonstrator, with the control program acquired from a past RoboCup team from 2004 as described in Section 3.2.1. All demonstrators are highly skilled, often scoring goals and very rarely scoring on their own goal. A demonstrator only scores on its own goal as a result of the more degenerate field configuration, shown on the left in Figure 4.4. The total number of goals each demonstrator scored during all 50 of their individual demonstrations is given in Table 4.6.

Demonstrator	**Goals Scored**	**Wrong Goals Scored**
RC2004	27	4
Citizen	15	3
Bioloid	12	1

Table 4.6: The number of goals and wrong goals scored for each demonstrator.

I wanted to determine if the order in which an imitator using my approach is

exposed to the various demonstrators - specifically, the degree of heterogeneity - had any effect on its learning. I chose to order the demonstrators in two ways. The first is in order of similarity to the imitator. In this ordering, the RC2004 demonstrator is first, then the Citizen demonstrator (which is much smaller than the imitator, but still a two-wheeled robot), and finally the Bioloid demonstrator. The shorthand I have adopted for this ordering is *RCB*. The second ordering is the reverse of the first, that is, in order of most physical differences from the imitator. The second ordering is thus Bioloid, Citizen, RC2004, or *BCR* for short. The orderings determine what order the forward models for the given demonstrators are trained on their individual demonstrator data, and also the order that the imitator passes each demonstration to the demonstrators during the training of the forward model representing the imitator.

For each of the two orderings, I ran 100 trials, collecting all the statistics shown in Table 4.1 and Table 4.2. The results of the forward model training processes using the RCB and BCR demonstrator orderings are presented here. All the following data has been averaged over 100 trials.

Percentage of Prediction Rounds When Match Found		
Robot that Forward Model Represents	**RCB**	**BCR**
Imitator	91.4% (1.2%)	91.3% (0.9%)
RC2004	89.4% (0.6%)	89.4% (0.6%)
Citizen	91.6% (1.7%)	91.5% (1.1%)
Bioloid	81.3% (1.1%)	80.9% (1.3%)

Table 4.7: The percentage of prediction rounds where a prediction was found to match the demonstration are given, with their corresponding standard deviations in parentheses.

During the training process, each forward model has a number of prediction

rounds, where all the behaviours generate predictions of their effects on the current state of a demonstration (Section 3.4). The forward models representing the demonstrators go through their own demonstrations twice: once when first training the forward model to represent a demonstrator, and then again when the imitator passes demonstrations to the forward models to elicit nominations of candidate behaviours. This means that the forward models representing the demonstrators have one more set of demonstrations to go through than the forward model representing the imitator, and as a result they have more prediction rounds than the forward model for the imitator.

During a prediction round, a forward model compares all of the predictions it generated for the current demonstrator segment of that round, to the actual results of the demonstrations, looking for at least one prediction that matches one of the demonstrator segments. It is not always possible for a forward model to find a demonstrator segment that matches the predictions generated in a given prediction round. In Table 4.7, the percentage of prediction rounds where a prediction was found to match a segment in the demonstration are shown with the standard deviations given in parentheses. The forward models representing the two-wheeled demonstrators (the Citizen and RC2004) as well as the forward model representing the imitator have equally high prediction matching rates, while the forward models representing the Bioloid have a lower, though still respectable rate of matching predictions.

I believe the two-wheeled demonstrators have a slight advantage over matching their predictions to demonstrations than the Bioloid demonstrator. The similarity of physiologies for the Citizen and RC2004 demonstrators means that the demon-

strations that the forward models representing them are trying to approximate will mostly be from demonstrators that are also two-wheeled. This means that of all the demonstrations these two forward models are exposed to, 3/4 will be from two-wheeled robots (for example, the Citizen will view its own demonstrations twice, and the RC2004 demonstrations once, leaving only the Bioloid). It is interesting that the forward model for the Citizen demonstrator seems to have no difficulty keeping up with the forward model for the RC2004 demonstrator, even though the RC2004 demonstrator is identical in physiology to the imitator, while the Citizen is about 1/10 the imitator's size. This could be due to the fact that the simple motors on the Citizen robot only allow for similar motions as the imitator's primitives. Even though the Bioloid is very distinct physiologically from the imitator, the forward model for the Bioloid still performs well. This validates my approach to learning from physically heterogeneous demonstrators. The order that the demonstrators are exposed to the forward models in my implementation do not significantly impact these results. It would be interesting in future work to use an equal number of humanoid demonstrators as two-wheeled demonstrators to see if this levels the playing field.

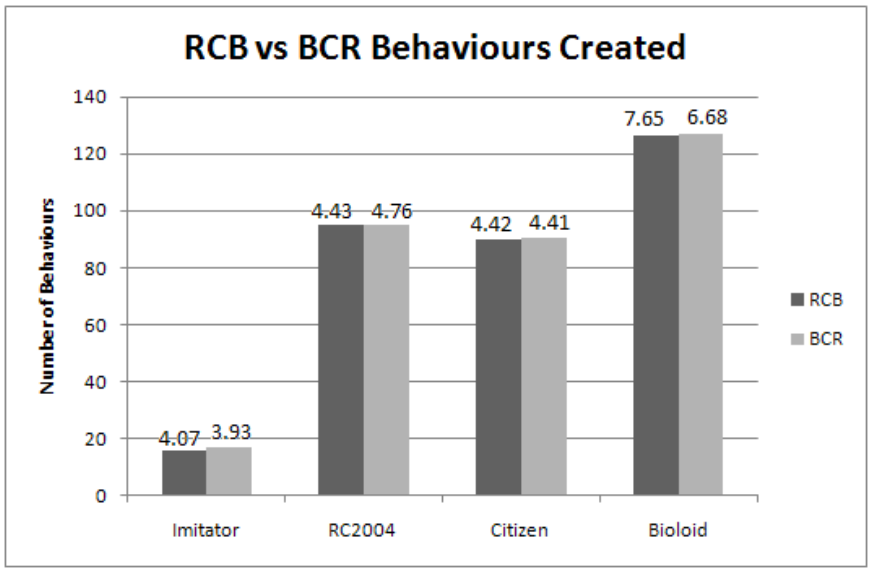

Figure 4.7: The number of behaviours created, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

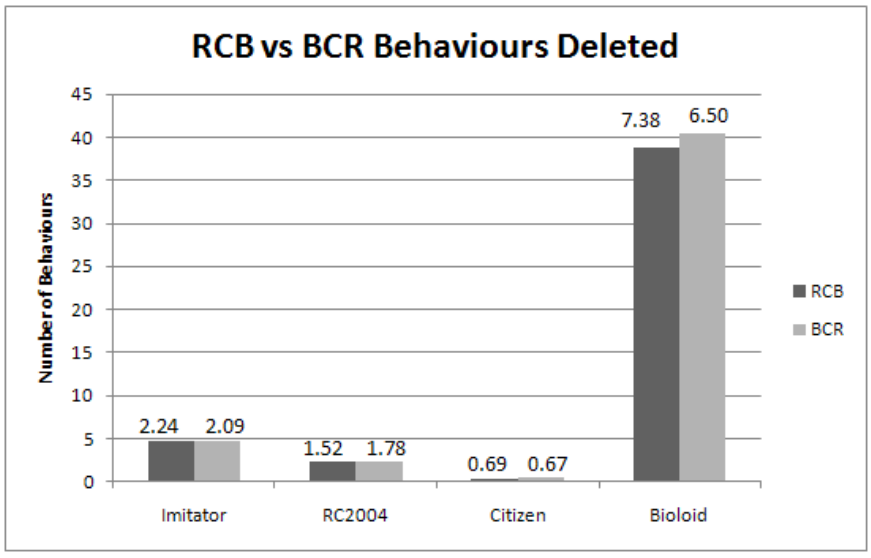

Figure 4.8: The number of behaviours deleted, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

Figures 4.7 and 4.8 show results for the number of behaviours created and deleted for each of the forward models representing the given demonstrators, with the two orderings for comparison purposes and standard deviations given above each bar. It can be seen that the RCB and BCR demonstration orderings do not affect the number of behaviours created or deleted from any of the forward models. The forward models representing the Bioloid demonstrator can be seen to create many more behaviours than the other forward models (and have a higher standard deviation), but they also end up deleting many more than the others. Perhaps the vast difference in physiology from the other two-wheeled robots cause the forward models representing the humanoid to build many behaviours in an attempt to match the visual outcome of the Bioloid's demonstrations. When trying to use those behaviours to predict the outcome of the other two-wheeled robot demonstrators, they do not match frequently enough (i.e. they are not a useful basis for imitation), and are eventually deleted as a result. This seems to be the case, as the forward models representing the two-wheeled (RC2004 and Citizen) demonstrators have roughly the same numbers of behaviours created and deleted, across both RCB and BCR orderings. The forward models representing the two-wheeled demonstrators also have far fewer behaviours deleted, indicating that the forward models had an easier time modelling those demonstrators with the imitator's primitives. This makes sense, as they would have roughly the same movement patterns as the imitator, due to their shared physiologies. The results of the conversion process discussed in Section 4.3 make a similar indication, as both the RC2004 and Citizen demonstrators have higher conversion accuracies than the Bioloid.

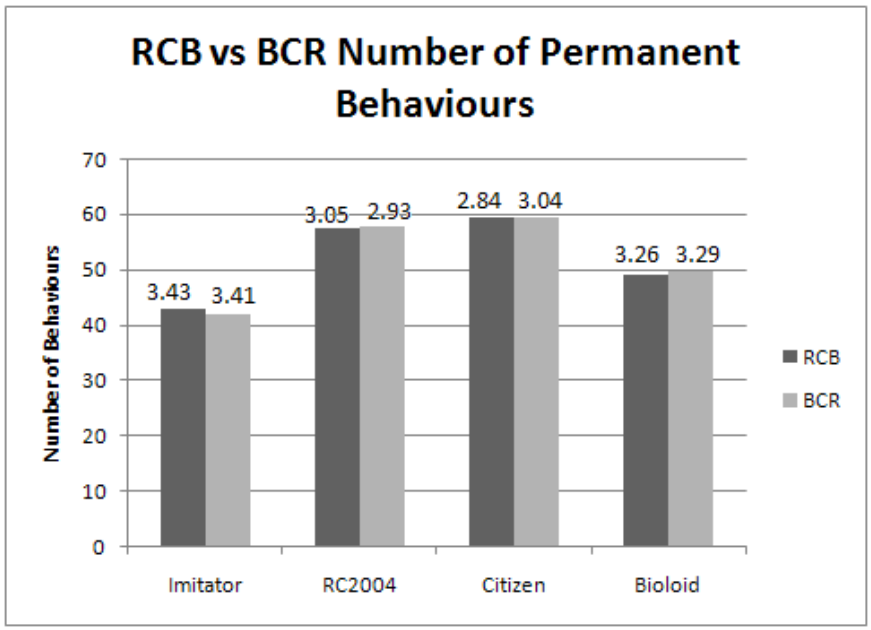

Figure 4.9: The number of permanent behaviours in each forward model, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

In Figure 4.9, the number of permanent behaviours for each of the forward models are shown along with standard deviations above each bar, grouped by RCB and BCR to see any effect on demonstrator orderings. It can be seen that the orderings do not affect the number of behaviours made permanent to any of the forward models, indicating that ordering does not affect the number of useful behaviours acquired by the forward models representing the demonstrators, or the imitator itself. Even though the Bioloid has a very different physiology, the forward models representing its actions still learn a relatively similar number of behaviours as the other two forward models for the other demonstrators. The forward models representing the imitator have fewer permanent behaviours, partly because the forward model for an imitator filters the candidate behaviours given to it by the forward models representing the

demonstrators, but it could also be due to the fact that the imitator is only exposed to each set of demonstrations once, while the other forward models see all demonstrations once, but the demonstrations for their particular demonstrator twice.

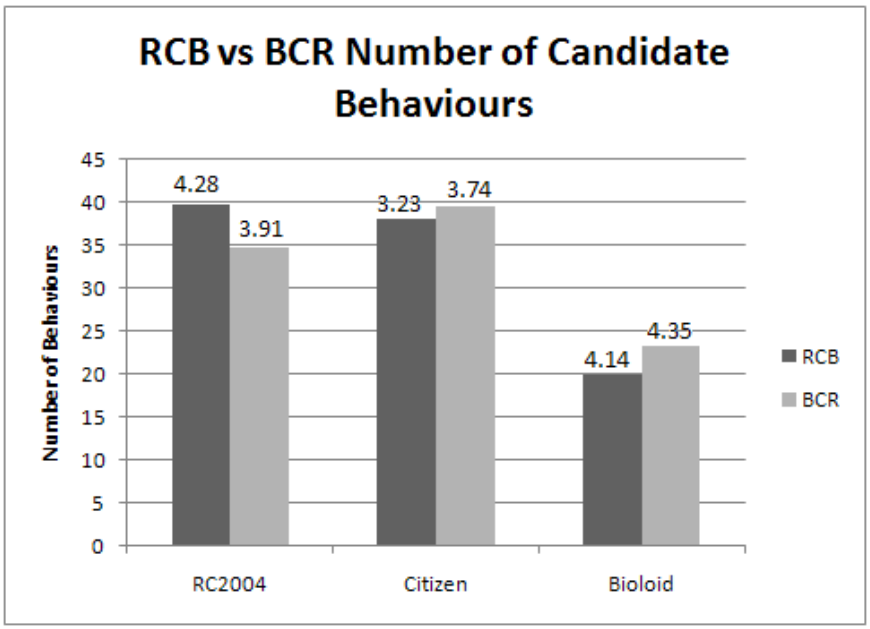

Figure 4.10: The number of candidate behaviours moved to the forward model representing the imitator, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

Figure 4.10 shows the number of candidate behaviours nominated by each forward model representing the demonstrators, as well as their standard deviations above each bar. It can be seen that the order in which the forward models are trained slightly affects the number of candidate behaviours moved to the forward model representing the imitator. The forward models for the Citizen demonstrator are mostly unaffected by ordering, which makes sense as they are in the middle for both orderings. The forward models representing the RC2004 demonstrator have slightly more candidate behaviours nominated when they are first (the RCB ordering) than when they are

last (the BCR ordering). The forward models representing the Bioloid have similar results (though with the opposite orderings), with more candidate behaviours in the BCR ordering than the RCB ordering. I believe this is largely due to the number of candidate behaviours that get rejected because they already exist in the forward model representing the imitator, but the standard deviation involved could also explain this. The results for duplicate candidate behaviours can be seen in Figure 4.11. The forward models for the RC2004 demonstrator have fewer duplicates rejected when they are first (RCB), as is true for the forward models for the Bioloid when the Bioloid is first (BCR). In both cases, however, this is not much of a difference given the standard deviations involved. The forward models for both demonstrators appear to have more candidate behaviours rejected when they are last in the ordering, but this could also be explained simply by the standard deviations involved. Again, the forward models representing the Citizen demonstrators are less affected by ordering, as they appear in the middle both times.

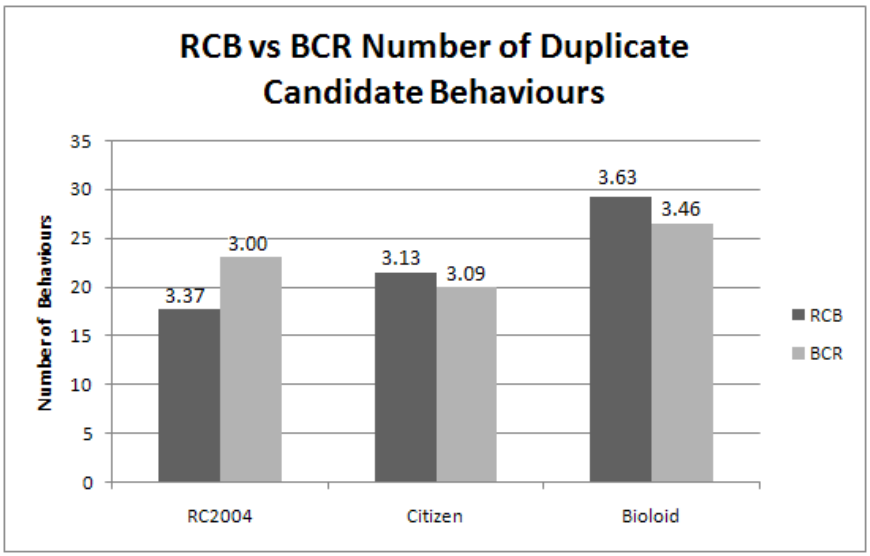

Figure 4.11: The number of candidate behaviours not moved to the forward model representing the imitator, because they were already there, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

Similar results are found when looking at the number of candidate behaviours that achieve permanency to the forward model representing the imitator. In Figure 4.12, the forward models representing the RC2004 and Bioloid demonstrators can be seen to have more behaviours made permanent to the forward model for the imitator when their demonstrations appear first in the ordering, but this is explainable by considering the standard deviation. The forward models representing the two-wheeled demonstrators seem to have an advantage in the number of their candidate behaviours becoming permanent to the forward model for the imitator over the forward models representing the Bioloid demonstrator. This is likely due to the same reasons of physiology discussed when looking at the number of behaviours created and deleted by each of the forward models.

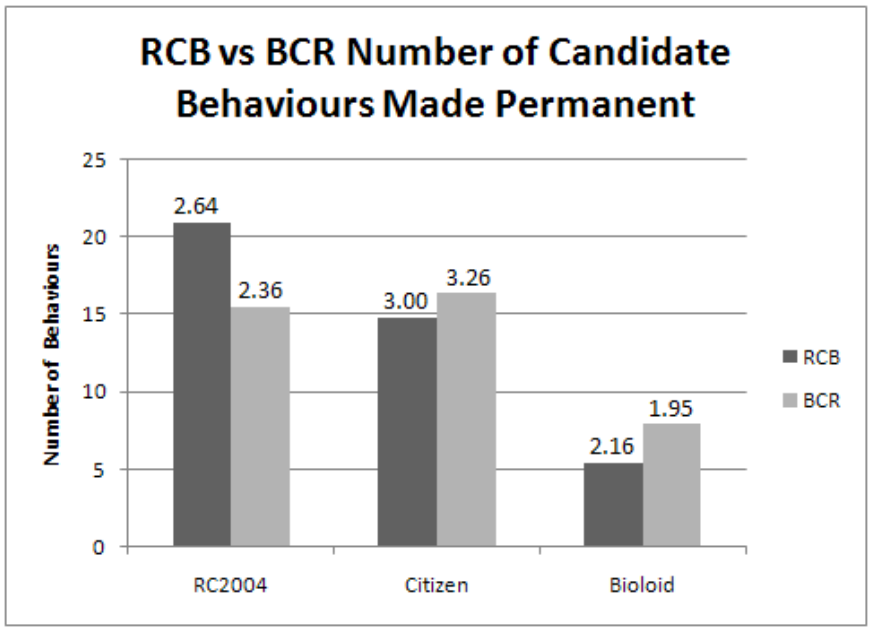

Figure 4.12: The number of candidate behaviours that earned permanency after being moved to the forward model representing the imitator, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

To get an idea of how the learning preference (LP) of each forward model changes during the training process, I recorded the value of the LP at regular intervals. Figures 4.13 - 4.18 all show intervals of approximately every 200th LP change. All charts show the LP changes over time for all 100 trials. I have grouped the forward models for each demonstrator together with the two different training orders, so that it can be seen that the ordering does not affect the LP changes over time. It can be seen that all demonstrators in these differing physiology trials are considered to be skilled by the imitation learning architecture, as each has its forward model's LP increase rapidly (at least within the first half of training and usually much faster) to the maximum value of 1, and then makes small fluctuations around this maximum. It is interesting

to note the effects of physiology on the LP over time, as the forward models for the Bioloid demonstrator take longer to reach the maximum than the forward models for the Citizen or RC2004 demonstrators.

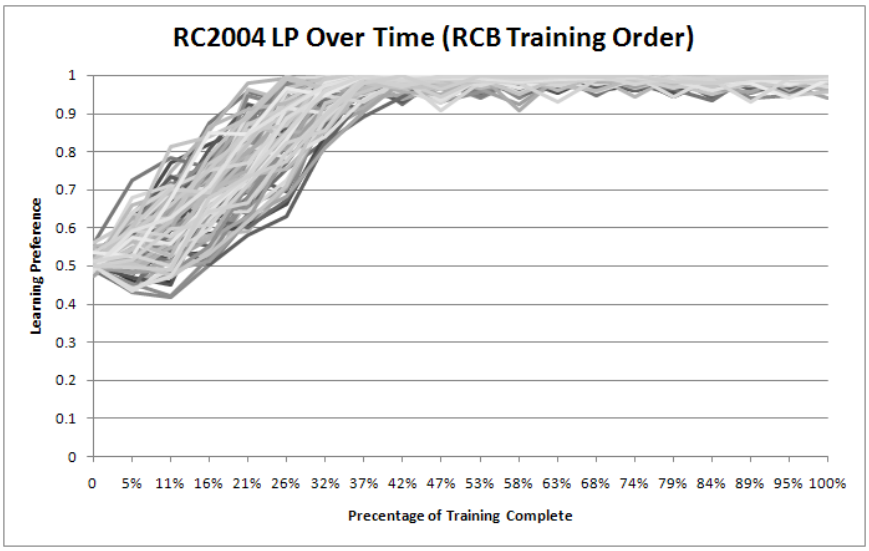

Figure 4.13: The change in LP over time for the RC2004 demonstrator (RCB ordering).

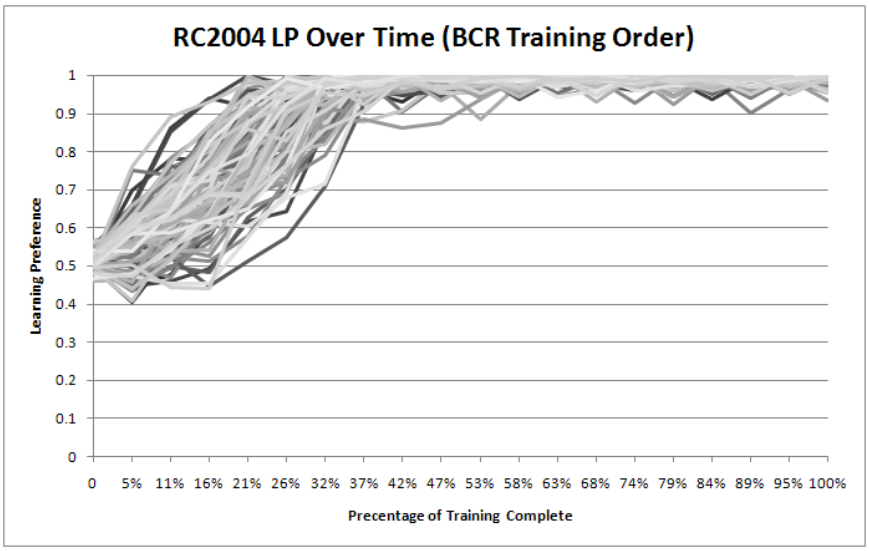

Figure 4.14: The change in LP over time for the RC2004 demonstrator (BCR ordering).

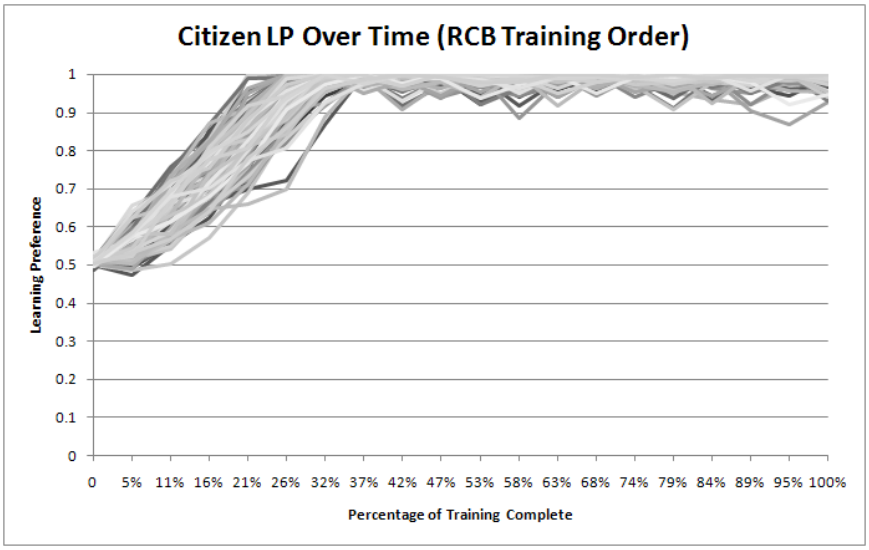

Figure 4.15: The change in LP over time for the Citizen demonstrator (RCB ordering).

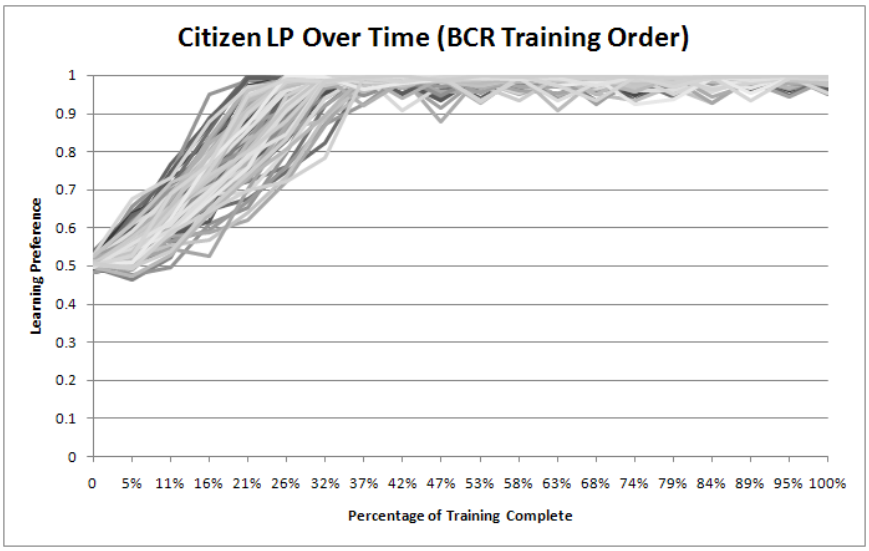

Figure 4.16: The change in LP over time for the Citizen demonstrator (BCR ordering).

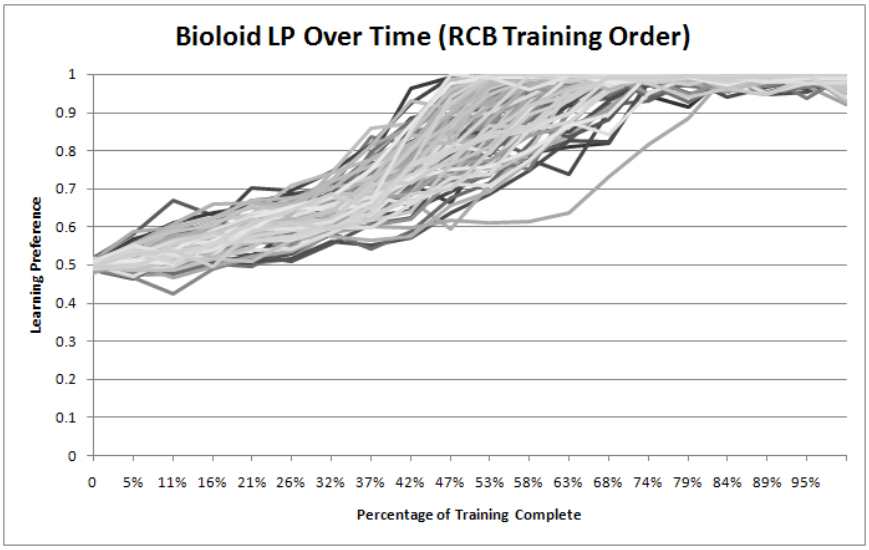

Figure 4.17: The change in LP over time for the Bioloid demonstrator (RCB ordering).

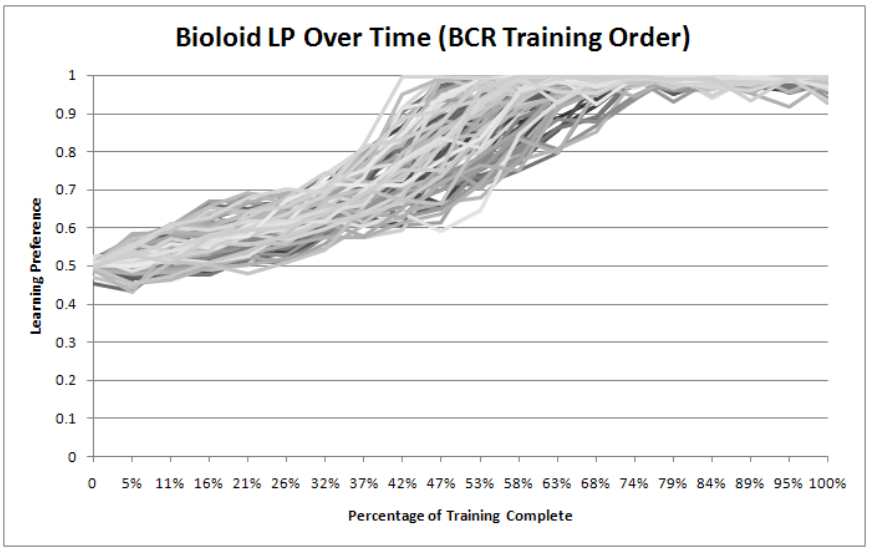

Figure 4.18: The change in LP over time for the Bioloid demonstrator (BCR ordering).

Demonstrator Ordering	Goals Scored	Wrong Goals Scored
RCB	11	9
BCR	7	13

Table 4.8: The number of goals and wrong goals scored for two imitators trained with the different demonstrator orderings.

To evaluate the performance of the imitators trained using this approach, I selected two imitators from the learning trials evaluated in this section at random (one from the RCB training order, and one from the BCR order). I used the forward models to control (as described in Section 3.4) the Lego Mindstorms robots and recorded them in exactly the same way that I recorded the demonstrators, for 25 shots on goal in each of the two field configurations (Figure 4.4) for a total of 50 trials. Table 4.8 shows the results of these penalty kick attempts by the two imitators trained using my framework. The results are not great for either ordering, but the orderings

themselves do not seem to indicate a large difference between them. I believe that the reason the number of goals are poor for both orderings is due to the statistical methods of attributing ball movement to behaviours (Section 3.3) and behaviour preconditions (Section 3.4). These methods rely on having many samples of the visual effects of behaviours moving the ball, and they did not have enough to properly predict them. By not having proper ball predictions or preconditions, the behaviours were left depending too much on driving towards the ball as the best option given the LP shaping criteria used to choose the best behaviour at a given time (Section 3.4). Another possible reason the trained imitator did not perform particularly well is that the current behaviour being executed was not stopped if another behaviour became more applicable during its execution, causing the imitator to stick to a chosen behaviour, even if using that behaviour resulted in poor results (such as scoring in the wrong goal). The only time the execution of a behaviour was stopped was when the primitives it was about to execute predicted that it might move the imitator off the field, since the imitator was not tasked with learning behaviours that kept it on the field. My architecture could be used to learn behaviours that keep the imitator on the field if the LP shaping was modified, and it would be interesting to see how well an imitator could perform given this additional LP shaping criteria. The predictions generated by behaviours with larger prediction sequences may also affect these results, as each prediction relating to a primitive is chained into the next prediction. The prediction deviation becomes large fairly rapidly, which prevented very large and elaborate behaviours from forming, since their predictions would never be chosen to match the demonstration.

In the next section I will evaluate my implementation in the scenario where the imitator learns from three demonstrators that are identical in physiologies, but vary in skill level. For these trials I required a demonstrator that had an average performance relative to the ExpertDemonstrator and PoorDemonstrator. I chose the RCB imitator trials from Table 4.8 to be used as a demonstrator of average skill, and so I call it AverageDemonstrator in the next section. I chose this model because it exhibited a more average range of goals scored and goals scored on itself than the BCR imitator, when compared to the PoorDemonstrator and the ExpertDemonstrator in the next section. I chose to use a forward model trained with my system to control the average demonstrator to show that an imitator trained using my architecture can pass knowledge on to other imitators also using my architecture.

4.4.2 Learning from Demonstrators of Varying Skill

This section discusses the results of my implementation's ability to train an imitator through the observation of demonstrators of varying skill. I chose to use three demonstrators: one poorly skilled demonstrator that simply drives straight at the ball referred to as the PoorDemonstrator, a demonstrator of average skill referred to as the AverageDemonstrator, and a highly skilled demonstrator (the same RC2004 demonstrator as used in the differing physiology trials) referred to as the ExpertDemonstrator in this phase of experiments for greater clarity. To avoid any influence of demonstrator ordering on these experiments, during the phase where the forward models representing the demonstrators are trained, each demonstration is chosen randomly. This ensures that the order will not affect the number of candidate behaviours

Percentage of Prediction Rounds When Match Found	
Robot that Forward Model Represents	**Percentage**
Imitator	85.9% (4.14%)
PoorDemonstrator	81.3% (0.07%)
AverageDemonstrator	79.1% (0.04%)
ExpertDemonstrator	87.2% (2.74%)

Table 4.10: The percentage of prediction rounds where a prediction was found to match the demonstration for each forward model representing a given demonstrator, with their corresponding standard deviations in parentheses.

that are nominated by any one demonstrator during the first phase of training. In the second phase of training, where the forward model representing the imitator is trained, demonstrations are processed in random order as they always are in this phase. The number of goals and wrong goals scored by each of the demonstrators can be seen in Table 4.9. This verifies the performance of each of the demonstrator types as good, average, or poor.

Demonstrator	Goals Scored	Wrong Goals Scored
PoorDemonstrator	13	23
AverageDemonstrator	11	9
ExpertDemonstrator	27	4

Table 4.9: The number of goals and wrong goals scored for each demonstrator.

To evaluate my system on demonstrators of varying skill I ran 100 trials, collecting all the statistics shown in Table 4.1 and Table 4.2. The results of the forward model training processes are presented here. All the following data has been averaged over 100 trials.

In Table 4.10, the percentage of prediction rounds where a prediction was found

to match one of the segments in a demonstration are shown, with their standard deviations in parentheses. The prediction rounds are explained in Section 3.4, and a brief discussion of the statistics gathering is in Section 4.4.1 (accompanying Table 4.7). The results in Table 4.10 show that the forward models for the demonstrator with the highest skill also has the highest prediction match percentage. This is due largely to the fact that the forward models for the ExpertDemonstrator develop higher LPs, and they create and keep more behaviours that the other two demonstrators. Figures 4.19 and 4.20 show the number of behaviours created and deleted for the various forward models as well as the standard deviations above each bar. It can be seen that the forward models for the ExpertDemonstrator have fewer behaviours created than the others, though they also have far fewer of them deleted. This indicates that the behaviours learned by the forward models for the ExpertDemonstrator are more useful than those learned by the other models. The standard deviations of the number of behaviours created by the forward models for the imitator is fairly high, but this is dependent on how many candidate behaviours were added and made permanent to the forward model for the imitator by the various forward models representing the demonstrators. This is because the forward models representing the imitator must develop their own behaviours if they are not given adequate candidate behaviours to work with. The standard deviations of the PoorDemonstrator and AverageDemonstrator might be so high because of the random order that demonstrations are exposed to the forward models during training. The amount of demonstration examples of a particular quality that a forward model is exposed to for its particular demonstrator (for example, if the PoorDemonstrator is exposed to most of the demonstrations

where it scored in the wrong goal) could cause the LP of a forward model for a given demonstrator to be temporarily skewed, though exposing forward models to demonstrations in random order should address this.

There is not a large difference between the models representing the PoorDemonstrator or AverageDemonstrator. I believe that this is related to the rough statistics used when a forward model is controlling the imitator as was done for the AverageDemonstrator. The LP shaping criteria are used during the control process for selecting a behaviour to execute as discussed in Section 4.4.1. The statistical methods used to calculate preconditions were not robust enough given the task at hand, and had small sample sizes to work with. This resulted in the criteria of the robot driving closer to the ball overriding the other LP criteria in most cases. When this happened, the imitator behaved in a very similar manner to the PoorDemonstrator. This could be avoided if future work explored methods of gathering more precondition statistics, possibly in simulation for initial training, moving to physical robots later. My work focused on imitation learning with physical robots, and so the number of training sequences recorded were far fewer than a simulation may be able to provide. My work has been designed from the ground up to deal with the many problems that arise with physical robots and vision systems. My implementation could be used to learn from simulated demonstrations with little to no modification. An imitation learner developed in simulation in contrast, would require a great deal of work to deal with all the control and vision problems as well as the break-downs that arise when dealing with physical robots and vision systems.

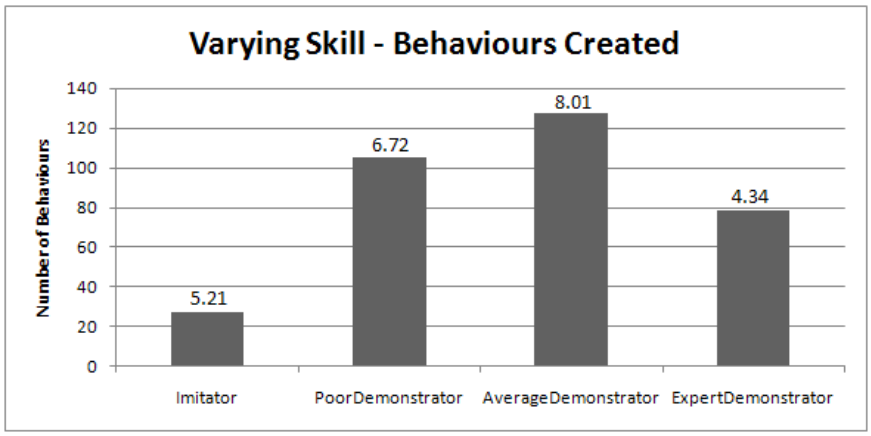

Figure 4.19: The number of behaviours created. Corresponding standard deviations are given at the top of each bar.

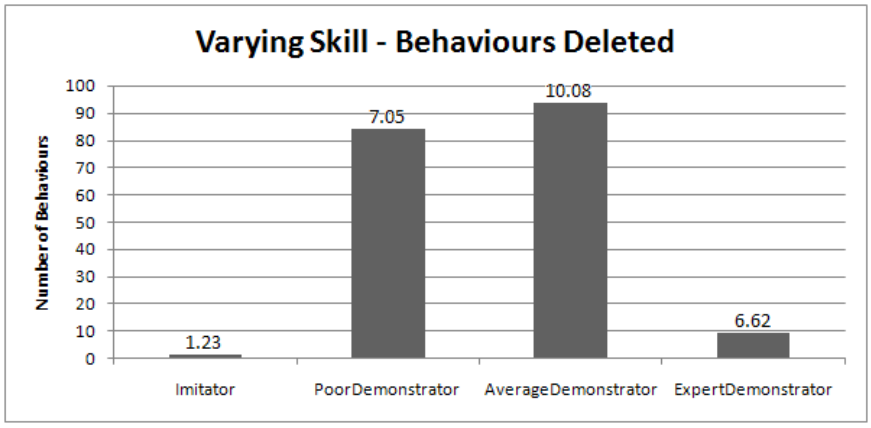

Figure 4.20: The number of behaviours deleted. Corresponding standard deviations are given at the top of each bar.

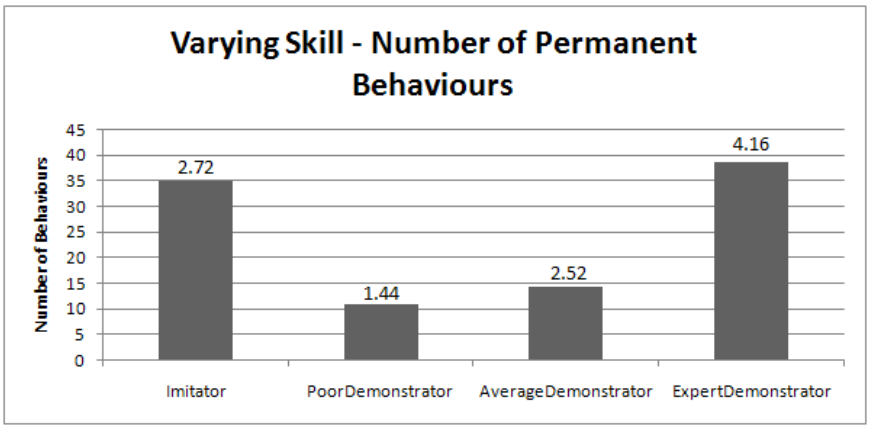

Figure 4.21: The number of permanent behaviours. Corresponding standard deviations are given at the top of each bar.

Figure 4.21 shows that the forward models for the ExpertDemonstrator retain more of the behaviours they create (make them permanent) than the other forward models. This validates my approach to behaviour permanencies that decay over time. The less skilled demonstrators have lower LPs, and therefore higher decay rates (detailed in Section 3.4). Since the forward models representing the ExpertDemonstrator have a higher LP than the others (shown in Figure 4.25), the forward models learn behaviours more quickly, and have their behaviours decay more slowly. The number of behaviours retained by each model is thus strongly related to the LP, which was my intention when employing demonstrator specific learning rates. Figure 4.22 shows that the number of candidate behaviours for the forward models for the less skilled demonstrators is far less than that of the forward models for the ExpertDemonstrator. This also shows the effect of LP on the behaviours that get nominated as candidates for the forward model representing the imitator. These results show that my imitation learning architecture weights its learning towards demonstrators that are highly

skilled relative to the other demonstrators.

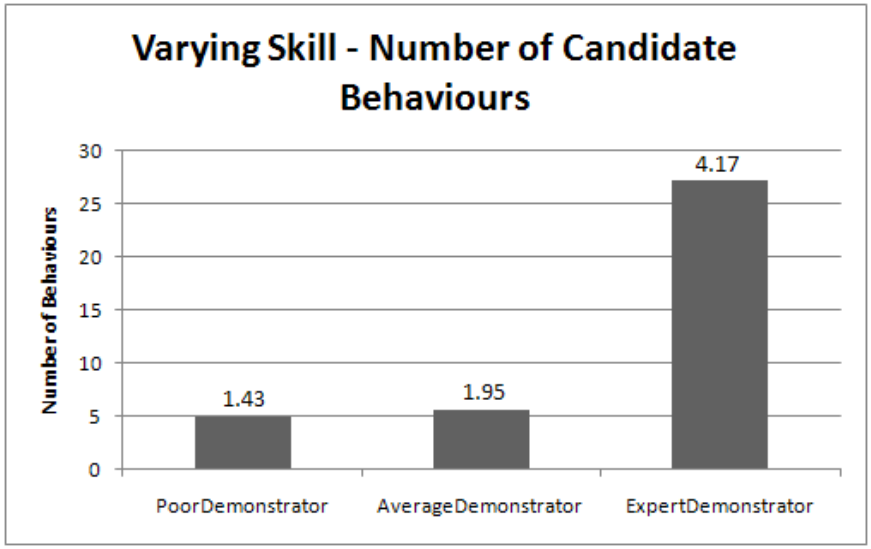

Figure 4.22: The number of candidate behaviours moved to the forward model representing the imitator. Corresponding standard deviations are given at the top of each bar.

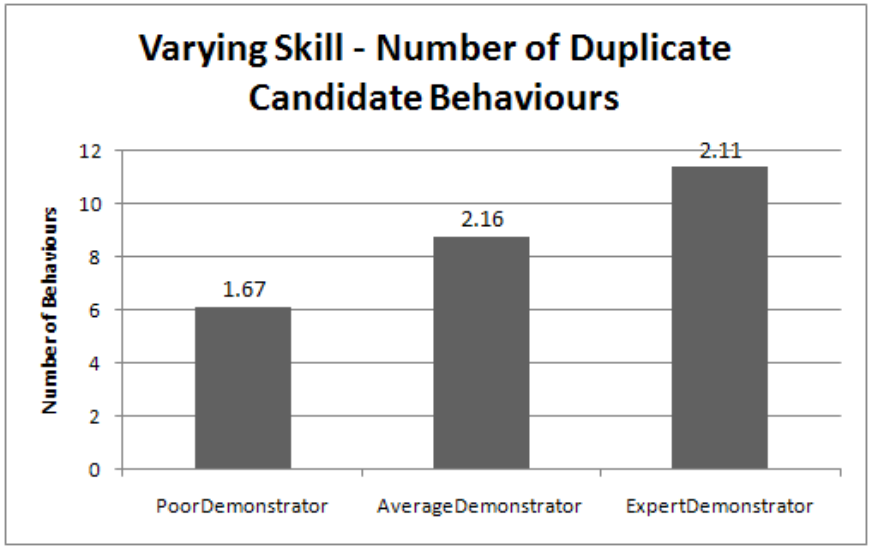

Figure 4.23: The number of candidate behaviours not moved to the forward model representing the imitator because they were already there. Corresponding standard deviations are given at the top of each bar.

It can be seen from Figure 4.23 that the forward models for the ExpertDemonstrator have many more candidate behaviours behaviours rejected due to duplication than the other forward models. This is simply due to the fact that the forward models for the ExpertDemonstrator nominate more candidate behaviours than the others both because the ExpertDemonstrator exhibits more useful behaviours, and as a result, the forward models representing the ExpertDemonstrator develop higher LPs. These higher LPs cause the forward models to create behaviours more quickly, as well as have existing behaviours become permanent and nominated as candidate behaviours faster than forward models with lower LPs. When the forward models with lower LPs do nominate a candidate behaviour, there is a good chance it is one that the ExpertDemonstrator has already nominated.

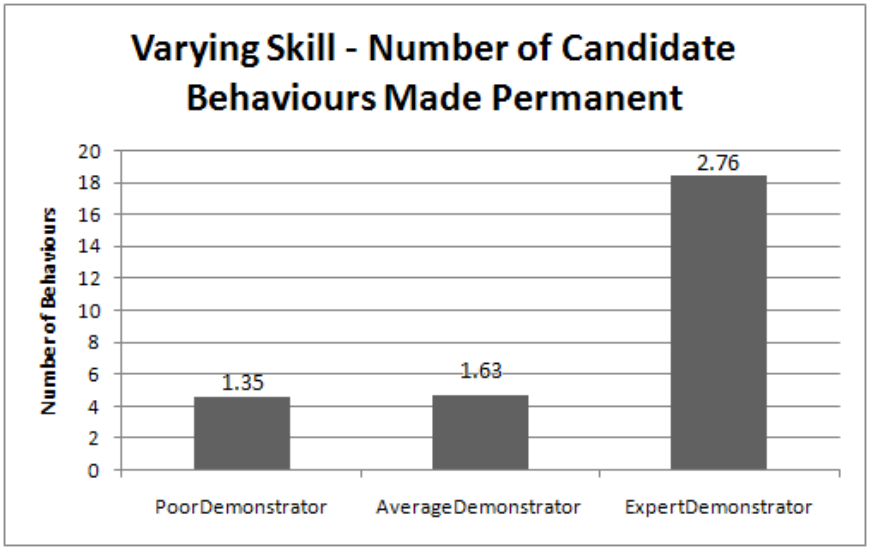

Figure 4.24: The number of candidate behaviours that earned permanency after being moved to the forward model representing the imitator. Corresponding standard deviations are given at the top of each bar.

The forward models representing the imitator receive many more of their perma-

nent behaviours from the forward models representing the ExpertDemonstrator than the other forward models. Figure 4.24 shows that the forward models representing the ExpertDemonstrators have far more of their candidate behaviours made permanent to the forward models representing the imitators. This is due to the larger quantity of the candidate behaviours they nominate, meaning that they have more useful behaviours throughout the training process than the other models.

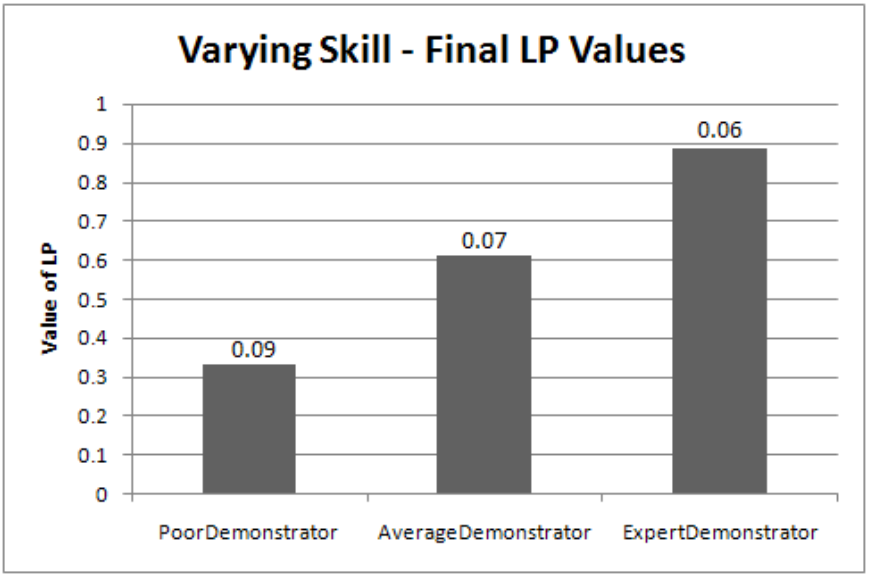

Figure 4.25: The final LP values assigned to each forward model. Corresponding standard deviations are given at the top of each bar.

In Figure 4.25 the final values of forward models for all the demonstrators are given. It can be seen that the poorly skilled PoorDemonstrator resulted in lower LP values for all the forward models trained to recognize it, the forward models for the average skilled AverageDemonstrator had roughly average LP values, and the forward models for the highly skilled ExpertDemonstrator ended up with relatively high LP values. This is exactly what would be expected of an approach where more is learned

from better-performing demonstrators.

To get an idea of how the learning preference (LP) of each forward model changes during the training process, I recorded the value of the LP at regular intervals. The following charts all show intervals of approximately every 200th LP change. All charts show the LP changes over time for all 100 trials. It can be seen that the PoorDemonstrator in these trials is considered to be poorly skilled by the imitation learning architecture fairly quickly, as the forward models representing it have their LP decrease below the average LP value (0.5), and then fluctuate around 0.3 but rarely surpasses the average LP value. The trend is downwards for most of the PoorDemonstrator's LP over time, but it does appear to trend slightly upward as training progresses. I believe that the few behaviours the PoorDemonstrator learns later in the training phase aid in generating predictions that match the demonstration, which in turn increases the LP of the PoorDemonstrator later in the training phase.

The fluctuations are more pronounced for all demonstrator's LP over time when the imitator learns from demonstrators of varying skill, compared to when it learns from demonstrators that have heterogeneous physiologies (Figures 4.13:4.18). I believe this is due to the two poorly skilled demonstrators exhibiting similar behaviours by often simply driving straight at the ball. This makes it easier for the forward models to match their predictions to these demonstrations, increasing the LP. When the ExpertDemonstrator's demonstrations are attempted to be matched to forward model predictions however, the forward models representing the more poorly skilled demonstrators will have a more difficult time matching predictions because many behaviours exhibited by the ExpertDemonstrator are never performed by the more

poorly skilled demonstrators (for example, performing a backwards turn into the ball to score a goal). If a forward model for a given demonstrator has difficulty finding prediction matches, the LP of that forward model will decrease. Switching between learning from demonstrators that exhibit similar behaviours to demonstrators that exhibit vastly different behaviours could be causing the forward models' LP over time. Similarly, switching between learning from demonstrators that exhibit poor skill levels to demonstrators that exhibit superior skill levels could also be influencing the forward models' LP over time to fluctuate. The forward models representing the AverageDemonstrator fluctuate just slightly above the average mark for the duration of the training process. The forward models representing the ExpertDemonstrator quickly exceed the average and fluctuate around the 0.9 mark.

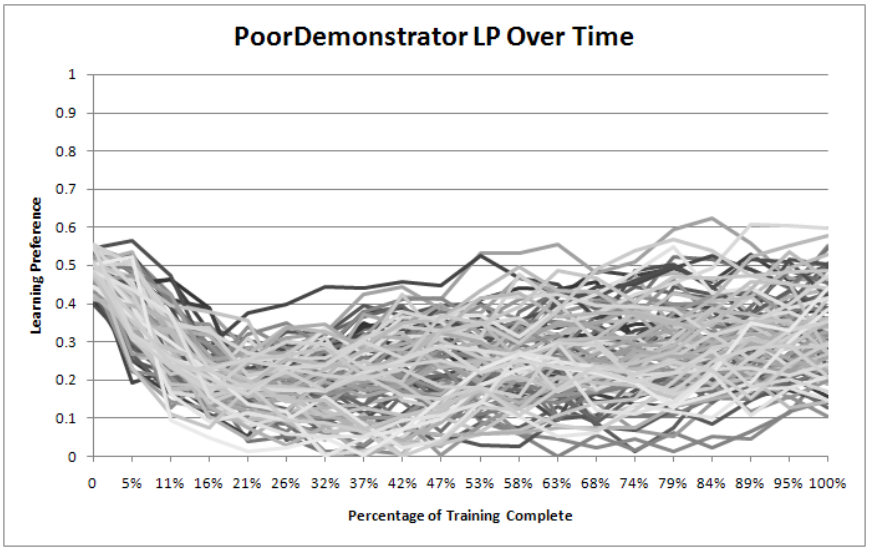

Figure 4.26: The change in LP over time for the PoorDemonstrator.

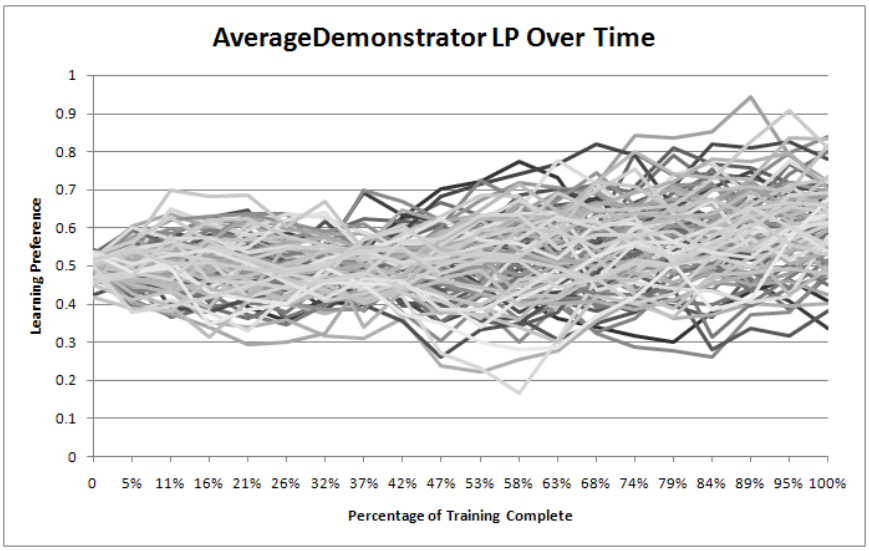

Figure 4.27: The change in LP over time for the AverageDemonstrator.

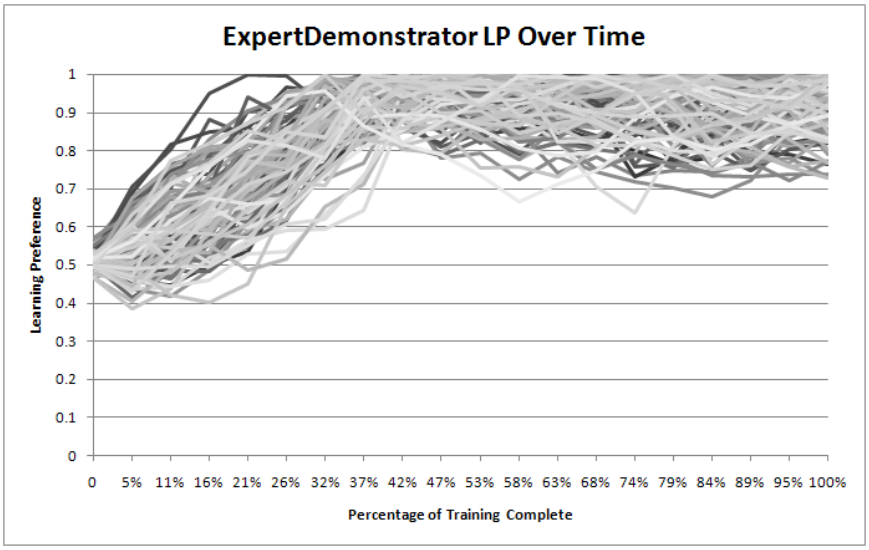

Figure 4.28: The change in LP over time for the ExpertDemonstrator.

Imitator	Goals Scored	Wrong Goals Scored
VaryingSkillTrained	11	13

Table 4.11: The number of goals and wrong goals scored for the imitator trained with demonstrators of varying skill levels.

To evaluate the performance of the imitators trained using this approach, I selected an imitator from these trials at random. I used the forward model to control (as described in Section 3.4) the Lego Mindstorms robot and recorded it in exactly the same way that I recorded the demonstrators, for 25 shots on goal in each of the two field configurations (Figure 4.4) for a total of 50 trials. Table 4.11 shows the results of these penalty kick attempts by the imitator trained from demonstrators of varying skill. The results are not any better than the results for the imitators trained from demonstrators of heterogeneous physiologies, though it is promising that they are roughly comparable (just a small difference in the number of goals scored). This indicates that the imitator was able to properly weight its learning towards the more highly skilled demonstrators. Though somewhat disappointing in an absolute sense, the performance of a robot using the imitator as a control program still showed that the imitator can learn behaviours from demonstrators and perform the same tasks as the demonstrators. Moreover, this imitator is achieving roughly the same results as that trained only with expert demonstrators in the previous experiment, despite having average and even poor demonstrators working with it. Future work could focus on the use of the forward models to be used for control more than behaviours generation.

4.5 Summary

In this Chapter I have presented the results and analysis of the experiments used to evaluate my approach to developing an imitation learning architecture that can learn from multiple demonstrators of varying physiologies and skill levels. The results in Section 4.4.1 show that my approach can learn from demonstrators that have heterogeneous physiologies. The humanoid demonstrator was not learned from as much as the two-wheeled robots that had similar physiologies to the imitator, but the imitator still learned approximately 12% of its permanent behaviours from the Bioloid, as seen in Figures 4.12 and 4.9. The Citizen robot was nearly as effective a demonstrator as the RC2004 robot. This is somewhat surprising, as the RC2004 robot has an identical physiology to the imitator, while the Citizen robot is approximately 1/10 the imitator's size. This could be partially due to the fact that the Citizen robot has the same limited command set as the imitator, compared to the vastly expanded set of primitive commands available to the RC2004. The Citizen moves much slower due to its size, so the demonstration conversion process must have compensated substantially to give the Citizen demonstrator results so close to the RC2004 demonstrator. Size differences, apparently, are easier to compensate for than differences in physiology, at least to the degree of the differences between wheeled and humanoid robots.

The LP over time charts in Section 4.4.1 show that all of the demonstrators are considered skilled by my framework, even though they differ greatly in physiologies. It can be seen that the Bioloid demonstrator's LP takes a bit longer during training to reach the maximum, but once there it fluctuates near the highest LP value just like the other demonstrators do. These results along with the prediction accuracies

given by Table 4.7 show that my system is adept at learning from demonstrators of heterogeneous physiologies.

My results presented in Section 4.4.1 also show that my framework is not affected drastically by the order that demonstrators are presented to the forward models. There is some effects from candidate behaviours being rejected if a forward model for a given demonstrator is the last to be trained, since the other forward models have already had a chance to get their candidate behaviours added, increasing chances of duplicates. In practice this does not seem to adversely affect the LP of any of the forward models, and so the order of demonstrations is mostly negligible.

The results given in Section 4.4.2 show that my system is also able to properly classify the relative ability of demonstrators that vary in skill level. The LP over time results strongly indicate that a demonstrator represented by a forward model in my framework will be judged appropriately. The results in Section 4.4.2 also show that the learning in my system is properly weighted towards the more skilled demonstrator. The ExpertDemonstrator had more permanent behaviours in the forward model representing it than any of the other less skilled demonstrators, but it also had more candidate behaviours made permanent by the forward model representing the imitator. This means that the more skilled demonstrator had more of its behaviours learned an incorporated into the final model that represents the imitator.

The results for the performance of my forward models when used as control systems did not perform as well as the expert demonstrators, but they still were able to control the imitator adequately. The main focus on my research was in developing an imitation learning architecture that could learn from multiple demonstrators of vary-

ing physiologies and skill levels. The results of the conversion processes, predictions, and the influence that the LP of a forward model for a given demonstrator has on what an imitator learns from that particular demonstrator all indicate that the learning architecture I have devised is capable of properly modelling relative demonstrator skill levels as well as learn from physiologically distinct demonstrators. A stronger focus on the refinement of behaviour preconditions and control could make my entire system more robust.

In the next Chapter I will discuss how these results address the research questions asked by this thesis. I will also contrast my work with other imitation learning approaches that influenced it. I will also draw conclusions from this research and discuss future work.

Chapter 5

Conclusion

This thesis has presented an approach to developing an imitation learning system that can learn from multiple demonstrators of varying physiologies and skill levels. In this chapter I will address the research questions this thesis sought to answer, distinguish important differences between this and similar previous work, and discuss my contributions and explore some possible avenues of future work.

5.1 Answers to Research Questions

Chapter 1 put forward several research questions that have driven the development of the work presented in this thesis. I will repeat these questions here, and then address them from the perspective of my implementation and its evaluation.

1. Can imitation learning be done from multiple heterogeneous demonstrators by modelling each demonstrator individually?

2. Can learning be favoured towards more skilled demonstrators by comparing

their individual models?

3. Are atomic primitives adequate starting points for learning complex behaviours?

The results presented in Chapter 4 show that imitation learning can be done successfully by individually modelling each demonstrator that the imitator is exposed to. My architecture successfully modelled multiple heterogeneous demonstrators by modelling each one individually, which enabled the imitator to build unique models of each demonstrator's actions. These unique models were then used to successfully train an imitation learner to perform the same task as that of the demonstrators. The skill of the imitator at the task was not great, but the forward models used to model the individual demonstrators performed very well. Each forward model was able to model its demonstrator, even when the demonstrators were very different in physiologies than the imitator.

The results of my experiments also show that the imitator's learning was favoured towards the more skilled demonstrators. In my results comparing the learning preference of the various forward models representing the demonstrators of varying skill, the expert demonstrator had a higher LP than the less skilled demonstrators. The averagely performing demonstrator had an average LP value, and the poorly performing demonstrator had the lowest LP of all three. The number of behaviours that the imitator took as candidate behaviours was also higher from the expert demonstrator. This indicates that the LP had the desired effect of speeding the rate with which the imitator learned from more skilled demonstrators, while slowing the rate that behaviours are learned from demonstrators of less skill. Moreover, the imitator trained in this experiment was roughly equivalent in skill to the imitator trained in

the experiment where all demonstrators differed in physiology but were highly skilled. This means that despite having many poor demonstrations, it learned as well as an imitator that had only skilled agents to learn from.

The atomic primitives used in my approach were used to make behaviours, though these were not necessarily complex. The behaviours learned did not encompass a large number of primitives, and rarely predicted the ball (some of the behaviours created during experimentation can be seen in Appendix B. This is mostly due to the small field size used in my experiments, as most of the demonstrators could cover the field very quickly, leaving less data for the imitator to construct more complex behaviours from. The nature of my imitation learning system allows for the primitives to be defined relatively easily, as they were manually selected to be the most atomic motions available to the robot. This could allow for an initial round of training the imitator model using the atomic primitives that I used in my work. The second round of training could then take the most used behaviours learned in the first round and use them as the primitives. This could allow for more elaborate behaviour creation, though it would be time consuming, and the behaviour prediction capabilities would need to be excellent even for behaviours with many primitives. I believe that the predictions were not accurate enough once behaviours contained too many primitives, as the prediction inaccuracy became compounded. The behaviours created by my system were adequate for the imitator to perform the demonstrator penalty kicking task, though it did not learn any subtleties of ball manipulation. The imitator also rarely used its preconditions when using its forward model for control. I believe that more robust statistical methods for both the predictions and preconditions could aid

the performance of my system.

While the above questions were the focus of this thesis, I have additional questions specifically related to elements of the approach I have developed for imitation:

1. Can counting the frequencies of primitives and behaviours occurring in sequence be used to adequately learn behaviours through imitation?

2. Is a behaviour decay system enough to keep behaviour growth in check?

3. Can a forward model designed to learn a behaviour repertoire through observations also be used to control a robot with minimal intervention?

My results show that the method of triggering behavior creation based on frequency counting to building behaviours worked very well at building behaviours of moderate length. I believe that this approach is valid, and that with better prediction accuracies, could be used to learn much more complex behaviours. The problem with counting frequencies of occurrence in sequence is that my system cannot learn behaviours that are composed of behaviours and primitives occurring simultaneously. This would be necessary for a humanoid robot, where locomotion can go hand in hand with manipulation of objects, for example. It may be possible to adapt my system of frequency counting to also count when two or more behaviours and primitives occur at the same time. Designing a recognition algorithm to recognize multiple simultaneously occurring behaviours would not be a simple task however, since taking a single visual outcome and attributing responsibility for it to more than one act on the part of the robot is extremely difficult. Primitives would have to be learned not only in terms of their own visual outcomes, but in terms of those when two (or more) are carried

out simultaneously. Some groupings may make no sense, while other combinations may result in similar visual outcomes.

My results also showed that a behaviour decay system can keep the number of behaviours in the forward models of my implementation in check. The decay of behaviours also has the added benefit of focusing the behaviours kept by the imitator on the most useful behaviours to predict the demonstrations. Having the LP affect the decay rate of behaviours also worked well, as the poor demonstrators had fewer of their behaviours added to the imitator's final forward model, while the expert demonstrator had more, as its decay rate was lower, so more of its behaviours became permanent to the forward model representing it.

Using my forward models to control the imitator was somewhat disappointing, though the imitator still performed better than the poor demonstrator used in my experiments on learning from demonstrators of varying skill, and roughly approximated an average demonstrator. The preconditions and ball predictions did not guide the use of the imitator's behaviours as much as I had hoped they would. This was simply a matter of too few samples for each behaviour to calculate statistics with, as the experiments were all run on physical robots and so could not be repeated ad nauseam until a substantial amount of statistics were gathered. Coupling my implementation with a preliminary learning phase where the imitator learns in simulation may solve this problem.

5.2 Contributions

The primary contribution of this thesis is an imitation learning architecture that can learn from multiple demonstrators that vary both in physiologies and skill levels. Very little previous research in imitation learning has focused on imitating multiple demonstrators previously. The forward models I have designed for this purpose are also a contribution. I am not aware of any other research where forward models that represent individual demonstrators are used to nominate candidate behaviours for the imitator to learn during their training process.

The demonstrator-specific learning preference used by the forward models in my implementation is a contribution, as is the permanency attribute of the behaviours which decays over time. I have not encountered any other research where behaviours decayed over time, and also had the rate their behaviours were made permanent or deleted affected by the skill level of the demonstrator from which the behaviour was learned. The idea of increasing the permanency of behaviours based on how useful those behaviours are is also unique when it is weighted by the LP.

The architecture I have developed is fairly modular. The forward models used to learn the behaviours and model the demonstrators could be used with different primitive recognition and prediction techniques than those I have employed. As long as a demonstration can be converted into whatever primitives the imitator uses, my forward models could be used to learn sequential behaviours from the demonstrations.

The fact that this has been developed using real heterogeneous robots in a physical environment also makes contributions. I have noted how difficulties brought about by operation in the real world affect even the simplest aspects of evaluation, such as

comparing recognized primitives to recorded commands (Chapter 4). In an era where much evaluation of robotic techniques is evaluated in simulation, more discussion of and adaptation to such difficulties is warranted.

I have also made contributions in terms of artifacts: not only is my approach to imitation learning a contribution, it is embodied in a running computer programs that can be adapted to other application environments. Screenshots of the applications developed are shown in Appendix A. Developing these programs and evaluating their performance involved much underlying work in robotic control code, clustering algorithms and graphical techniques to display clusters, GUI design to ensure all settings could be easily manipulated and tested, and design of elaborate forward models that must work with components of various applications.

Through the development of the Hidden Markov Models used in my implementation, I have also contributed a straightforward and concise description of the equations used to implement discrete HMMs, as well as the various errata and some pitfalls encountered when developing them.

5.3 Relationship to Prior Work

Very little previous work has been done to model each demonstrator an imitator is exposed to individually for the explicit purpose of comparing demonstrator performance and weight learning towards demonstrators of higher skill. Yamaguchi et al. [1996]'s work as well as Yamaguchi et al. [1997]'s work involved learning from the teammate that is considered most skilled. The higher skill is determined by the amount of reinforcement a teammate has received, as opposed to the various meth-

ods that the LP of a demonstrator can be adjusted in my work, all of which are related to the usefulness of the individual behaviours that the demonstrator exhibits. Yamaguchi et al. [1997] only use homogenous teammates, while I have shown that my system works using demonstrators that are physically heterogeneous as well as homogenous. Their results showed that an agent can learn more if there is a skilled demonstrator, though their system reverts to reinforcement learning if no agents are considered to be more skilled. My system, in contrast, has been shown to learn from poorly skilled and average demonstrators and experts, as well as multiple demonstrators simultaneously. Moreover, it does not exclude demonstrators that are poor (one of the techniques used by Anderson et al. [2002], but allows useful elements of a poorly-skilled agent's performance to be learned if no other agent supplies them first. That is, it can take advantage of the good elements that are likely present in any demonstrator, no matter how poor.

Price and Boutilier [2003] implement imitation learning from multiple demonstrators, though they extract information about the demonstrators simultaneously and combine it without comparing their abilities as my work does. This method would not handle the case when the imitator is exposed to poor demonstrators, unlike my work where learning is weighted appropriately towards demonstrators based on their skill levels relative to the other demonstrators.

Demiris and Hayes [2002] implemented forward models in their work that can predict the effects of individual behaviours. In their work, each behaviour is assigned a forward model, as opposed to my work, where a forward model is trained to represent a given demonstrator by learning that demonstrator's behaviours. Their models were

not designed to learn from multiple demonstrators and then combine all the forward models representing the demonstrators into a final imitator model as my work does. Their work also focused more on the refinement of behaviour control, whereas my work did not. This could be an interesting way to improve the performance of my behaviour predictive capabilities in future work, by using the forward models that Demiris and Hayes [2002] use for each individual behaviour in my implementation, but still represent demonstrators using the forward models I have developed for my system.

5.4 Future Work

Future work in this area could go a number of different directions, but I think the most important thing that my system would benefit from would be more accurate predictions. From the standpoint of the learning architecture, an obvious extension would be an initial training phase that is performed in simulation to gain a large amount of data with which to gather statistics about the robot's behaviours. Additionally, it would be useful to have a phase where the imitator is placed on the field and must refine each behaviour's predictions, possibly using the forward models for each individual behaviour as done by Demiris and Hayes [2002]. This approach would allow the forward models I use to represent the demonstrators to make better predictions, which should improve the overall performance of my approach. The additional statistics gathered during simulation could aid in proper precondition calculations, as well as more accurate ball predictions.

It would also be valuable to explore the forward model architecture with more

advanced primitives, and to control more complex robots, such as humanoids instead of the two-wheeled imitator I used in my experiments. The forward models would have to be adjusted to allow for the learning of behaviours that are composed of two or more behaviours simultaneously for this to be feasible.

This system was designed to be run in real-time with minimal changes. Converting the various applications used at the various stages to one single program would be interesting to see if a real-time imitator could still adequately shape its learning towards the more skilled demonstrators.

I would also like to explore different primitive classification techniques, including Gaussian mixture models, and continuous HMMs. It would be interesting to explore the impact of various recognition techniques on the forward models in my implementation.

5.5 Summary

It is my hope that the success of my approach at learning behaviours from physiologically distinct demonstrators of varying skill levels will lead to more research in imitation learning that focuses on both heterogeneity and skill differences. Humans are constantly learning tasks from many sources of varying skill simultaneously. If robots can learn through imitating many demonstrators, as well as learning to weight their learning towards demonstrators of more skill, it would make them much easier for the average person to work with. To have robots truly pervade our lives in an unobtrusive way, we will need to give them the ability to learn from the people around them. By modelling each person individually, I believe imitating robots will be able

to filter what they learn with little to no communication from humans, ultimately making it painless to integrate them into daily life.

Bibliography

J. Anderson, B. Tanner, and R. Wegner. Peer reinforcement in homogeneous and heterogeneous multi-agent learning. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC2002)*, pages 13–18, Banff, AB, July 2002.

J. Anderson, J. Baltes, and T. Liu. Robobisons 2004. In *Proceedings of RoboCup-2004 (Team Description Papers)*, Lisbon, 2004a.

J. Anderson, B. Tanner, and J. Baltes. Reinforcement learning from teammates of varying skill in robotic soccer. In *Proceedings of the 2004 FIRA Robot World Congress*, Busan, Korea, October 2004b. FIRA.

R. C. Arkin. *Behavior-Based Robotics*. Cambridge: MIT Press, Cambridge, MA, 1998.

J. Baltes and J. Anderson. Intelligent global vision for teams of mobile robots. In S. Kolski, editor, *Mobile Robots: Perception & Navigation*, chapter 9, pages 165–186. Advanced Robotic Systems International/pro literatur Verlag, Vienna, Austria, 2007. ISBN 3-86611-283-1.

J. Baltes, J. Bagot, and J. Anderson. Humanoid robots: Storm, Rogue, and Beast. In *Proceedings of RoboCup-2008 (Team Description Papers)*, Suzhou, China, July 2008.

A. Billard. Play, Dreams and Imitation in Robota. In *Workshop on Interactive Robotics and Entertainment*, CMU, Pittsburgh, 2000. LASA.

A. Billard and K. Dautenhahn. Experiments in learning by imitation - grounding and use of communication in robotic agents. *Adaptive Behavior*, 7(3/4):415–438, 1999.

A. Billard and M. J. Matarić. A biologically inspired robotic model for learning by imitation. In *Proceedings, Autonomous Agents 2000*, pages 373–380, Barcelona, Spain, June 2000.

C. Breazeal and B. Scassellati. Challenges in building robots that imitate people. In K. Dautenhahn and C. Nehaniv, editors, *Imitation in Animals and Artifacts*, pages 363–390. MIT Press, 2002a.

C. Breazeal and B. Scassellati. Robots that imitate humans. *Trends in Cognitive Sciences*, 6(11):481–487, 2002b. ISSN 1364-6613.

R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

R. W. Byrne. Imitation without intentionality. Using string parsing to copy the organization of behaviour. *Animal Cognition*, 2(2):63–72, June 1996.

C. A. Calderon and H. Hu. Goal and actions: Learning by imitation. In *Proceed-*

ings of the AISB 03 Second International Symposium on Imitation in Animals and Artifacts, pages 179–182, Aberystwyth, Wales, 2003.

S. Calinon and A. Billard. Learning of Gestures by Imitation in a Humanoid Robot. In K. Dautenhahn and C. N. (Eds), editors, *Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions*, pages 153–177. Cambridge University Press, 2007.

S. Chernova and M. Veloso. Confidence-based learning from demonstration using Gaussian mixture models. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, AAMAS'07, 2007*, 2007.

F. Crabbe and M. Dyer. Observation and imitation: Goal sequence learning in neurally controlled construction animats: VI-MAXSON. In *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior (SAB)*, pages 373–382, 2000.

A. Dearden and Y. Demiris. Learning forward models for robots. In *Proceedings of IJCAI-05*, pages 1440–1445, Edinburgh Scotland, August 2005.

J. Demiris and G. Hayes. Imitation as a dual-route process featuring predictive and learning components: A biologically plausible computational model. In K. Dautenhahn and C. Nehaniv, editors, *Imitation in Animals and Artifacts*, pages 327–361. MIT Press, 2002.

Y. Demiris and M. Johnson. Distributed, predictive perception of actions: a biologi-

cally inspired robotics architecture for imitation and learning. *Connection Science*, 15(4):231243, December 2003.

P. F. Ferrari, V. Gallese, G. Rizzolatti, and L. Fogassi. Mirror neurons responding to the observation of ingestive and communicative mouth actions in the monkey ventral premotor cortex. *European Journal of Neuroscience*, 17(8):1703–1714, 2003.

J. A. Hartigan and M. A. Wong. Algorithm AS 136: A k-means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979.

G. Hovland, P. Sikka, and B. McCarragher. Skill acquisition from human demonstration using a hidden markov model. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, volume 3, pages 2706–2711, April 1996.

T. Inamura, I. Toshima, and Y. Nakamura. Acquiring motion elements for bidirectional computation of motion recognition and generation. In *Proceedings of the International Symposium On Experimental Robotics (ISER)*, pages 357–366, Sant'Angelo d'Ischia, Italy, July 2002.

T. Inamura, Y. Nakamura, I. Toshima, and H. Tanie. Embodied symbol emergence based on mimesis theory. *International Journal of Robotics Research*, 23(4):363–377, 2004.

O. Jenkins and M. J. Matarić. Automated derivation of behavior vocabularies for autonomous humanoid motion. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 225–232, Melbourne, Austrailia, July 2003.

O. C. Jenkins, M. J. Mataric, and S. Weber. Primitive-based movement classification for humanoid imitation. In *Proceedings, First IEEE-RAS International Conference on Humanoid Robotics*, Cambridge, MA, October 2000.

H. Kautz. A formal theory of plan recognition and its implementation. In J. F. Allen, H. A. Kautz, R. Pelavin, and J. Tenenberg, editors, *Reasoning About Plans*, pages 69–125. Morgan Kaufmann, San Mateo, CA, 1991.

Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, January 1980. ISSN 0090-6778.

G. F. Luger. *Artificial Intelligence*. Pearson Education Limited, 5 edition, 2005.

M. J. Matarić. Getting humanoids to move and imitate. *IEEE Intelligent Systems*, pages 18–24, July 2000.

M. J. Matarić. Sensory-motor primitives as a basis for imitation: linking perception to action and biology to robotics. In K. Dautenhahn and C. Nehaniv, editors, *Imitation in Animals and Artifacts*, pages 391–422. MIT Press, 2002.

M. J. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83, 1997. ISSN 0929-5593.

A. N. Meltzoff and M. K. Moore. Explaining facial imitation: A theoretical model. In *Early Development and Parenting*, volume 6, pages 179–192. John Wiley and Sons, Ltd., 1997.

A. Moore. K-means and hierarchical clustering. http://www.autonlab.org/tutorials/kmeans.html, 2004.

C. L. Nehaniv and K. Dautenhahn. Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications. In J. Demiris and A. Birk, editors, *Interdisciplinary Approaches to Robot Learning*, volume 24, pages 136–161. World Scientific Press, 2000.

M. Nicolescu and M. J. Matarić. Natural methods for robot task learning: Instructive demonstration, generalization and practice. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 241–248, Melbourne, Australia, July 2003.

B. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

L. Rabiner and B. Juang. An introduction to Hidden Markov Models. *IEEE ASSP Magazine*, pages 4–16, 1986.

A. Rahimi. An erratum for "a tutorial on Hidden Markov Models and selected applications in speech recognition". http://xenia.media.mit.edu/ rahimi/rabiner/rabiner-errata/rabiner-errata.html, 2000.

G. Rigoll, A. Kosmala, J. Rattland, and C. Neukirchen. A comparison between continuous and discrete density hidden markov models for cursive handwriting recognition. volume 2, pages 205–209, Aug 1996.

P. Riley and M. Veloso. Coaching a simulated soccer team by opponent model recognition. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 155–156, May 2001.

G. Rizzolatti, L. Fadiga, V. Gallese, and L. Fogassi. Premotor cortex and the recognition of motor actions. *Cognitive Brain Research*, 3(2):131 – 141, 1996. ISSN 0926-6410.

J. Saunders, C. Nehaniv, and K. Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of the 1st ACM/IEEE International Conference on Human-Robot Interactions, HRI'06*, 2006.

M. Stamp. A revealing introduction to Hidden Markov Models. http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf, 2004.

M. Walter, A. Psarrou, and S. Gong. Auto-clustering for unsupervised learning of atomic gesture components using minimum description length. In *Proceedings of the IEEE ICCV Workshop on Recognition, Analysis and Tracking of Faces and Gestures in Real-time Systems*, pages 157–163, Vancouver, Canada, July 2001.

S. Weber, M. J. Matarić, and O. Jenkins. Experiments in imitation using perceptuomotor primitives. In C. Sierra, M. Gini, and J. S. Rosenschein, editors, *Proceedings of the fourth international conference on Autonomous agents*, pages 136–137, Barcelona, Catalonia, Spain, June 2000. ACM Press.

T. Yamaguchi, M. Miura, and M. Yachida. Multi-agent reinforcement learning with

adaptive mimetism. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation, 1996. EFTA '96.*, volume 1, pages 288–294, 1996.

T. Yamaguchi, Y. Tanaka, and M. Yachida. Speed up reinforcement learning between two agents with adaptive mimetism. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS-97).*, volume 2, pages 594–600, 1997.

J. Yang and Y. Xu. Hidden Markov Model for control strategy learning. Technical Report CMU-RI-TR-94-11, Carnegie Mellon University, Pittsburgh, PA, May 1994.

J. Yang, Y. Xu, and C. Chen. Human action learning via Hidden Markov Model. *IEEE Transactions on System, Man, and Cybernetics*, 27(1):34–44, 1997.

Appendix A

Applications Developed

This appendix contains images of the various applications used to develop and evaluate this thesis. All applications shown were developed by me specifically for this thesis work. All applications were developed using the *Qt* platform in *Ubuntu* Linux.

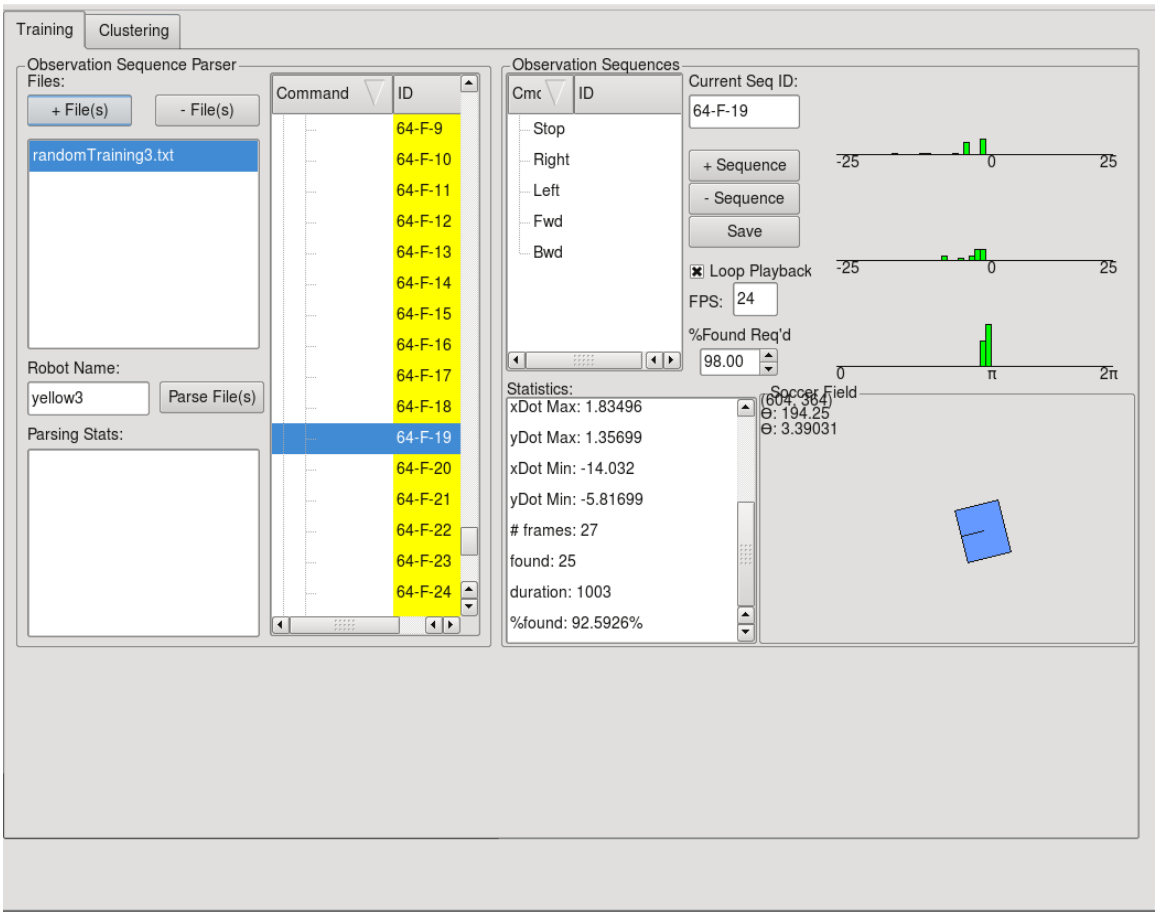

Figure A.1: The application used to sort primitive training data.

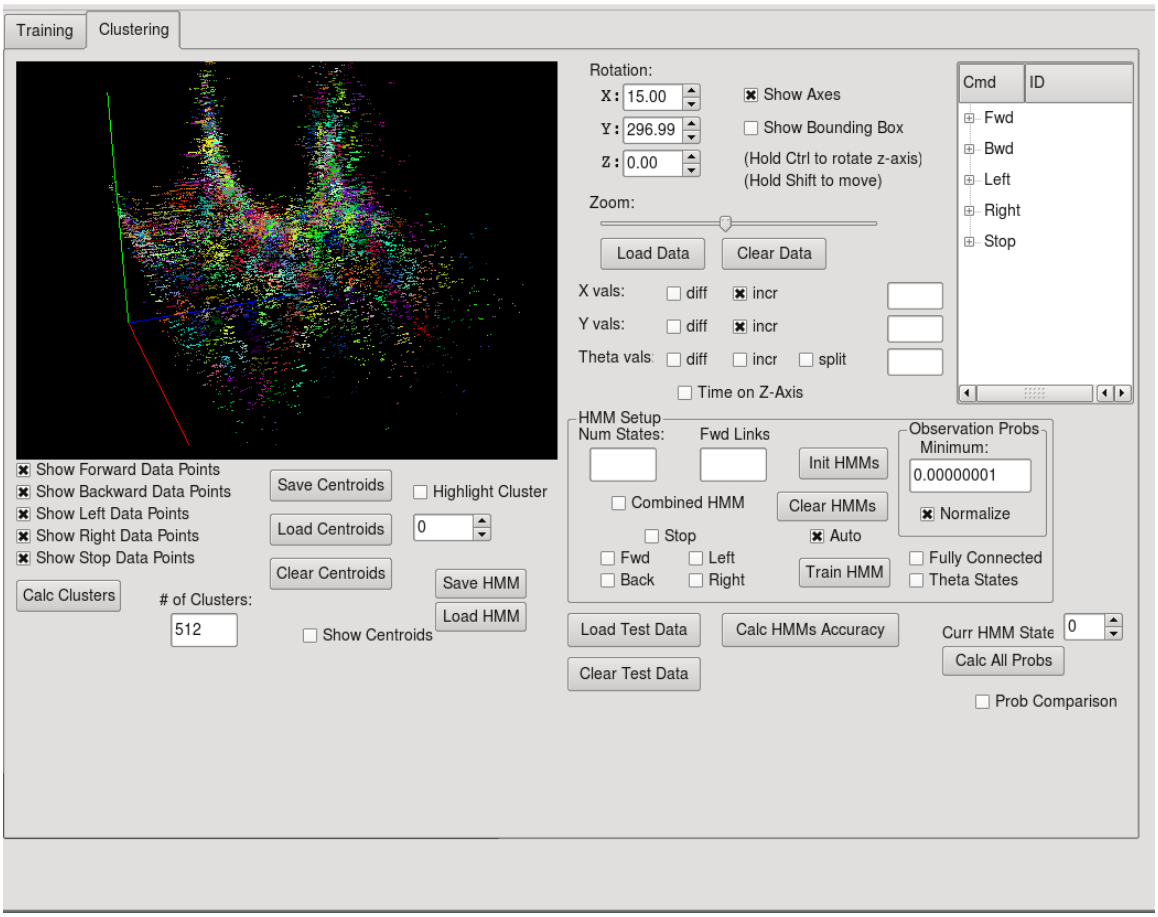

Figure A.2: The application that is used to cluster visual data points and use them to train Hidden Markov Models.

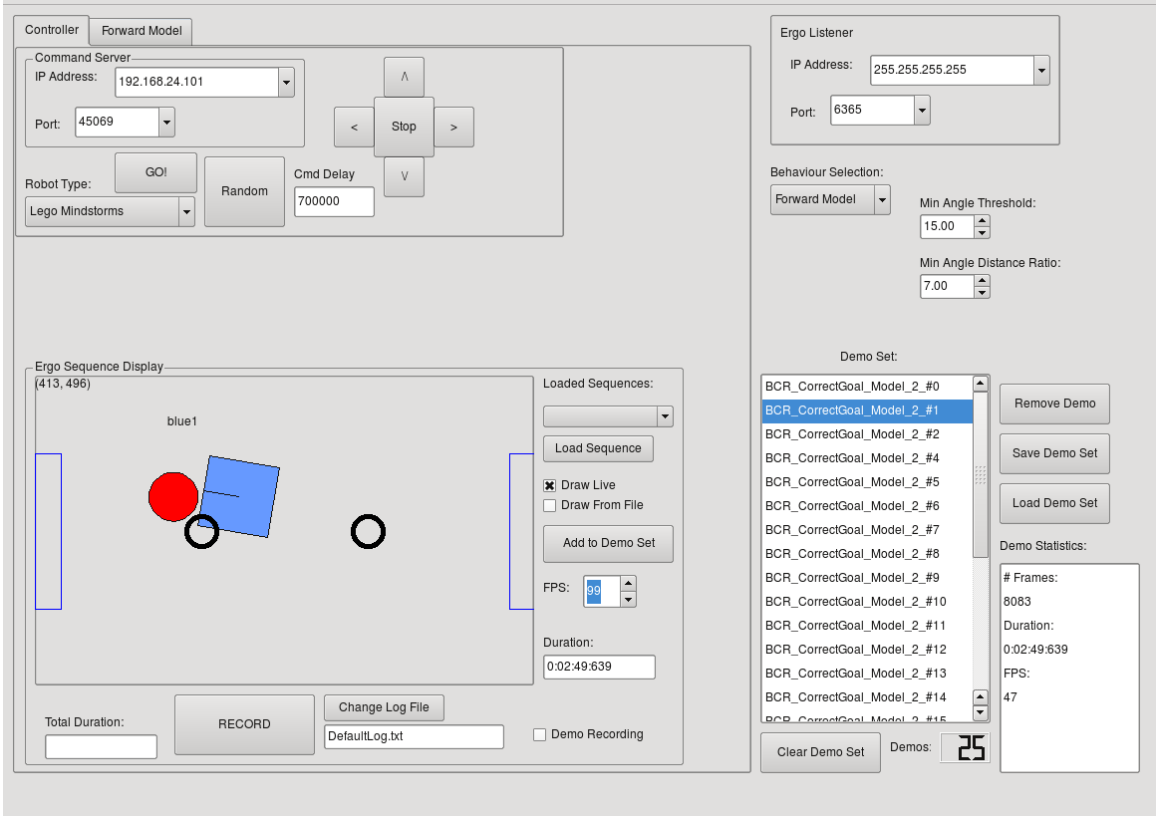

Figure A.3: The application that records all demonstrations.

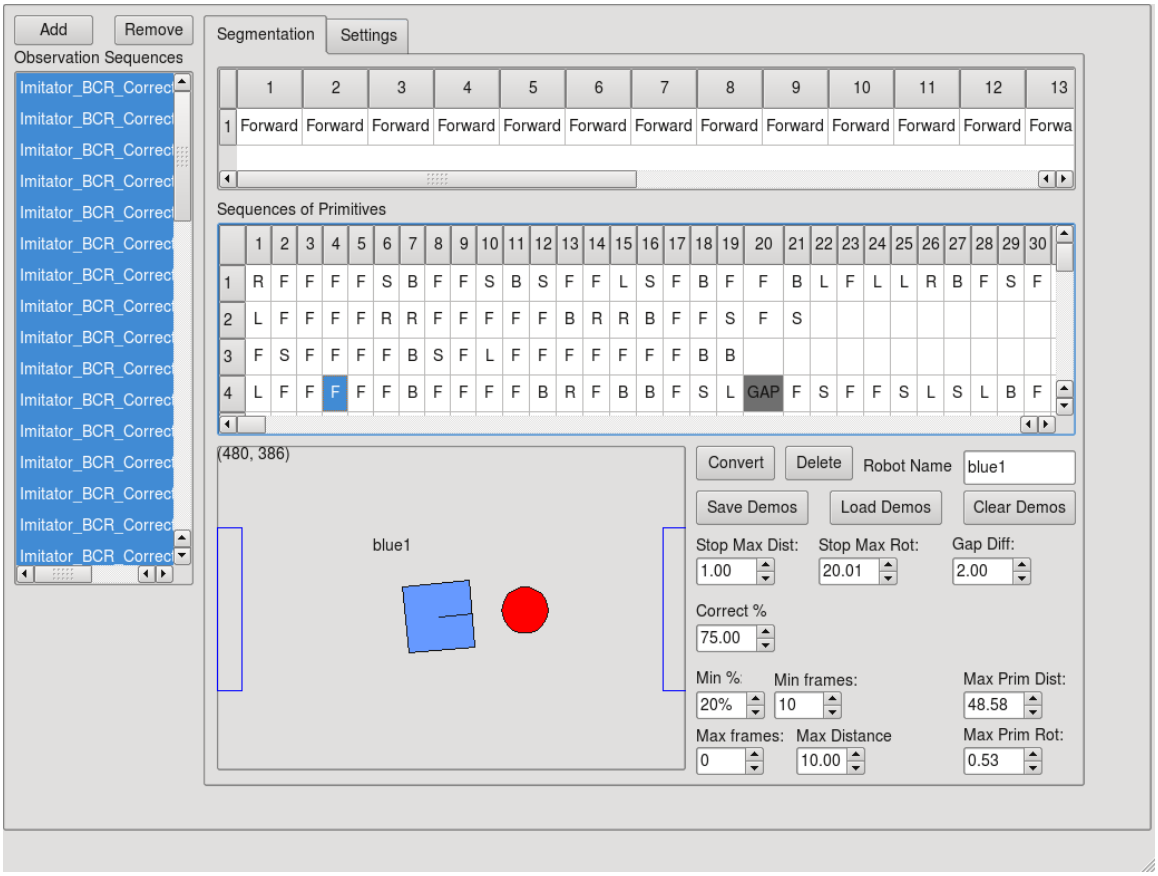

Figure A.4: The application that converts demonstrations into sequences of primitive symbols.

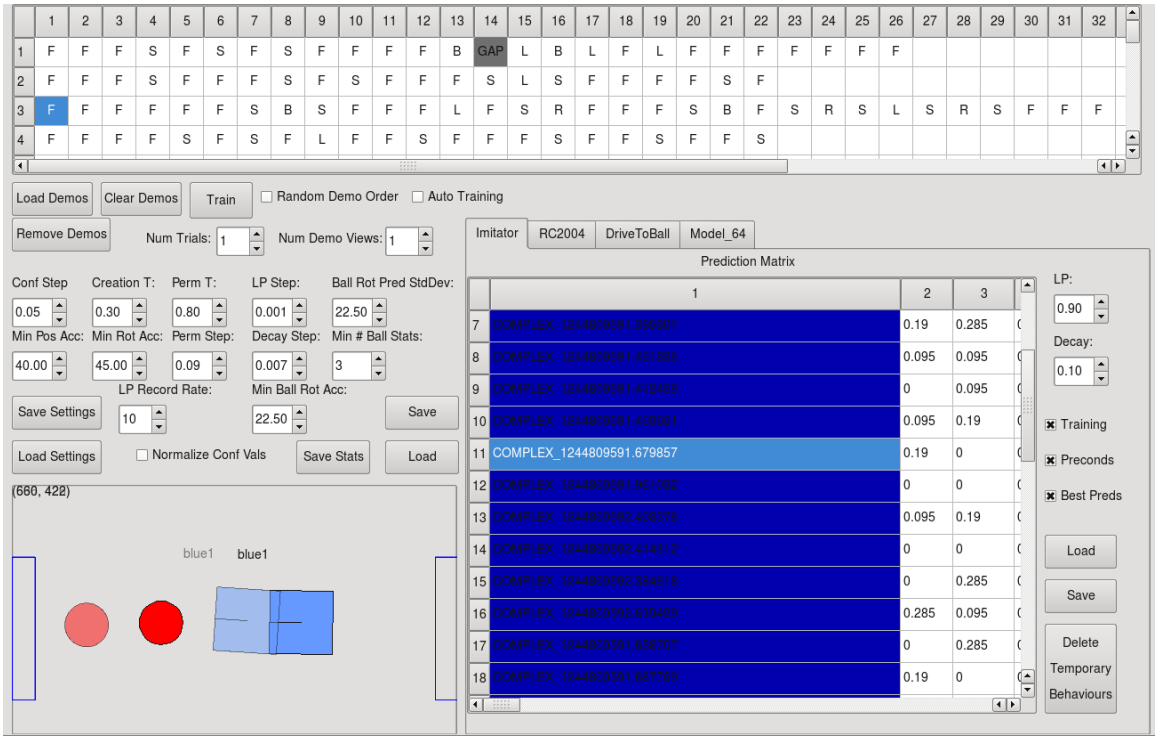

Figure A.5: The application that trains the forward models.

Appendix B

Sample of Behaviours Created

This appendix contains a sample of the behaviours learned by my system. A full listing is unreasonable as there were 2368 unique behaviours created across all models and all trials of my experiments. Each table in this section was compiled from 1200 forward models (3 models per trial, 100 trials per set of experiments, and 4 forward models for each trial). The counts shown for each behaviour are the total number of times it appeared across all the models. The same is true for the number of times a behaviour was made permanent, as well as the number of times a behaviour had the ability to predicted the ball. As a result, the maximum value is 1200, meaning every single model had that behaviour (or made it permanent, etc.).

Count	**#Made Permanent**	**# Predicting Ball**	**Primitive List**
1200	1200	73	F,F
1200	1200	0	B,B
1200	1200	65	F,F,F
1199	1199	76	F,F,F,F
1187	1165	0	B,B,B
1182	1106	74	F,F,F,F,F
1175	1165	5	L,B
1168	1062	69	F,F,F,F,F,F
1139	1100	24	F,L
1107	1002	0	L,L
1103	1024	16	R,F
1091	1000	7	R,B
1079	923	21	L,F
1058	929	2	B,B,B,B
1015	859	0	R,R
1005	489	0	B,L
975	577	1	B,R
970	477	22	F,F,F,F,F,F,F
965	206	5	B,F
964	683	7	F,F,L
952	840	30	F,L,B
949	765	20	F,R,F
941	729	6	F,L,F
924	524	1	B,L,B
915	823	0	L,L,L,L
878	53	2	F,B
872	409	1	B,B,B,B,B
868	639	75	F,R
837	656	2	L,B,B
816	571	23	L,F,F

Table B.1: The 30 most commonly created behaviours from all forward models trained during experimentation.

Count	#Made Permanent	# Predicting Ball	Primitive List
1200	1200	73	F,F
1200	1200	0	B,B
1200	1200	65	F,F,F
1199	1199	76	F,F,F,F
1187	1165	0	B,B,B
1175	1165	5	L,B
1182	1106	74	F,F,F,F,F
1139	1100	24	F,L
1168	1062	69	F,F,F,F,F,F
1103	1024	16	R,F
1107	1002	0	L,L
1091	1000	7	R,B
1058	929	2	B,B,B,B
1079	923	21	L,F
1015	859	0	R,R
952	840	30	F,L,B
915	823	0	L,L,L,L
949	765	20	F,R,F
941	729	6	F,L,F
964	683	7	F,F,L
837	656	2	L,B,B
868	639	75	F,R
731	632	85	F,R,B
725	621	0	L,L,L
724	609	0	L,L,L,L,L,L
700	580	0	R,B,B
975	577	1	B,R
816	571	23	L,F,F
756	558	11	L,B,B,B
753	554	0	L,L,B

Table B.2: The 30 behaviours most commonly made permanent to the various forward models trained during experimentation.

Count	**#Made Permanent**	**# Predicting Ball**	**Primitive List**
502	390	95	F,F,R,B
731	632	85	F,R,B
1199	1199	76	F,F,F,F
868	639	75	F,R
1182	1106	74	F,F,F,F,F
1200	1200	73	F,F
1168	1062	69	F,F,F,F,F,F
1200	1200	65	F,F,F
805	314	57	B,R,F
186	83	37	F,F,F,R,B
629	405	32	R,F,F
952	840	30	F,L,B
137	57	26	L,F,R
186	90	25	F,F,R,B,B
1139	1100	24	F,L
816	571	23	L,F,F
970	477	22	F,F,F,F,F,F,F
1079	923	21	L,F
949	765	20	F,R,F
741	192	18	F,F,F,F,F,F,F,F
1103	1024	16	R,F
480	136	15	F,F,R
479	66	15	L,B,R,F
756	558	11	L,B,B,B
383	173	11	R,F,L
397	154	10	F,R,B,B
114	76	10	R,F,F,L,B
95	65	10	B,R,F,F
595	181	9	F,F,R,F
34	18	9	F,F,L,L,L,L,F,L,L,L,L,L,L

Table B.3: The 30 behaviours that most commonly learned to predict the ball from all forward models trained during experimentation.

Count	Primitive List
1	L,L,F,L,B,L,F,L,B,L,F,L,B,L,F,L,B,L
1	L,F,L,B,L,F,L,B,L,L,F,L,B,L,F,L,B,L
1	L,F,L,B,L,F,L,B,F,B,L,F,L,B,L,F,L,B
1	L,F,L,B,L,F,L,B,F,L,F,L,B,L,F,L,B,L
1	L,F,L,B,L,F,L,F,L,B,L,L,F,L,B,L,L,L
1	L,L,F,L,B,L,L,F,L,B,L,F,L,B,L,F,L,B
1	L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L
1	R,R,R,R,R,R,R,R,F,L,L,R,R,R,R,R,R,R,R
1	L,L,L,L,L,L,L,F,L,L,L,L,L,L,L,B,B,B
1	B,L,L,L,L,L,L,F,L,L,L,L,L,L,L,B,B,B
1	L,L,L,L,F,L,L,L,L,L,L,L,L,L,L,L,L,L
1	L,L,L,L,L,L,L,L,F,L,L,L,L,L,L,L,L,F
1	L,F,R,R,R,R,R,R,R,R,F,R,R,R,R,R,R,R,F
1	L,F,L,B,L,L,F,L,B,L,L,F,L,B,L,L,F,L,B,L
1	L,L,L,F,L,B,L,F,L,F,L,B,L,L,F,L,B,L,L,L
1	L,L,F,L,B,L,L,F,L,B,F,L,L,B,L,L,L,F,L,B
1	F,L,B,L,L,F,L,B,L,L,F,L,B,L,L,F,L,B,L,L
1	L,F,R,R,R,R,R,R,R,R,F,L,L,R,R,R,R,R,R,R
1	L,F,L,B,L,F,L,B,L,L,F,L,F,L,B,L,F,L,B,L,L,F
1	L,L,L,L,L,L,L,L,L,L,B,B,L,L,L,L,L,L,L,L,L,B,B
1	L,L,L,L,F,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,L
1	F,L,L,F,L,L,L,L,L,L,L,L,L,L,F,L,L,L,L,L,L,L,L
1	L,L,L,L,L,L,L,L,F,L,L,L,F,L,L,L,L,L,L,L,F,L,L,L
1	F,L,L,L,L,L,L,L,L,F,L,L,L,F,L,L,L,L,L,L,L,F,L,L,L
1	L,L,L,L,F,L,L,L,L,L,L,L,L,F,L,L,L,L,F,L,L,L,L,L,L,L
1	F,L,L,L,L,L,L,L,L,L,L,F,L,L,L,L,L,L,L,L,L,L,L,F,L,L
1	L,L,L,L,L,L,L,L,L,F,L,L,L,L,F,L,L,L,L,F,L,L,L,L,L,L,L
1	L,L,L,L,L,L,L,L,L,F,L,L,L,L,L,L,L,L,L,L,L,L,F,L,L,L,L,L
1	F,F,L,L,L,L,F,L,L,L,L,L,L,L,L,F,L,L,L,L,F,L,L,L,L,L,L,L,L
1	L,L,L,L,L,L,L,L,L,L,F,L,L,L,L,L,L,L,L,L,L,L,L,F,L,L,L,L,F

Table B.4: The 30 longest behaviours from all forward models trained during experimentation. Note that none of these behaviours achieved permanency or had the ability to predict the ball's motion across all models.