Learning from Observation Using Primitives

A Thesis
Presented to
The Academic Faculty

by

Darrin C. Bentivegna

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
July 2004

Learning from Observation Using Primitives

Approved by:

Christopher G. Atkeson, Adviser

Jessica Hodgins

Tucker Balch

Sven Koenig

Gordon Cheng

Date Approved: 30 June 2004

I can do everything through him who gives me strength.
- Philippians 4:13 (NIV) [96]

To Him.

ACKNOWLEDGEMENTS

This thesis would not be possible without the Lord's constant guidance and encouragement. The Lord also works through people and He has blessed me with very special people to fulfill all my needs during this time. I met Christopher Atkeson during my first week at GA Tech and he has encouraged me since that very first meeting to explore and learn all that is possible. Chris has had a tremendous amount of patience and understanding during the process of transforming me into a researcher. The freedom and support he provided for me allowed me to explore many related research fields and interact with researchers from around the world. During this time Chris has become much more than just an advisor to me.

I thank the other members of my research committee for their feedback and guidance and for being extremely flexible in this unique situation of coordinating between people located in four different locations around the world. I thank Ronald Arkin and Gregory Abowd for their guidance during my first years at GA Tech. I thank Dr. Mitsuo Kawato for providing me with support and a great environment in which I could interact with many senior researchers and make use of advanced robotic research platforms.

My wife, Akiko, ensured that my education expanded beyond my thesis research and during this time together we learned more about the Lord's ways than we ever thought possible. I thank my parents for supporting me in all my adventures and for providing me with the consistency of a loving home in which to return to.

The Lord has truly blessed me by giving me a desire to learn and laying out a path full of wonderful people to support me.

Financial support for this research was provided in part by ATR Computational Neuroscience Laboratories, Department of Humanoid Robotics and the National Institute of Information and Communications Technology (NiCT), Kyoto, Japan. It was also supported in part by the Japan Science and Technology Agency, ICORP, Computational Brain Project, Kyoto, Japan, and by the National Science Foundation Award ECS-0325383.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

For God so loved the world that he gave his one and only Son, that whoever believes in him shall not perish but have eternal life.
- John 3:16 (NIV) [96]

This thesis presents a method for using observation data and the knowledge of primitives to increase the learning rate of robots that operate in large and continuous environment state spaces in simulation and in the real world. The research presented in this thesis addresses many of the difficulties/challenges that arise when implementing real-world systems that learn from observation and practice using primitives. How to represent the learned information, what primitive should be performed in a given situation, and how to perform the primitive are examples of the challenges that are addressed in this thesis. At least one possible solution is provided for each of the difficulties presented and various alternative solutions are also discussed. The complexity of the solutions range from the simple solution of having the human specify what is needed to the more complex solution of having the robot learn about the information while it operates in the environment.

A significant challenge for robots operating in dynamic environments is choosing subgoals that lead to the overall task objective. Within this research, the robots initially learn about subgoal information using a combination of specified and observed information. But because the environment space is large and continuous, it may not be possible to learn all the information needed from only a few observations. Therefore the robots will need to have the ability to also learn what are good and bad subgoals as they operate in the environment. This thesis presents a novel algorithm that combines locally weighted learning and reinforcement learning techniques that allow the robot to change its action selection and subgoal generation behavior as it performs the task.

Virtual and hardware environments of air hockey, Figure 1, and the marble maze game, Figure 2, have been designed as testbeds in which to conduct this research. The implementations of our learning system highlight the challenges of creating robotic systems that will learn from observation and practice. The air hockey and marble maze domains have been chosen for a variety of reasons.

Figure 1: The software air hockey game on the left and playing air hockey with a humanoid robot on the right.

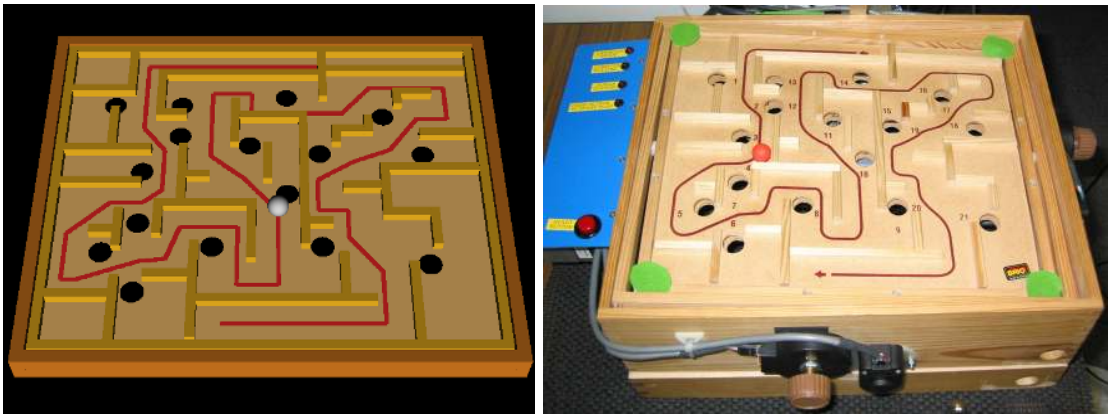

Figure 2: The virtual marble maze game on the left modeled after the hardware version on the right. The maze on the right has motors and sensors to control the board tilt angles and a vision system to observe the position of the marble.

First, virtual implementations of these domains could be created that allow a human player to operate in the environment using a mouse. These virtual implementations provide a useful environment for initial testing. Second, hardware implementations that would allow a robot to operate in the environment can be created in many ways that will fit within a lab environment. Additionally, because the playing areas are two dimensional, sensing and moving is simplified.

The domains also provide different conditions under which to evaluate this research. Air hockey is a very dynamic environment where the puck is almost always moving. It is fast, requiring rapid perception, thinking, and movements. It is demanding, requiring considerable movement accuracy. Perceptual and movement time delays, board placement and leveling, varying board surface conditions and friction, air flow, temperature effects, and actuator dynamics all make air hockey a complex domain. There are disturbances, modeling errors, and an adversary, so there is much to learn. Air hockey is closely related to racquet sports such as ping-pong and tennis, as well as other interactive games such as playing catch. We believe that our approach to air hockey can generalize to a wide range of intermittent dynamic tasks.

Marble maze, on the other hand, is a single player game and is therefore fairly constant from game to game. Once a robust policy has been created, it can always be used to solve that maze. The robot in the marble maze can also practice the task by itself without the need for an opponent. Within the maze environment there is also the concept of a local environment state space representation. This may be used to provide a robot with the ability to play in mazes that it has not previously seen.

The air hockey task is also being explored by Spong and colleagues [23, 97, 99]. Bishop et al. discuss difficulties, such as table position errors and specifying a puck movement model, that our learning and adaptation models try to overcome. Whereas the research of Spong et al. [23] provides insight into the physical interactions of air hockey, our research seeks methods for humanoid robots to quickly learn task strategies and environment models through observation and practice. The marble maze task is similar to parts orientation using tray tilting [39].

1.1 Primitives

Robots typically must generate commands to all their actuators at regular intervals. The analog controllers for our 30-degree of freedom humanoid robot are given desired torques for each joint at

420Hz. Thus, a task with a one second duration is parameterized with $30 * 420 = 12600$ parameters. Learning in this high dimensional space can be quite slow or can fail totally. Random search in such a space is not possible. Because robot movements take place in real time, learning approaches that require more than hundreds of practice movements are often not feasible. Special purpose techniques have been developed to deal with this problem, such as trajectory learning [6], learning from observation [11, 12, 48, 70, 13, 37, 50, 56], postural primitives [132], and other techniques that decompose complex tasks or movements into smaller parts [7, 20, 81].

This research explores the use of primitives to reduce the dimensionality of the learning problem [7, 116]. Primitives are units of behavior above the level of motor or muscle commands. There have been many proposals for such units of behavior in neuroscience, psychology, robotics, artificial intelligence, and machine learning [7, 116, 107, 18]. There is a great deal of evidence that biological systems have units of behavior above the level of activating individual motor neurons, and that the organization of the brain reflects those units of behavior. We know that in human eye movement, for example, there are only a few types of movements including saccades, smooth pursuit, VOR, OKN, and vergence, that more complex eye movements are generated as sequences of these behavioral units, and that there are distinct brain regions dedicated to generating and controlling each type of eye movement. We know that there are discrete locomotion patterns, or gaits, for animals with legs. Whether there are corresponding units of behavior for upper limb movement in humans and other primates is not yet clear. There is evidence from neuroscience and brain imaging that there are distinct areas of the brain for different types of movements. Developing a computational theory that explains the roles of primitives in generating behavior and learning is an important step towards understanding how biological systems generate behavior and learn. This thesis represents a small step towards that goal by presenting implementations that must handle many of the challenges involved when using primitives to perform a task.

1.2 Challenges with Using Primitives

This section outlines many of the challenges/difficulties involved with using primitives in systems that learn from observation and practice and the research articles referenced in this section provide examples of the methods by which the challenges are being handled. These challenges are further

explored in Chapter 2, Related Research.

A task that is to be performed using primitives must first be decomposed into a set of primitives that includes all the actions needed to perform the task. This challenge can be dealt with by having a human task expert define the set of primitives [108] or have the robot discover primitives automatically after observing a performance of the task [43] or operating in the task environment [84]. Once a set of primitives is defined, the robot must have a way to learn how to perform them. Some research deals with this challenge by explicitly programming the primitive performance policy into the robot [27] or having the robot learn the policy using learning techniques [1]. Information obtained while observing an expert performer has been used to reduce the time needed by a robot to learn the performance of a primitive [115]. Given a set of primitives and a task, the robot must decide which primitive to perform at any given time. This selection process has been accomplished by a human specifying the sequence of primitive types to be performed [81], using a planning system [127], or having the robot learn the sequence from observed data [70]. Choosing a primitive from among a small set sounds like a simple procedure. But almost all primitives have parameters such as speed of execution and desired ending state [134]. These parameters can also have continuous values and therefore can be difficult to select or learn through trial and error [74]. These low-level challenges are part of a higher level question about the amount of prior knowledge that should be provided to the robot [108].

The advantages of using primitives includes the ability for them to be used multiple times while performing a task and to also use primitives learned in one task in the performance of similar tasks. Therefore another challenge is how to represent primitives in a general way so that they may be used many times in the designed task and also be able to be used in other tasks [36].

Learning can occur at many levels when learning through practice using primitives. Among the items that can be learned while operating in the task environment are the primitive type which should be performed [14], the parameters to use with the chosen primitive [100], and the primitive execution policy [76]. The related research shows many methods that can be used to learn each of these items and a challenge is how to choose a method that works best to learn this information. If the robot learns all the information simultaneously while operating in the environment it may be that as its primitive policy improves, the parameters that were learned no longer lead to success

and therefore new parameters need to be learned. Due to the increase in skill of performing this primitive, it may now be more appropriate to use it in a situation where a different primitive type is being used. Therefore the robot must have a method to know it should try this more skillful primitive type even though it may have failed in the past [108].

When using observed data to learn a task other challenges are introduced. From all the information that the robot is presented with, it must choose what is relevant for learning the task [61]. The observed data must also be segmented into primitives if it is to learn such things as primitive sequence performance, primitive execution policy, or primitive parameters from the observed data. Segmenting and learning relevant features to observe can be accomplished in many ways. Explicitly providing only the pre-segmented information [33], specifying conditions that represent segmentation points [87, 68], and using hidden Markov models [52] are some of the methods that have been explored in the literature. The job of the observation processing algorithms is further hampered due to the variability in which a person performs the actions.

1.3 Strategy for Primitive Use

Figure 3 shows the framework in which this research has been performed. The framework is briefly presented in this section and the details of the algorithms used within each of the modules are fully explained in Chapters 4, 5, and 6. The algorithms presented in this thesis are just one possible set that fulfill the requirements of the modules within the framework. One of our main concerns when choosing the algorithms was the ability of each of the algorithms to learn from observation data and also learn through practice. Robots that use this framework are first given the ability to observe the task.

1.3.1 Perceiving the Primitive

In the research presented in this thesis, a human, using domain knowledge, designs the candidate primitives that are to be explored. The primitive recognition module uses the domain knowledge to segment the observed behavior into the chosen primitives. This module extracts the context or state in which the human has performed each primitive and formats the segmented data so that it can be used as training data for the primitive selection, subgoal generation, and action generation modules.

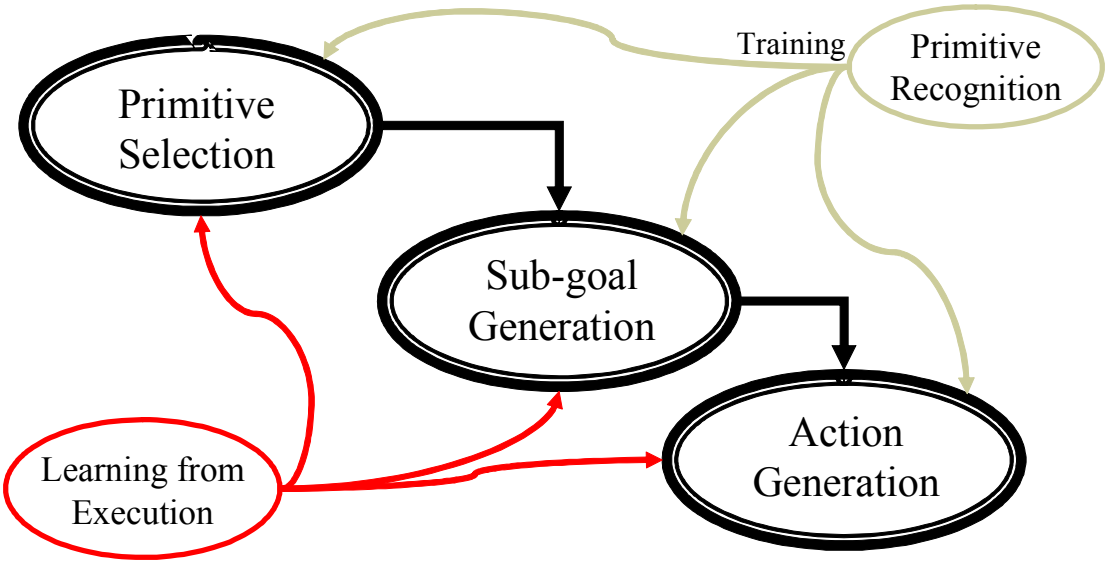

Figure 3: Framework for conducting research in learning from observation using primitives.

In our implementation, the observed primitive executions are stored in a way that encodes the state of the environment at the time the primitive was performed, the primitive type that was performed, and the outcome of performing that primitive.

1.3.2 Choosing a Primitive to Use and Associated Parameters

The primitive selection module chooses the primitive type based on the current state and prior observations of primitives that have been executed. The desired outcome, or goal, of the primitive to be performed is specified by the subgoal generation module. This module also uses the formatted data from the primitive recognition module to compute the subgoal.

1.3.3 Performing the Primitive

The action generation module provides the actuator commands needed to execute the chosen primitive type to move the environment to the computed subgoal state. There is a different action generation module for each primitive type. The policy used within the action generation module can be local and therefore can be used at multiple locations within the task domain and, if properly formatted, can also be used in similar domains.

1.3.4 Learning From Practice

After the robot has obtained initial training from observing human performance, it should then increase its skill at the task through practice. Up to this point the robot's only high-level goal is to perform like the teacher and it has not received any feedback on its performance. Its only encoding of the goal is in the implicit encoding in the observed primitives performed. The learning from execution module contains the information needed to evaluate the performance of each of the modules toward obtaining a high-level task objective. This information is used to update the modules and improve performance.

1.3.5 Learning From Observation Versus Coaching

In this research the player being observed is referred to by various names such as the observed human or teacher. This research focuses on learning from normal task performances and it is not necessarily the observed player's responsibility to perform in a way that will make it easy for an observer to learn the task from their performance. Therefore the player being observed may not perform any differently because they are being observed. In other learning from observation research, such as coaching, the teacher may perform in particular ways to provide feedback or to stress important characteristics of the task to the student [105, 93].

1.4 Locally Weighted Learning

This research makes extensive use of locally weighted learning (LWL) techniques as described in [10]. We have found that these techniques work very well with observed information. With these techniques new data can be added easily and the new information is available for use immediately without having to train the model on the new information off-line. This research also uses more advanced locally weighted learning techniques such as Locally Weighted Projection Regression (LWPR) [129], to represent learned models. With most locally weighted learning methods each data point added to the model increases the time needed to compute a solution, but LWPR maintains a reasonably constant lookup time so data may continuously be added. LWPR is a nonparametric local learning system that uses locally linear models, spanned by a small number of univariate regressions in selected directions in the input space. LWPR has proved its usefulness in such tasks

as inverse-dynamics learning [113] and inverse kinematics learning [130].

1.5 Thesis Layout and Terminology

After related research is presented in Chapter 2, the testbeds are fully described in Chapter 3. How the information is retrieved from observing the task and formatted for use in the "Learning from Observation using Primitives" framework is shown in Chapter 4. Chapter 5 shows how that information is used by the framework to give robots the ability to operate in the environment. The results of robots that have learned to perform using only the observed information are shown. Chapter 6 presents methods that the robots use to improve performance through practice. By end of Chapter 6 the reader should have a firm understanding of the framework details and challenges that the framework must deal with. A discussion of the framework details is presented in Chapter 7 along with a comparison of our learning method with a direct reinforcement learning approach. A summary and future directions are given in Chapter 8.

Within this thesis the term goal and subgoal are used many times and sometimes interchangeably. The term goal refers to the desired outcome of performing a task. For example, the goal of the marble maze task is to have the marble at the end location. But the term goal can also refer to the desired outcome of performing a single action. For example, the goal of tilting the marble maze board may be to move the marble to a corner location. Therefore the goal of the marble maze task consists of many smaller goals such as moving the marble to a position closer the end location. Within this thesis, these smaller goals are referred to as subgoals when we want to clearly distinguish them from the goal of the task. It is assumed that when the goal of the task has been obtained, the task ends or the environment is reset to a predetermined configuration before the task resumes. But when a subgoal has been completed, there will be a new subgoal to work on that will lead to the task goal.

Because this research is being performed on hardware and in simulation, the term "robot" refers to the set of algorithms that control both the robots in the real-world and the robots in the simulation environments.

CHAPTER II

RELATED RESEARCH

"Dear friends, since God so loved us, we also ought to love one another."
- 1 John 4:11 (NIV) [96]

Primitives are used in many fields. A math operator in MATLAB [82] and a grasp used by a robot hand to manipulate an object [85, 90] are examples of primitives. This section does not seek to present primitives used in all possible fields but instead discusses dynamic primitives used in the research of robotics, graphics, and reinforcement learning. Research in these areas seek to use primitives to make the overall task easier to learn or perform. The related research presented in this section shows methods that have been used to deal with some of the challenges that are introduced when using primitives in various domains.

2.1 An Early Use of Primitives in Robotics

In 1969 the Stanford Research Institute created a robot called Shakey to investigate the use of logic-based problem solving [94]. Shakey's environment consisted of various rooms and boxes of different sizes and colors. Shakey was given the task to find a box of a given size, shape and color and move the box to a designated position. Shakey used STRIPS to create a plan to perform the task [42]. The actions that Shakey can take are described by the operators (primitives) of the STRIPS system. **Go to an object** and **push open a door** are examples of operators. The operators have preconditions and expected post conditions. For example a precondition to **push open a door** would be to be near the door and the expected post condition would be that the door is open. An improvement to the basic STRIPS system was the addition of MACROPS that provided a way to generalize a sequence of primitives so that parts of the plan could be reused [41]. Sacerdoti created a hierarchical framework for STRIPS called ABSTRIPS [110]. His goal was to decrease the search space by ignoring the details in the first attempt at a solution. Criticality values were assigned to the details and determined which details were ignored at each level. After a plan was created at

the top level, some details were added iteratively and a search found primitives to fill in the those details.

2.2 The Use of Primitives in Robot Assembly Tasks

One of the major costs of assembly robots is programming the robot. In an attempt to reduce that cost, methods such as direct teaching [8] and robot-level textual programming [47] have been explored. Although the assembly environment is generally highly structured, inaccuracies and changes can cause errors. The increasing availability of sophisticated sensor systems has allowed for the development of robots that can adapt to such inaccuracies.

Many frameworks have been created that break a task up into subtasks [89, 119, 126]. These frameworks seek to map the robot actions to a symbolic level that is more intuitive and therefore makes it easier for the human to instruct the robot. Within these frameworks a task can be specified by a list of subtasks, also called task-level primitives. Once a subtask is learned by the robot, it can be reused. The programming is performed at the task level and the low level actions of the robot, once programmed or learned, can be ignored. Instructing a robot using these frameworks can be performed by task experts with little training in robot programming. The use of vision and force-based sensorimotor primitives has been demonstrated in assembly tasks [91]. The sequence of skills to perform in a task and the skills themselves are defined and programmed by a human. It is also possible to use primitives in a planning system and have the robot automatically plan the primitive execution sequence to complete the task [125].

2.3 Using Observations

Humans can usually increase their skill after observing an expert perform a task. A flexible method for robot programing would be to have the robot learn from observation in a similar way. Data collected from observing a task expert can be used to decrease the learning time of the robot by providing knowledge of how to perform a primitive and the sequence of primitives needed. Robots learning how to perform primitives using observation are typically presented with data that is observed only during the execution of the primitive. In other words, a human segments the data before it is used for learning. Systems that learn primitives and primitive execution sequences from

continuous observation data must have the ability to automatically segment the observed data into primitives [71, 68].

2.3.1 Using Observed Information to Learn Primitive Execution Policies

The robot could be guided through the task with a teach pendant, joystick or dataglove [127], or observe the task using a vision system [21]. When learning an execution policy from the demonstration the robot is not just learning a trajectory that it will play back, but is learning how to map its sensor information to appropriate actions [11]. Delson and West [33] present an approach that allows a robot to learn an acceptable range of robot force and motion trajectories from human demonstrations using a teaching gripper. The human guides the robot through the task a number of times while introducing variations in the work space. In the case of unconstrained movement in the presence of stationary objects, the region within the observed trajectories is considered to be obstacle free and a smooth trajectory within this range is created [32].

The research of Kaiser et al. [62] addresses various issues in learning a single primitive execution policy from a human demonstration. They present methods to help identify the relevant perception and action components and a method to remove incorrect motions from the observed data. Their research also addresses the need for the robot to learn from practice after the initial training phase and discusses the use of radial-basis functions and reinforcement learning as a framework in which to achieve this [122]. Their approach was demonstrated on a robot that learned a **peg insertion** and an **opening door** primitive from data collected when a human guided the robot.

2.3.2 Using Observation Information to Learn a Primitive Execution Sequence

Once a robot has acquired a set of primitives, it can also use the observed information to learn the sequence of primitive types and parameters needed to complete a task. Kang [66] showed that it is possible for a robot to use information collected using a light-stripe rangefinder and a CyberGlove with a Polhemus device to produce a subtask description of a manipulation task and then use that description to execute the task. Kang [67] used a grasp abstraction hierarchy as a structure to represent and identify grasp configurations. Kuniyoshi et al. [70] present a system that learns task performance information using only a vision system in a block assembly task.

2.4 Mobile Robots

Assembly robots are mostly configured to be stationary and another class of robots that can be considered is robots that have the ability to move around autonomously in the environment. These robots, called mobile robots, may have special sensors or manipulators on them that allow them to interact with the environment. Robots with these abilities are quickly proving their usefulness. The Mars Pathfinder [86] and the Mars rovers Spirit and Opportunity [78] successfully navigated and conducted scientific samplings on the Martian surface where, due to the communications time delay, it was difficult to teleoperate. The use of these robots in rescue missions during disasters is being explored [92, 31] and Blackburn et al. [24] report on the lessons learned from deploying a mobile robot in the search and rescue efforts following the 11 September 2001 terrorist attack upon the World Trade Center in New York City.

Mobile robots obtain information from the environment through the use of their sensors. Just like other autonomous systems, they must map the sensor reading to an action. Given that the robot may contain many sensors and that the sensor data may be continuous, deciding on an action to take for every possible combination of sensor readings is difficult. Providing robots with primitives greatly aids in the programming of the robot because it limits the possible state-action pairs that need to be considered and provides a higher-level description of the task that needs to be performed [120].

2.4.1 Mobile Robot Architectures that Use Primitives

In the motor schema architecture, robot primitives are referred to as motor schema behaviors [7]. A behavior maps significant sensor readings to motor actions for a specific function. Motor schemas run in parallel and output a vector that represents the direction and speed command for the robot. The motor schema outputs are combined to produce the motor command and can have parameters that affect its operation. An **avoid obstacle** behavior, for example, contains a parameter that controls the sphere of influence of a sensed object and a gain parameter that controls the output vector size. By varying these parameters, the behavior a robot exhibits to sensed obstacles varies. In this architecture there are also perceptual schemas that provide high level specific environment information modeled as action-oriented perception [3, 16, 131].

Another widely used mobile robot architecture is the subsumption architecture in which robot primitives, called behaviors, are organized in layers as augmented finite state machines (AFSM) [27]. Each behavior has a small task and its own policy to map sensor readings to actions. An AFSM is given only the sensor information that is needed to perform its function. The behaviors are organized in layers and have the ability to suppress sensor readings to, and inhibit motor commands from, other AFSMs. Unlike the motor schema architecture in which active schema outputs are combined, only one behavior has control of the robot at any instant in the subsumption architecture. Neigher architectures contains a world model. Genghis is a six legged robot which walks using the subsumption architecture [28]. Various other robots that have been programmed using the subsumption architecture [26].

2.4.2 Learning Behavior

The parameters in motor schemas increase their flexibility. The parameter setting of the **avoid obstacle** discussed above, for example, might determine if the robot can squeeze through two objects or must find a way around them. Due to the wide variety of possible environments the robot may operate in and the fact that there may be a large number of parameters which are continuous and may affect each other, selecting a desirable set of parameters for a given environment is not an easy task. If the environment is not fully known, the parameters chosen can only be a guess for what is needed for the robot to succeed in the environment. Robots with the ability to adjust and learn these parameters can adapt to variations in the environment. One way for the robot to change parameters is for it to continually evaluate its progress and make parameter adjustments based on its self evaluation [30]. This enhancement creates more parameters, in the form of adjustment parameters, which control the amount of change that should be made for an evaluation result. Research is being conducted to store these parameters that will be recalled when the environment matches the situation [103, 74]. Ongoing research is exploring this method on hardware robots as a fully integrated component [73]. Genetic algorithms have also been used to learn parameters for a specified environment to give robots an ”ecological niche” for similar environments [100].

The motor schema architecture offers lots of flexibility in the combining and selection of active behaviors. An example of this flexibility can be seen in a team of robots that were given the task

of collecting trash, in the form of soda cans, and depositing them to a pre-specified location [15]. There were nine behaviors arranged as a finite state machine with perceptual triggers to switch between the states. The states represent a configuration of behaviors that perform a task such as **move to trash**. These groups of behaviors are known as assemblages. In the trash collecting task, all parameters, gains, and state transition conditions were selected prior to the robot's operation and remained constant throughout the task. Balch presents a method that uses reinforcement learning to learn the appropriate sequence of assemblages for a perceived situation [14].

In all of the previously discussed mobile robot research, the primitives are designed and constructed by a human. The work of Mahadevan and Connell is a step toward developing learning algorithms that allow a robot to learn new behaviors in an initially unknown environment [77]. They explore the usability of reinforcement learning to learn the behaviors of a subsumption style architecture and demonstrate the performance of the algorithms in a robot box pushing environment. Each behavioral module evaluates the state of the environment and decides if the behavior is applicable. The behavioral modules use reinforcement learning and are given a reward function that guides their behavior and the conditions under which each module is applicable. The priority of each module in the overall task is also specified.

Methods are also being developed that allow mobile robots to learn primitive execution policies from observation data. The behaviors of **approach object and stop** and **exit by the door** [54], and a **tree tending skill** [46] have been learned by a mobile robot after it was guided through the task multiple times by a human using a joystick. If the observed training data includes bad examples, it may need to be filtered before being presented to the robot for learning. This is process tedious and is usually conducted by a human task expert. Larson and Voyles [72] are working on methods to automatically select acceptable training data from the available data. They collected data as a human controls a robot via a joystick in **left wall following** and **hallway traversal** behaviors. Robots that performed the behavior using the filtered data performed better then those that used only the raw data.

2.5 Using Primitives in Learning Skills on Robots with Many DOFs

This section focuses on research on robots that have many degrees of freedom (DOFs), may be capable of performing a large number of tasks, and are specifically targeted for non-factory environments. Robonaut is an example of a high DOF robot that is being designed to perform a large variety of tasks in a spacecraft environment [4]. Increased interest in autonomous robots with forms that resemble dogs [136, 17] and humans [4, 9, 21, 45, 49, 65, 124, 104] has led to increased research in high DOF robotic systems that operate in non-factory environments and interact with humans. Autonomous legged robotic systems are increasingly being used in competition events such as RoboCup soccer [75, 106]; that community has the goal of having a team of humanoid robots play a game of soccer against the 2050 Worldcup champions and win [29].

Environments outside the factory are much more difficulty to programming robots for than the controlled environment inside the factory. The group of tasks that an assembly robot will be required to perform is usually limited and it is constructed with those tasks in mind. On the other hand, the range of tasks that a general ability robot, such as a humanoid robot, may be called upon to perform is much larger. Robots in the factory environment have limited interaction with humans and the humans that they do interact with are skilled in the operation of the robot. Outside the factory, robots will interact with a large variety of people with various skills. These difficulties have led to a large amount of research into teaching these robots using learning from observation methods and/or using primitives [13, 34, 37, 69, 112].

Primitives in these robotic systems are mostly being explored at two levels; task [1] and motor [133, 114]. Task-level primitives are specialized to the task that needs to be performed and the underlying hardware may or may not be taken into account. Motor-level primitives provide the robot with basic movement skills and can be more general in that the primitive can be performed in a variety of tasks. It is possible for task-level primitives to use motor-level primitives [2]. For example in the primitive task of **balancing a pole** the task-level primitive would provide the movement needed at the bottom of the pole to keep it upright. A motor primitive could then supply the robot joint angles that would cause the robot to perform the desired movement.

2.5.1 Learning Motor-Level Primitives

Wolpert and Kawato [133] and Billard [22] were inspired by biology to create models for motor control. Wolpert and Kawato propose the use of multiple paired forward and inverse models (MPFIM). Evidence has been shown for their existence in the human brain [57]. Each MPFIM contains an inverse model that can provide motor commands and a forward model of the environment that predicts the next state of the environment after observing the current state and the given motor commands. When the next state is observed a prediction error is generated for each MPFIM. The MPFIMs that are selected to drive the system and the ones that will learn during the current movement are based on the MPFIM's prediction errors. Therefore motor primitive selection and execution policy learning occur simultaneously.

Billard's model is inspired by the discovery of the mirror neuron system in the premotor cortex of monkeys [35]. These neurons become active during a specific movement and also when the monkey observes a movement that is similar to that specific movement. The modules in Billard's model are named after parts of the brain and provide similar functions. This research seeks to use neural networks as much as possible and a model was shown that learns arm trajectories from joint angle data collected from a person while performing the trajectory [22]. The data is segmented by observing the changes in direction and velocity of the joint angles. From this segmentation, speed and direction of movement parameters can be obtained.

The motor primitives research of Dillmann et al. [38] and Schaal et al. [114] seeks to find movement policies for specific types of movements. Rapid transfer movements (open loop trajectories) and slow accurate motion (sensor-based skills) learned from observation are being explored by Dillmann et al. Their research presents a method to segment observation data into trajectory primitives of a fuzzy nature where the observed trajectory may exhibit characteristics of two primitives that slightly overlap. Dillmann also discusses the need to smooth the observed data and to observe a minimum number of measurements to find trends in the data.

The research of Schaal et al. [114] also investigates the learning and use of two motor primitive policies. One movement primitive is through a discrete dynamic system that creates point-to-point movements based on internal or external specifications. The trajectory primitives are similar to

those of Dillmann but the learning method is inspired by data from neurobiology. The other type of primitive being explored is a rhythmic system that can add an additional oscillatory movement relative to the current position of the discrete system. This research addresses the need, and provides a structure, to couple the actuators of a multi-joint system so they remain phase-locked and to provide parameters to specify a certain rhythmic movement.

2.5.2 Learning Task-Level Primitives

The early work of Aboaf et al. [1] explored the use of task-level commands to increase a robot's performance at the 3D task of **juggling a tennis ball**. By restricting learning to the task level, the degrees of freedom are greatly reduced. The underlying robot control policy is held constant and the aim points are adjusted to overcome inaccuracies in the policy. A disadvantage of this approach is that the information learned is specific to the task and cannot easily be used in other tasks. This research identified other issues involved with task-level learning. In the system, the modeling errors could not be adequately compensated for by providing a constant correction to the task-level command. The authors chose the task-level variables to be adjusted on an ad hoc basis but indicate that the modeling errors may depend on factors which are difficult to determine.

Kamon et al. [64] use primitives for grasping and evaluate the candidate grasp before it is used. Their system uses a single vision image from which they derive a small number of object features. The features are used to select a candidate grip from among grips that are learned off-line and stored in a case like representation similar to those used with motor schemas [74]. Another module then evaluates the probability of success of the selected grip on the observed object by using previous performance knowledge. Grips are continually selected and evaluated until a threshold is reached at which time the grip is actually performed. The success of the selected grip is then stored in the knowledge base. This gives a robot working within this framework the ability to learn and increase its skill while operating in the environment.

The tasks of **swing up** [11] and **balancing** [115] using an anthropomorphic robot arm were explored by Atkeson and Schaal. This research uses human observation information to learn a parametric and non-parametric model of the environment. In the **swing up** task, the robot first follows the trajectory performed by the human. The robot fails to perform the task whit it merely mimics

the movements of the human, but data collected during this attempt is used to update the model. A trajectory optimization technique then uses the model to create a new trajectory using the human trajectory as an initial guess. Using a parametric model the robot succeeded on the second try and using a non-parametric model succeeded on the third iteration. Success is defined by swinging the pendulum up with a final velocity small enough to be captured by the **balancing** primitive. The research on the **balancing** primitive explores methods to use observed information with reinforcement learning to map appropriate actions to perceived states. The observed information is used in a variety of methods and Schaal [115] presents the usefulness of each.

2.5.3 Discovering and Locating Primitives in Observed Data

Some of the research presented on high DOF robots uses observation information to increase the learning rate [11, 61] and therefore the continuous observation data must be segmented if it is to be used in these systems. The research of Jenkin et al. [58] and Mori et al. [87] seek to segment the observation data into known primitives. Jenkin's research is part of a larger framework aimed at using primitives in imitation learning [79]. They used a 2D vision system to segment observed arm movements into **line**, **arc**, and **circle** primitive movements. The observed data is matched to a set of primitive descriptions that where created previously using observed supervised data. Mori's research seeks to provide human action descriptions, such as **walking**, **sitting**, and **jumping**, in a human like way. Their approach handles the case where an observed action simultaneously appears to belong to more than one action set and the possibility that more than one action is simultaneously being performed. A separate module is created for each action that judges if the input action is the assigned action and the captured data can simultaneously be presented to each module. The modules use a rule base and a fuzzy logic type system to create an output. The outputs of the modules are then used to vote on the classification of the observed action.

Most approaches using primitives predefine the set of primitives in the environment but there are a few methods that automatically discover primitive types in the observed data using predefined segmentation rules or template matching. A module in Mataric's imitation learning framework [79] automatically derives arm movement primitives from collected motion capture data [43]. This research addresses the fragility of choosing a proper segmentation of the data. The segmented data is

converted to a vector format and principle component analysis (PCA) is used to obtain the significant dimensions. The data is then clustered via a k-means clustering algorithm where each cluster represents a primitive type. Even though the research of Iba [55] seeks to classify movements so that latter portions of a partially observed movement can be predicted, it also has the components needed to generate abstract arm movement primitives from the observed joint data. The created framework represents the movement as a sequence of joint parameters stored in nodes of a tree. When observing a movement in joint space the algorithm seeks to match the movement to a previously stored movement. If the observed movement does not closely match a stored classification, a new node is created with a new classification.

2.6 The Use of Primitives in Reinforcement Learning

Reinforcement Learning (RL) techniques were used in some of the previously presented research as a tool. This section discusses research that explores issues of using primitives within the RL framework. There is a significant amount of research in this area that seeks to reduce the state space and problem complexity through the use of primitives. Whereas a majority of the previously described research was conducted on actual robots, research in this area is mostly characterized by block worlds and other simulation environments. Various RL frameworks have been created that apply the concept of a primitive. Kaelbling [60] presented an RL algorithm that creates goal achieving robots that learn policies to reach a goal. If a goal needs to be accomplished again later, the robot immediately knows what actions to take because policies are saved. Kaelbling [59] extended the use of this algorithm to a hierarchical structure. This research discusses the need for closed-loop primitives instead of open-loop macro actions with no feedback, and the issue of obtaining sub-optimal performance by using primitives. The primitives in this case were predefined by the goal and the algorithms were tested in a 10x10 grid world.

The research of Singh [117] also uses predefined closed loop primitives that are represented as abstract models. This research is conducted in an 8x8 grid world and addresses the issue of how long an abstract model should be and how often backups should be performed. The primitives, called variable time resolution models, are in an architecture called H-DYNA which is an extension of Sutton's DYNA architecture [121] and backups are performed on the non-controlling policies

whenever time is available. This allows primitive policies to be updated even if they are not directly controlling the plant.

Dietterich [36] clearly separates the primitive's control policy from the RL value function in MAXQ decomposition architecture. The subtasks can have any control policy and, similar to the primitives in Kaelbling's research [60], are defined by the state they terminate in. The task is configured as a graph of subtasks and low-level actions. This research explores running a chosen primitive until it is complete without allowing it to be interrupted. A robot in a simulated 5x5 grid world domain was switched to the single step primitive selection method after it learned a policy. The robot's reward per step initially decreased significantly and then increased to a value above that of the method that did not interrupt the primitives.

2.6.1 Primitives in Larger State Space Environments

The architecture of Parr and Russell [98] is similar to that of Dietterich in separating the RL value function from the primitive's control policy and is conducted in an environment containing approximately 3600 states. The primitives, called machines, are predefined and operate with limited actions and a partial description of the environment. Their approach uses RL to learn which machine should be chosen at predefined observation states, called choice points. A machine policy runs until a choice point is observed at which time the reward and state value is observed and a backup is performed. Huber's research [53] seeks to create a hierarchal RL architecture for a continuous state space. The research is conducted on a four legged robot that learned a policy for turning. The policy was then reused in a higher-level, goal-seeking task conducted in simulation. The robot in simulation outperformed other robots in simulation that did not contain prior knowledge of the turning behavior.

The research of Sutton et al. [123] rigorously addresses many of the issues related to using primitives in RL. Their framework contains temporally abstract primitives called options that allow a robot operating in the environment to use an option policy or a low-level action. Because primitives can lead to a suboptimal solution, this research evaluates the observed state after each action taken by an option to determine if the option policy should be interrupted This approach is similar to that of Dietterich [36] but the robot also has the ability to not use any option policy and instead resort to

using only low-level commands with the strict RL framework. They also explore the possibility of improving the primitive policy by specifying subgoals and having the robot increase its performance at obtaining those subgoals. The options presented are not general and can only be used in the state space they were learned. The option's policy and low-level actions are tightly coupled to the value function in pure RL techniques and the learning progresses entirely unsupervised with no prior information given to the robot.

Methods to automatically find primitives, called macro-actions, within the continuous state space of a robot learning using RL is being explored by McGovern [83]. Instead of searching for the primitives themselves, this research searches for subgoals in the state space that are then used to create subgoal-seeking primitive policies. Subgoal states are identified by observing a peak in the reward history and because the state space is continuous, k-means clustering is used to classify the subgoals. McGovern and Barto [84] continued this research in the options framework discussed above. They explore other methods to find subgoals and the need to have a system to add and delete options.

2.6.2 Using Observed Information

The research of Schaal [115] presents a method in which data obtained from observing a pole balancing task performance is used to create a model that can be used in the RL framework. In this research, the movements of the robot directly affect the control of the pole and determine the reward received. In similar research being conducted by Price and Boutilier [102, 101], the robots are mobile robots given the task of moving to a goal location. In this domain, rewards are given as a function of the state of the robot instead of the state of an external controlled object such as the pole. This research is conducted in the context of homogeneous [102] and heterogeneous [101] robots. The robots can only observe the resulting state of an action taken by another robot; they do not know the action that was taken to produce that outcome. Price and Boutilier present methods in which this information can be used to learn a value function or guide the robot towards relevant areas of the state space. In the case of robots with similar abilities, the robot knows that it can also make the observed state transition. But if the robots do not have the same abilities, the observing robot must have a way to know if it is capable of making such a transition.

Ryan and Pendrith's architecture combines RL and teleo-operators [19], based on Nilsson's teleo-reactive formalism [95], and makes use of both domain knowledge and observed information [109]. The architecture is used by a robot that learns to fly an airplane in simulation [108]. Knowledge from task experts is used to decompose the flying task into maneuvers that are composed of simple behaviors. Because the primitives are defined by preconditions and post conditions, appropriate rewards are assigned based on whether the conditions are met. Data obtained from observing expert pilots is also used to increase the learning rate of the defined behaviors.

2.7 The Use of Primitives in Computer Graphics

The use of virtual characters in computer animation, games, and virtual environments is increasing. If the character is humanoid, it must also have realistic and believable movements and for virtual environments and games the movements must be produced in real time. Some of the research in this area uses primitives to accomplish these goals.

The research of Mataric et al. [81, 80] explores the use of three different control strategies to give a virtual humanoid a set of predefined primitive behaviors. The three approaches are PD servo control, torque-field control, and impedance control. Their research presents some of the pros and cons of using each implementation. A sequence of subtasks of dancing the Macarena is defined and the primitives needed to accomplish those subtasks are specified so that the humanoid can perform the dance. Multiple primitives can run simultaneously and the outputs can be combined to produce the desired trajectories. For example, the **go-to-point** and **avoid-collisions** primitives can run concurrently to ensure a movement to the desired point without colliding with objects that may be in the way. The primitives can also have parameters that specify how it is to be performed or the desired outcome.

Hodgins et al. [51] have also used primitives, in the form of control algorithms, in their research in animating virtual humanoids in the performance of various dynamic athletic behaviors. Wooten and Hodgins [134] presented difficulties of transitions between primitives with dissimilar behavior and provided a solution in the domain of animated humans that have the primitive behavior of **leaping**, **tumbling**, **landing**, and **balancing**. The primitives within their framework, referred to as basis behaviors, also have parameters that control their performance. The primitives types are

selected in such a way that the outcome of one primitive can be used as the input to another. For example a **leaping** primitive can be followed by **landing** and then **balancing**. The parameters provide the ability to adjust the behavior of the primitives and ensure the resulting state will be sufficient for the next primitive to be successful.

Faloutsos et al. [40] present a framework for controlling animated characters using primitives that are pose-controllers. Their research also addresses the need for primitives to end in a state that is acceptable to the next primitive to be performed and stresses that the preconditions that will allow a primitive to be successful must be known. They present a learning approach that can be used to learn the possible preconditions but the preconditions can also be specified manually.

CHAPTER III

TESTBEDS

Search me, O God, and know my heart;
test me and know my anxious thoughts.
- Psalm 139:23 (NIV) [96]

Do not conform any longer to the pattern of this world, but be transformed by the renewing of your mind. Then you will be able to test and approve what God's will is–his good, pleasing and perfect will.
- Romans 12:2 (NIV) [96]

In this thesis we present various approaches to challenges involved in having robots learn from observation and from practice. We have tested our ideas on robots that operate in an air hockey and marble maze environment. We have also created software robots that perform these tasks. This chapter describes the construction of the testing environments.

3.1 Air Hockey

Air hockey is a game played by two people. They use round paddles to hit a flat round puck across a table. Air is forced up through many tiny holes in the table's surface which creates a cushion of air so that the puck slides with relatively little friction. The table has an edge around it that prevents the puck from going off the table, and the puck bounces off this edge with little loss of velocity. At each end of the table there is a goal area. The objective of the game is to hit the puck so that it goes into the opponent's goal area while also preventing it from going into your own goal area.

3.1.1 Software Air Hockey

Figure 4 shows the virtual air hockey game created that can be played on a computer. The game consists of two paddles, a puck and a board to play on. A human player uses a mouse to control one paddle. At the other end is a simulated or virtual player who uses his arm and hand to position the paddle. For a given desired paddle location, the arm and hand are placed to position the paddle in the appropriate location, and any redundancies are resolved so as to make the virtual player look

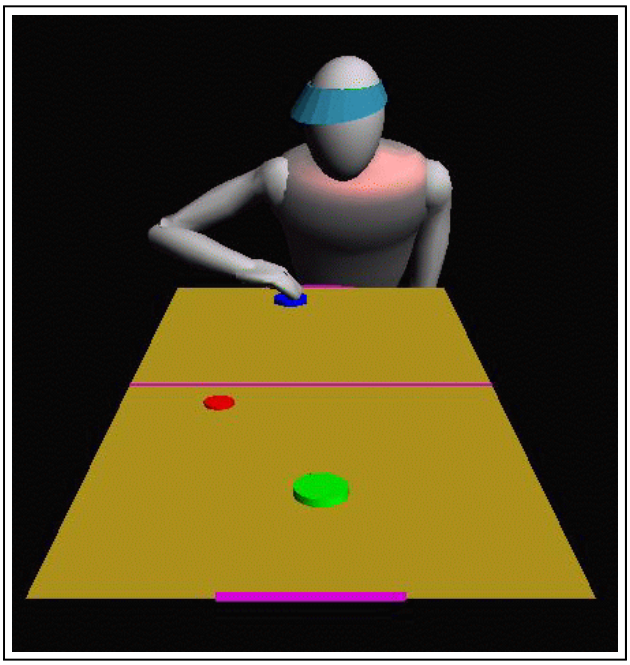

Figure 4: The virtual air hockey environment. The disc-shaped object near the centerline is a puck that slides on the table and bounces off the sides, and the other two disc shaped objects are the paddles. The virtual player controls the far paddle, and a human player controls the closer paddle by moving the mouse. The object of the game is to score points by making the puck hit the opposite goal (the purple/light area at the ends of the board).

"human-like." If the target is not within the limits of the board and the reach of the virtual player the location is adjusted to the nearest reachable point. The torso is currently fixed in space but could be programmed to move in a realistic manner. The virtual player's head moves so that it is always pointing in the direction of its hand, but is irrelevant to the task in this implementation except in that it makes the motion appear more natural.

The paddles and the puck are constrained to stay on the board. There is a small amount of friction between the puck and the board's surface and there is also energy loss in collisions between the puck and the walls of the board and the paddles. Spin of the puck is ignored in the simulation. The position of the two paddles and the puck, and any collisions are recorded.

3.1.2 Hardware Air Hockey

The hardware implementation of air hockey, Figure 5, consists of the humanoid robot DB [9], a small air hockey table, and a camera-based tracking system. The robot observes the position of the puck using its on-board cameras, Figure 6, and a vision processing system designed to find the

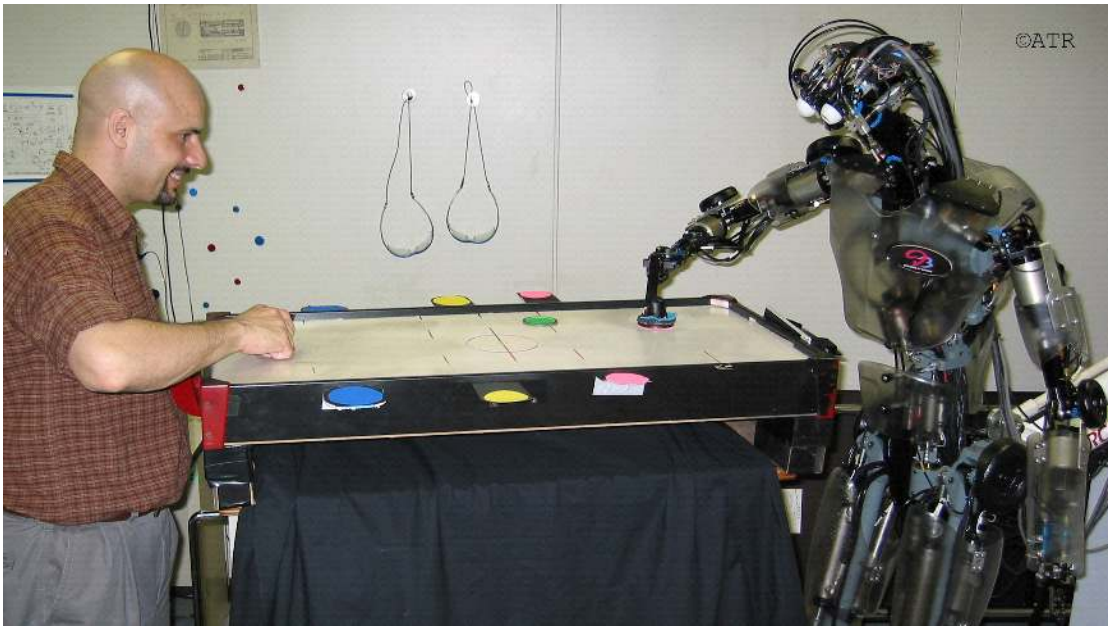

Figure 5: Playing air hockey with humanoid robot DB.

position of colored objects in the image. Because the humanoid's torso is moved during play to extend the reach of the robot, the head and eyes are moved so that the playing field is always within view.

3.1.2.1 Vision

Because air hockey is played on a flat surface, we can model the image plane to hockey board mapping as a perspective mapping between two planes. It is well known that such a mapping can be modeled by a 3x3 homography, which is defined up to a scale factor and thus has 8 degrees of freedom [135]. This mapping is invertible, so the information from one eye (camera) suffices to uniquely determine the position of the puck on the board. However, we must be able to update this mapping at every measurement time because the robot's head moves during the game. In theory we could accomplish this task by calibrating the camera at a preferred configuration and use forward kinematics to calculate the current image-to-board mapping, but this is impractical because the humanoid robot motion involves many degrees of freedom and is highly nonlinear. Instead the system is recalibrate at every time step. Because every homography has eight degrees of freedom, we must know the position of at least four points in the plane of the table and in the image to

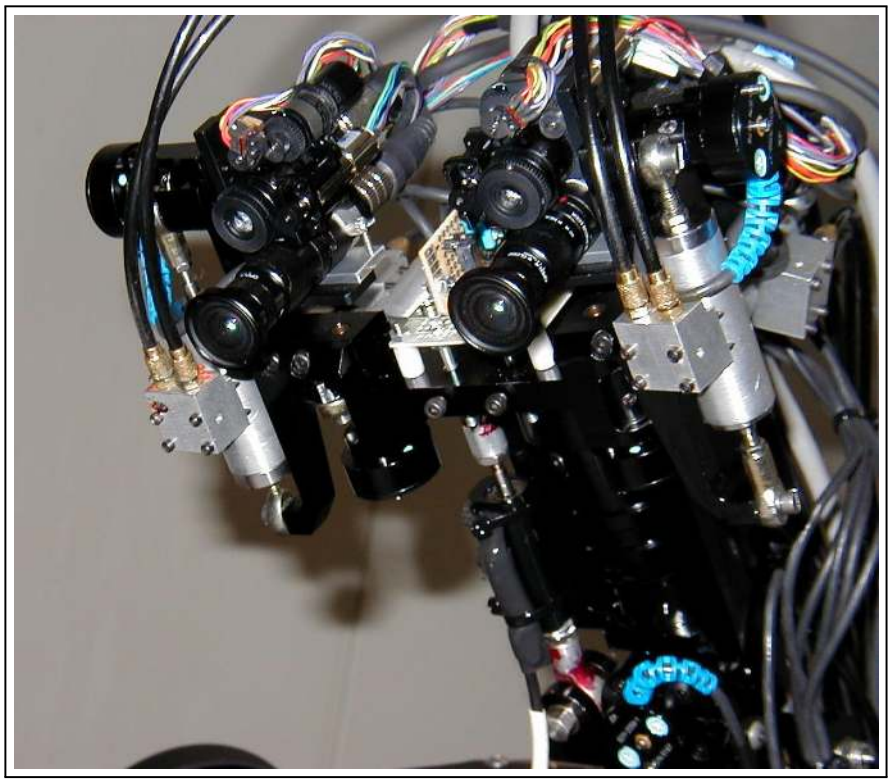

Figure 6: The head of the human robot contains two "eyes" each made up of a wide angle and narrow angle camera on pan-tilt mechanisms. The robot uses one wide angle camera for playing air hockey.

recalibrate the camera. Thus the number of objects that we need to track is at least seven; four fixed points on the board for calibration, puck, and both paddles. To make the recalibration more accurate and the tracking process more robust, the system makes use of more than four points to recalibrate the system.

A homography describing the perspective mapping between the image plane and the hockey board is given by

$$s\boldsymbol{x}_i(t) = \boldsymbol{H}(t)\boldsymbol{u}_i(t),\ i = 1, \ldots, N,\ N \geq 4, \tag{1}$$

where $\boldsymbol{x}_i(t) = [x_i(t), y_i(t), 1]^T$ are the known positions of the markers on the hockey board (we measure them by hand before starting the game) and $\boldsymbol{u}_i(t) = [u_i(t), v_i(t), 1]^T$ are the positions of the detected markers in the image. Let $\boldsymbol{h}_1(t)$, $\boldsymbol{h}_2(t)$ and $\boldsymbol{h}_3(t)$ be the columns of $\boldsymbol{H}(t)$. Writing $\boldsymbol{z}(t) = [\boldsymbol{h}_1(t)^T, \boldsymbol{h}_2(t)^T, \boldsymbol{h}_3(t)^T]^T$, Eq. (1) can be rewritten as

$$\begin{bmatrix} \boldsymbol{u}_i(t)^T & 0 & -x_i(t)\boldsymbol{u}_i(t)^T \\ 0 & \boldsymbol{u}_i(t)^T & -y_i(t)\boldsymbol{u}_i(t)^T \end{bmatrix} \boldsymbol{z}(t) = 0. \tag{2}$$

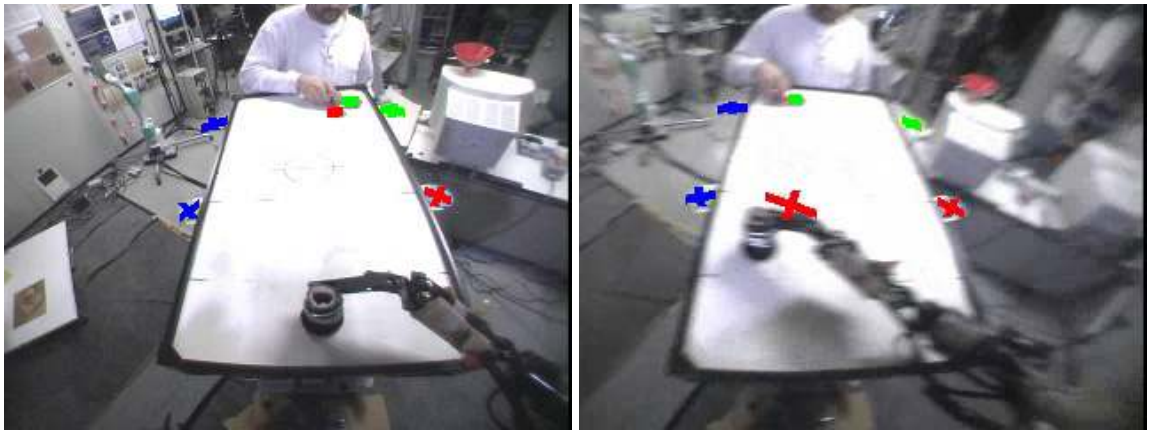

Figure 7: The view from the robot's eyes with the tracked objects marked. The puck is totally occluded in the right image. The right image is also blurred due to the movement of the robot.

Writing Eq. (2) in a matrix form results in a matrix equation $\boldsymbol{A}(t)\boldsymbol{z}(t) = 0$, where $\boldsymbol{A}(t)$ is a $2N \times 9$ matrix. As $\boldsymbol{H}(t)$ is defined only up to a scaling factor, the solution is well known to be the eigenvector associated with the smallest eigenvalue of the 9×9 matrix $\boldsymbol{A}^T(t)\boldsymbol{A}(t)$ [135]. Alternatively, one could solve Eq. (2) directly by setting one of the parameters, typically $h_{3,3}(t)$, to 1.

Strategy for Error Recovery Typically, a game of air hockey goes on for several minutes and the vision system is expected to provide locations of the objects of interest during this period without failure. Because it may not be possible to completely avoid tracking failures, we designed a specialized error recovery scheme that ensures the successful operation of the vision system over longer periods of time.

Most of the blobs never come close to each other in the air hockey game. There are three groups: the blobs fixed at the edge of the board, the two blobs associated with the paddles used for hitting the puck and the puck itself. The color tracking system [21] uses shape-related probability distributions and assigns larger values to the neighboring pixels, so there is no need for blobs from within these groups to have different colors. We can thus in principle use only three different colors to identify the objects of interest, although in practice it might be advantageous to use more colors.

Recovering from Occlusions There are a few situations when our tracker might fail. The most common problem is that a robot arm occludes the puck, Figure 7. Such cases are detected by

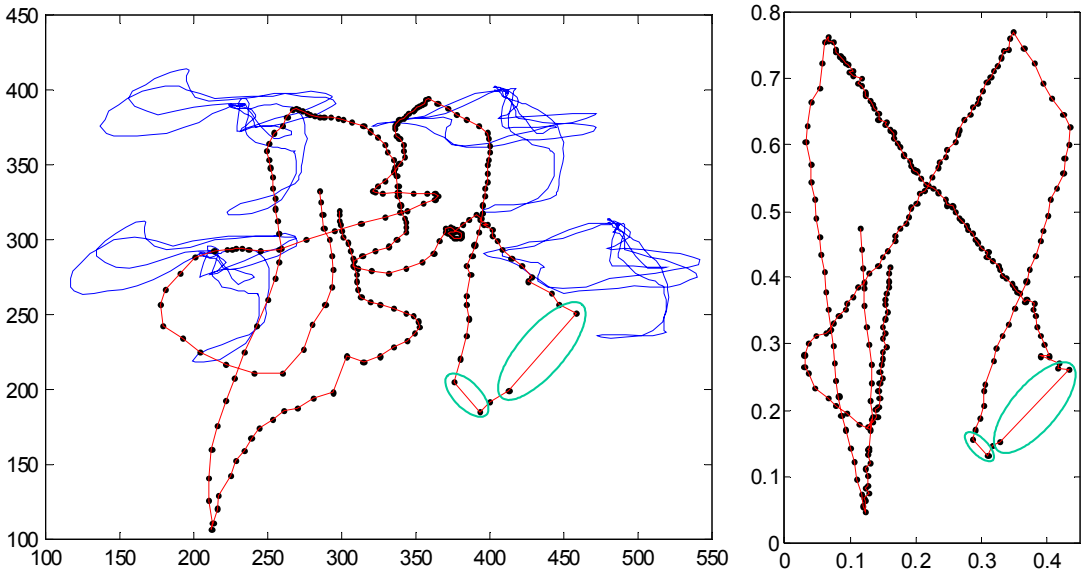

Figure 8: The left graph shows the raw vision cordinates (pixels) of four objects placed at known locations and the moving puck. On the right is the computed position (meters) of the puck based on the information shown on the left. The circled segments are where the vision system lost track of the puck for several samples.

calculating the sum of probabilities normalized by a number of pixels considered in the calculations. If this sum is less than 10% of the expected area covered by the puck within the region of interest, the calculation of the corresponding blob location becomes unstable and the tracker is assumed to have failed. When this happens, the tracker keeps looking for a puck for 0.5 seconds by starting the EM iterations using the latest detected location as an initial value.

After 0.5 seconds it is considered unlikely that the puck would still be near the last detected position and the tracker starts two new search processes: one in front of the opponent's paddle where we can assume that there will be a contact with the puck in the future and the other one in the region near the robot where the puck might still be located in the case of an occlusion. The second process randomly generates initial puck positions within the search region and thus ensures that the puck will eventually be found regardless of its current position within this region. The system does not embark on an exhaustive search covering the whole board, that could not be carried out at 60 Hz. The success of one of the search processes terminates them both.

Figure 8 shows the results of the visual tracking system using four fixed markers during 4.2

seconds of game play. During this interval the fixed markers are always being tracked, but on several occasions the puck was lost and then reacquired. The puck was lost just after the robot hit it. This is not a critical time since the robot's hit motions are fully planned and begin more than 100msec before the puck is hit.

Recovering from Tracking Errors Occasionally one of the side blobs used for recalibration is lost by the tracker. This problem is most often caused by a very fast motion of the robot's head when the robot performs a shot, which results in low-quality images and huge image motions. If the robot can't see at least four side blobs, the recalibration cannot be carried out and the robot cannot play the game. But fortunately this problem is not too serious because the robot calculates the motion trajectory before it starts executing the shot. After executing the shot, the robot returns to a preferred configuration and because the position of the board remains constant, the stored positions of the side blobs can be used to restart the tracking.

Although problems with paddle tracking are rare, we have implemented an error recovery scheme. Because the motion of both paddles is confined to a region along their respective ends of the hockey board we can start a search process in a region along the appropriate end of the board. The paddle positions are randomly generated in this region and used as initial values for an EM iteration until the paddle is found. This process is similar to the search when looking for a puck.

3.1.3 Positioning the Humanoid Robot

At first it may appear that the robot only needs to use one arm to play air hockey. Using only the arm simplifies the system in that there is less movement of the head and therefore of the cameras. It also eliminates many degrees of freedom. Our first implementation used this scheme and we found it to be impractical because it limited the moves the robot could make and the area that can be reached on the board. Moving the torso extends the robot's reach and abilities. But the movement of the torso causes the head to move in ways that are unnatural looking and may cause the air hockey board to not always be fully within view. Therefore the head and eyes must also be controlled to ensure the board is always within the field of view of the robot. The total degrees of freedom needed for this task is 17. The following joints are used in this approach: shoulder (2 joints), arm rotation, elbow, wrist rotation, hand (2 joints), waist (3 joints), head (3 joints), and eyes (2 joints each eye,

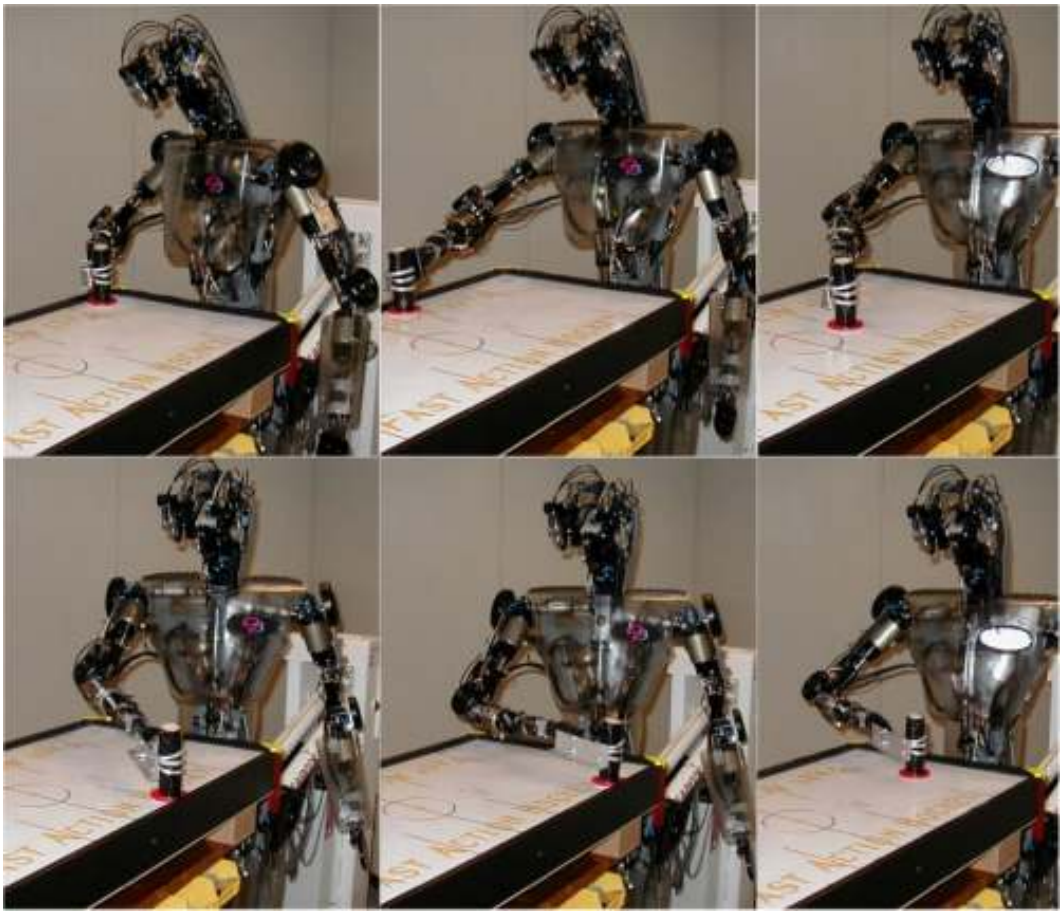

Figure 9: The six given configurations of the robot used to compute all enclosed configurations.

pan and tilt). Using all the joints except for the head and eyes, the robot must be positioned so that the paddle is flat on the board and moves smoothly from one location to another. The joints in the head and eyes are used to keep the entire board is in view at all times.

We have manually positioned the robot in several positions on the board while maintaining these constraints, figure 9. To get joint angles for any desired puck position, we interpolate using the four surrounding training positions and use an algorithm similar to that used in graphics for texture mapping [25]. This approach allows us to solve the inverse kinematics of the robot with redundancy in a simple way.

Figure 10 shows an example of using the four training positions *B1*, *B2*, *T1*, and *T2* to compute the configuration needed to place the paddle at the desired position of $P(x, y)$. The four set positions, *B1*, *B2*, *T1*, and *T2*, surrounding the desired position $P(x, y)$ define a polygon with Y up and X to the right. A vertical line is drawn at $P(x)$ and points are defined at the locations where this

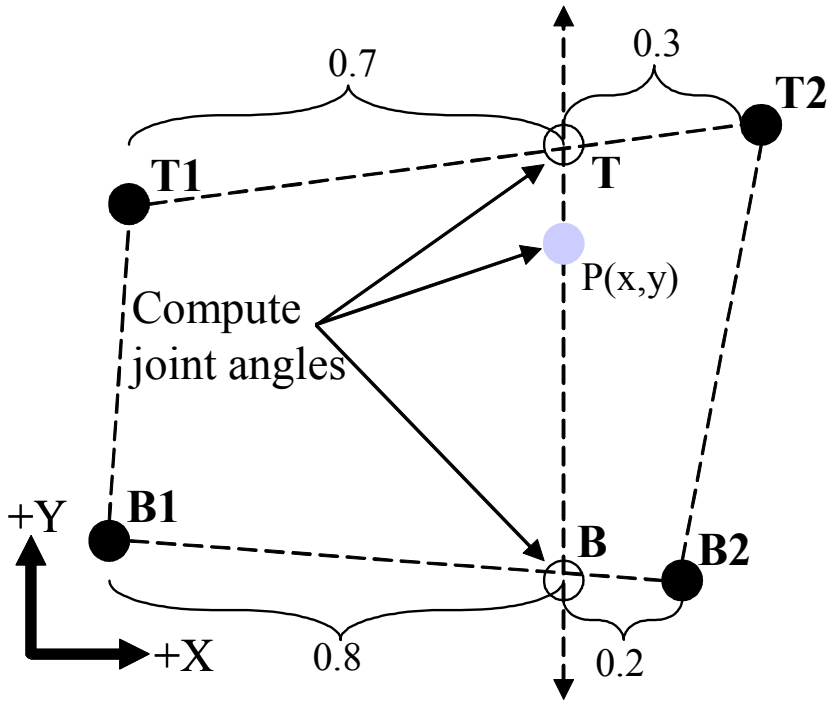

Figure 10: An example of using four given corner configurations to compute a configuration within the polygon.

line intersects the top and bottom of the polygon, points *T* and *B*. The joint angles are computed for the bottom-intersect location, *B*, by using the two joint configurations associated with the two bottom set locations, *B1* and *B2*. *B1* and *B2* configurations are averaged with a weighting computed from the percentage of the point *B* from *B1* and *B2*. As shown in the example in figure 10, the joint angles computed at point *B* will be 0.8 times the angles specified by point *B2* plus 0.2 times the angles specified at point *B1*. The configuration for the top point, *T*, is computed in the same way using the top two corners of the polygon, *T1* and *T2*. The configuration required to place the paddle at the locations *B* and *T* now exist and are used to compute the configuration needed to place the paddle the location $P(x, y)$. The percentage of the distance that $P(y)$ is from the bottom intersect point to the top intersect point is computed and is used as a weight to average the computed joint configurations of *B* and *T*.

The robot configuration needed to place the paddle at any location within the training examples can be computed using this algorithm. The robot is moved by specifying a desired board location along with a desired movement speed. A straight line trajectory is then computed from the present

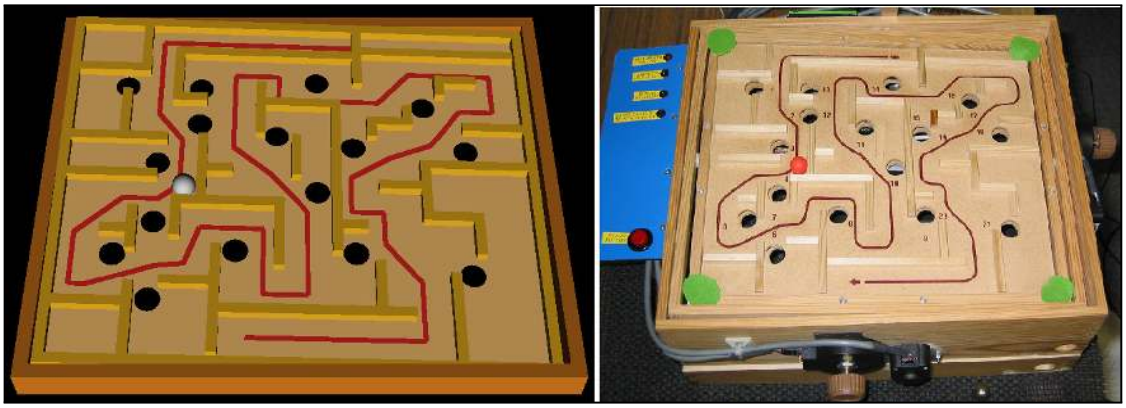

Figure 11: The virtual marble maze game on the left modeled after the hardware version on the right. The maze on the right has motors and sensors to control the board tilt angles and a vision system to observe the position of the marble.

location to the desired location. The robot is commanded to a position 420 times per second and therefore there must be a point on the trajectory at each 1/420 second interval. The trajectory points that the paddle will move through are computed using a fifth order polynomial equation with initial and final velocities and accelerations equal to zero. The robot is moved through the trajectory points and a robot configuration is computed for each point.

A new movement is only initiated when the robot is not moving and the desired end velocity and acceleration of a move are set to zero. With this strategy, the maximum velocity of the robot's move, and therefore the paddle movement, occurs at the halfway point of a straight line trajectory.

3.2 Marble Maze

In the marble maze game a player controls a marble through a maze by tilting the board that the marble is rolling on. We created two versions of this game that are shown in Figure 11. In these mazes the player must move the marble from the start, top center location, to the goal location at the bottom center of the maze. There are obstacles, in the form of holes, that the marble may fall into, and walls. The marble has a radius (r_m) of $0.7cm$, the holes have a radius of $0.75cm$, and the walls are $0.6cm$ thick. In the software version the human controls the board using a mouse. The simulator models the movement of the marble and noise is introduced into the simulator to make the outcome of actions less deterministic.

The human controls the board on the hardware version by using knobs connected to encoders.

The motor commands generated by the encoder system is read by the computer and sent to the motors. The position of the marble is obtained using a color tracking system [128]. The computer can also generate its own commands and send them to the motors. In both versions the time and the board and marble positions can be recorded as the game is played.

CHAPTER IV

OBSERVING THE TASK: PRIMITIVE RECOGNITION

"Now you have closely observed and diligently followed my teaching, conduct, purpose in life, faith, patience, love, steadfastness,"
- 2 Timothy 3:10 (AMP) [5]

To learn from observation, the robots must first have the ability to obtain information while a teacher performs the task. Chapter 3 describes the data that is collected while observing players in the air hockey and marble maze environments. This data consists of object positions and the time at which the observations were made. The simulation environments provide other information such as velocities and status of objects that are in contact. The data from both environments must be processed before it can be used.

The primitive recognition module of the framework uses definitions of the primitive types to segment and extract information from the observed data, Figure 12. This module finds the beginning and end of each observed primitive performed by the teacher. It then extracts details and formats the information as needed by other modules in the framework. This chapter shows the methods used to segment and format the information for use by the primitive selection and subgoal generation modules. The next chapter shows how the information is formatted for the action generation modules.

4.1 Air Hockey Primitive Recognition

We used human domain knowledge to define a set of primitives to work with in the air hockey environments, Figure 13. Three hit primitives are shown in Figure 14. The full list of primitives in air hockey is

- **Straight Shot**: A player hits the puck and the puck goes toward the opponent's goal without hitting a wall.
- **Bank Shot**: A player hits the puck and the puck hits the left or right wall and then goes toward

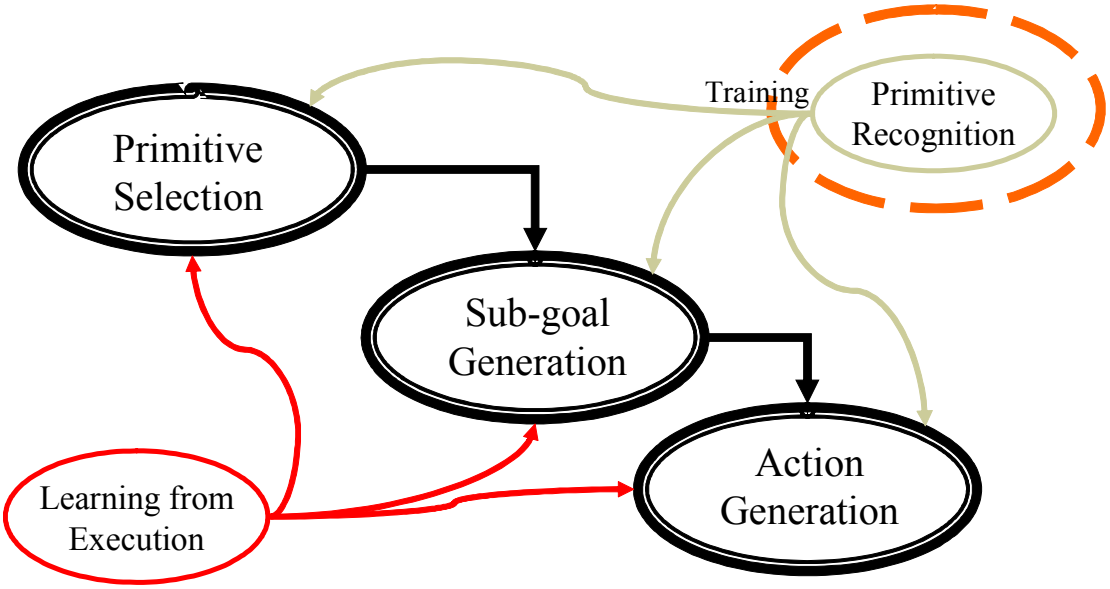

Figure 12: The primitive recognition module segments the observed data and formats it as needed for the other modules.

Figure 13: The software air hockey game on the left and air hockey playing with a humanoid robot on the right.

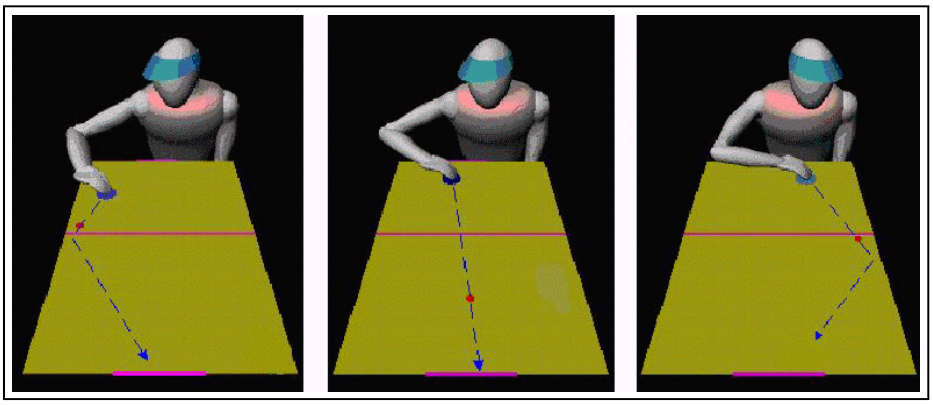

Figure 14: Three hit primitives being performed by the virtual player: right bank, straight, and left bank.

the opponent's goal.

- **Defend Goal**: A player moves to a position to prevent the puck from entering their goal area.
- **Slow Puck**: A player hits a slow moving puck that is within their reach.
- **Idle** : A player rests their paddle while the puck is on the opponent's side.

When the puck is moving quickly towards the player from the opponent's side the player can perform one of the shot primitives, **Straight Shot** or **Bank Shot**, or the **Defend Goal** primitive. When a shot primitive is selected for the fast moving puck, the player does not have time to move the paddle to an optimal setup position. Therefore the paddle begins in the idle position for these types of shots. Also, because the puck is moving quickly toward the player and time is limited, the player may not have time to move the paddle a large amount during the shot which means that hits must be made closer to the player's idle position.

The shot primitives can also be performed if the puck is moving slowly, or stopped, within reach of the player. When the puck is in this situation the player has time to first move the paddle to a more advantageous setup position before making the hit. The setup position will allow the player to hit the puck when it is farther from the player's idle position. The **Slow Puck** primitive encodes the action to be taken when the puck is in this slow moving situation.

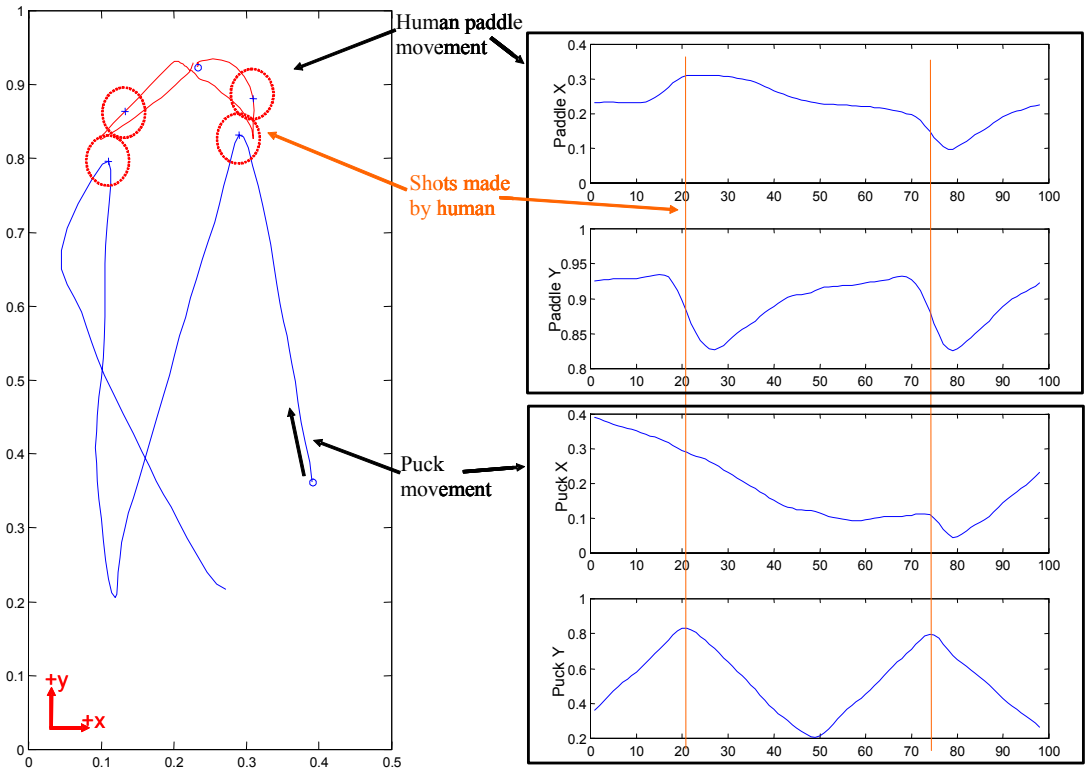

Figure 15: Raw data collected while observing a human making shots in the air hockey environment. The left figure shows the data plotted in two dimensions (meters). The right figure shows the data plotted against time. Collisions with the puck can be easily seen in this figure.

4.1.1 Retrieving Information from the Observed Data

This section describes *what*, and *how*, information is obtained from the data collected while observing air hockey. As the robot observes a task it looks for critical events. Critical events are large changes or discontinuities that can easily be recognized. A puck hitting a wall or a paddle is an example of a critical event. Figure 15 shows data collected from observing the hardware air hockey environment. The x and y displacement of the puck plotted against time in this figure shows that collisions cause an obvious change in the puck's movement trajectory.

The hit primitives describe the type of shot, straight or bank, that was made and the outcome, or subgoal, of making that shot. The subgoal of a shot is to have the puck hit a point on the back wall. Because the puck can be hit toward this point at various velocities, the absolute post-hit puck velocity is also included as an element of the subgoal. To learn the shot behavior from observing a player the robot follows these steps:

1. Look for when the puck is close enough to a paddle that it could be in contact with it. If the puck undergoes a discontinuous change in velocity, a hit is assumed at this location. If the puck has a large velocity toward the opponent's side, the position and velocity of the puck and paddle are recorded as the hit state.

2. Observe the puck's movements until it gets near the opponent's goal area while looking for collisions with the side walls. If a collision with a wall is observed, which side, left or right, is recorded. We are currently only considering shots with a single wall bounce.

3. Look for a collision of the puck with the opponent's wall or goal. If the puck reaches the opponent's wall or goal, the location is recorded as the target position.

4. Look for a collision with the opponent's paddle. If the puck is hit by the opponent before it reaches the opponent's wall, the location that the puck would go to if it was not blocked is predicted using a simple learned model and recorded as the target position.

From these observation steps the following information can be collected each time a shot is observed:

- The position of the puck when it is hit.
- The velocity of the puck just before it is hit; magnitude and direction.
- The angle between the puck and the paddle when the puck is hit.
- The velocity of the paddle when it hits the puck; magnitude and direction.
- The puck's velocity just after it is hit; magnitude and direction.
- The position on the back wall that the puck will go to if it is not blocked by the opponent.
- If the puck traveled towards the player from the opposite side of the board, the state of the puck before it crossed the center line heading towards the player.

The model used in step 4 enables the learning robot to estimate the target being attempted without

the shot having to be completed. The accuracy of the model can be critical in producing useful training data. Other methods can be used to reduce the reliance on the model, such as only considering shots that have actually hit the back wall without having hit the opponent's paddle.

4.1.2 Creating Primitive Observation Databases

Databases are created from the observed data that are used by the primitive selection and subgoal generation modules. This section describes the information that is recorded in the databases used in air hockey. Section 4.2 describes the databases used in the marble maze and the next chapter shows how the databases are used by robots operating in these environments.

A database has been created to encode the actions taken by an observed player when the puck crosses a pre-specified line heading towards the player. The actions contained in the database are the **Straight Shot**, **Bank Shot**, and **Defend Goal** primitives. The pre-specified line, called the decision line, is on the opponent's side and is just out of reach of the opponent, Figure 16. This means that the puck can no longer be influenced by the opponent's actions and future puck positions can be predicted. If, after the puck crosses the decision line, a collision with the player's paddle is observed and the puck goes toward the opposite side, it is classified as a shot primitive. In all other situations where the puck does not enter the player's goal, it is classified as a **Defend Goal** primitive.

If the **Defend Goal** primitive is observed, the position and velocity of the puck when it crosses the decision line and the location the paddle is moved to defend the goal is recorded. If a shot primitive is observed, the following information is recorded, Figure 16:

- The position and velocity of the puck when it crosses the decision line.
- The line the puck is hit at.
- The type of shot that was taken.
- The absolute velocity of the puck after it is hit.
- The target location on the back wall that the puck moved to after being hit.

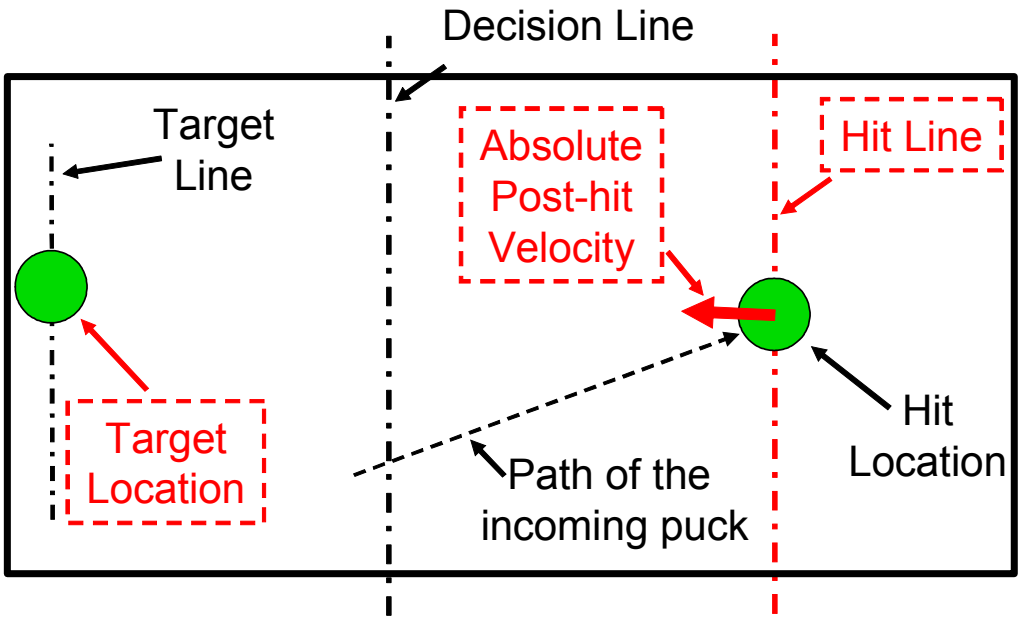

Figure 16: Information recorded in the fast-puck database when a shot is observed.

It is important to remember that this database will be used by the robot to decide actions it will take while operating in the environment. Because this is a continuous and dynamic environment it will not make sense to record the exact location where the puck is hit because the puck will not have exactly the same parameters while the robot is operating in the environment. Therefore the line where the puck is hit is recorded instead of the exact hit location. The hit line encodes the behavior of the player hitting the puck closer to themselves or closer to the center-line. The next chapter shows how a robot operating in the environment combines the hit line information with the incoming puck path to compute a hit location. The situation is also similar for the post-hit velocity vector. The direction of this vector only applies to a puck in the observed state. The absolute post-hit velocity gives an indication of how fast the player wants the puck to travel to the target location.

The puck is often moving very slowly within hitting range of the player. This situation occurs when the puck returns very slowly from the opposite side, after a failed shot attempt, or a **Defend Goal** primitive. The **Slow Puck** primitive is used in this situation and a slow-puck database is created to encode the required actions. A data point is created for the **Slow Puck** primitive database whenever a hit is observed and the incoming puck velocity is less then $0.5m/s$. The follow

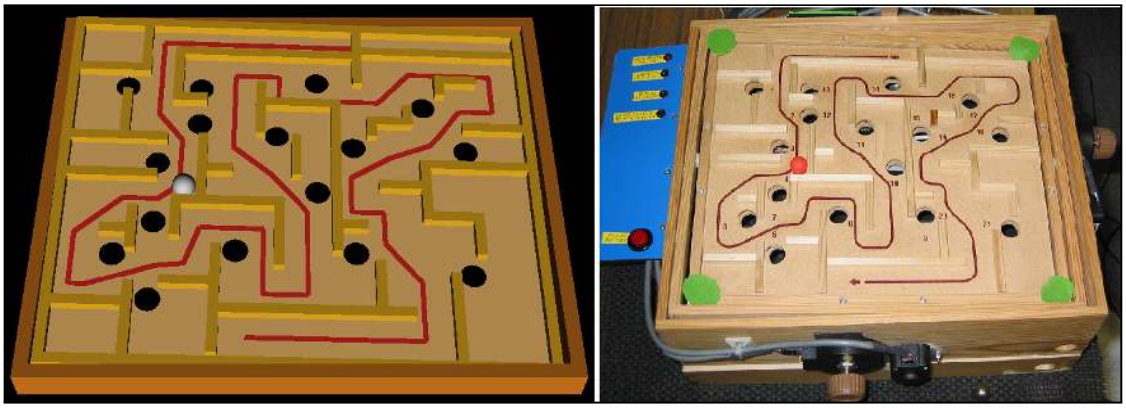

Figure 17: Software and hardware marble maze environments.

information is recorded in the database:

- The position of the puck when it is hit.
- The type of shot that was taken.
- The absolute velocity of the puck after it is hit.
- The target location on the back wall that the puck moved to after being hit.

Because the velocity of the puck is small, it is not included as a parameter in the database.

The **Idle** primitive is performed whenever the puck is on the side opposite the player and will continue until a shot is to be attempted or the goal is to be defended. A data point is created for the idle database by observing the puck's state when it is on the opposite side of the board and the position of the player's paddle. This database contains the position and velocity of the puck when it is on the opponent's side and the position of the observed player's paddle.

4.2 Marble Maze Primitive Recognition

The primitives for the marble maze game, Figure 17, are designed to give the robot the skills it will need to perform the task. The following primitives have been explored and are shown in Figure 18:

- **Roll To Corner**: The marble rolls along a wall and stops when it arrives at a corner.
- **Roll Off Wall**: The marble rolls along a wall and then rolls off the end of the wall.

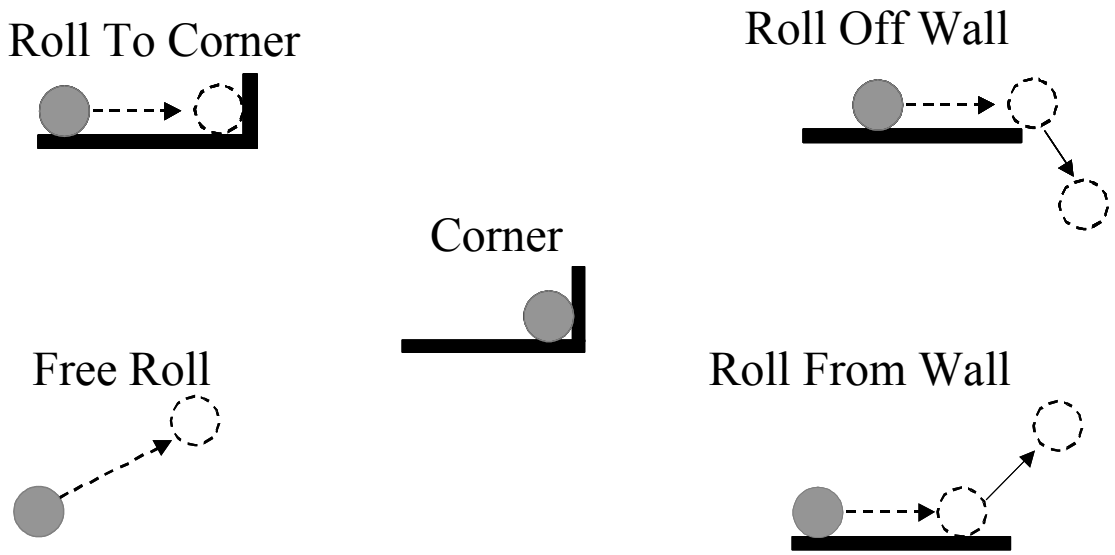

Figure 18: Primitives being explored in the marble maze environment.

- **Guide**: The marble is rolled from one location to another without touching a wall.
- **Roll From Wall**: The marble rolls along a wall and is then maneuvered off it.
- **Corner**: The marble is captured in a corner and then the board is positioned in preparation to move the marble from the corner location.

4.2.1 Retrieving Information from the Observed Data

The data collected in the simulator includes the status of marble-wall contacts. This status provides information on the side of the wall (top, bottom, left, right) that the marble is currently in contact with. The primitive definition is then used to create a sequence of critical events that allow the primitive to be observed in the data. The **Roll to Corner** primitive, for example, begins just before the marble makes contact with a wall that contains a corner and then rolls along that wall into the corner. An easy way to find an occurrence of this primitive is to look at each observed data point and find one where the status indicates that the marble is in contact with two connecting walls, such as the bottom and right walls, Figure 19. This is the location of the end of the primitive and the state of the environment, S_e, is recorded for this point. The velocity of the marble in the corner is obviously zero and therefore the velocity recorded as the goal velocity is the velocity of the marble

just before it hits the corner.

From this ending data point, the primitive recognition algorithm backs up through the data points and observes which wall the marble is rolling along. For the example shown in Figure 19, the marble is rolling along the bottom wall. The algorithm continues to backup through the data searching for the first occurrence of when the marble is no longer on the wall. This point is the start of the primitive and the environment state, S_s, is recorded. The parameters recorded for all environment states in the marble maze domain are the marble's position (Mx, My), velocity $(\dot{Mx}, \dot{My})$ and board tilt angles (Bx, By).

Using the above obtained information a primitive data point can be created that represents an action, PT (one of the specified primitive types), taken by the human while they were operating in the environment. This data point contains the following information:

- S_s - the state of the environment when the primitive was started.
- PT - the primitive type that was performed by the observed player.
- S_e - the state of the environment after the primitive is performed.

The goal of performing this primitive can be determined from S_e, the state of the environment at the completion of the primitive performance. This data point encodes what occurred during the observed performance in the following way: When the environment was in the state S_s, the human performed the primitive PT, and at the completion of that action the environment was in the state S_e.

A robot can use this information while operating in the environment in the following way. If the robot observes the state S_s, it should perform the primitive PT in such a way as to obtain the goal state of S_e. Because this domain has a continuous environment state space, it is unlikely that the robot will find itself in the exact state S_s. It makes sense to choose this same action, PT, for observed states in the vicinity of S_s but it is not known how far the marble can be from this state, S_s, and still allow the primitive to be successfully performed. The research of Faloutsos [40] presents the difficulties and a possible method of finding how far from S_s the environment state can be and this action still lead to success. Chapter 6.1.5 shows how this range of states where this primitive will succeed is learned by the robot as it operates in the marble maze environment.

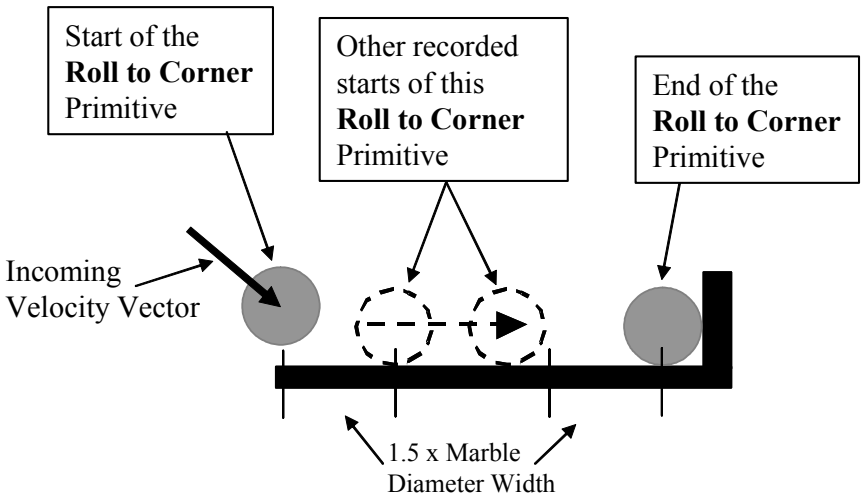

Figure 19: Creating **Roll to Corner** primitive datapoints from the observed data.

4.2.2 Creating Multiple Data Points for the Same Observed Primitive Performance

The marble moves from S_s to S_e while the primitive is performed. Therefore the primitive could also be started in any environment state between these two states. Because of this, other data points can be created with various starting states and the same end state. One method would be to create a new primitive data point for each observation occurrence. Since data is collected at $60Hz$, there would be $60 \times$ (time to complete the primitive) data points. Because the marble is may be moving very slowly during parts of the primitive, many of the data points will be redundant. Having too much redundant data in the database makes it difficult for the robot to select actions as explained in Section 5.0.5 and makes it difficult for the robot to learn from practice as explained in Chapter 6. In this implementation, a data point is saved at a position interval $1.5 \times d_m$, from S_s to $(S_e - (1.5 \times d_m))$ as shown in Figure 19 (d_m is the diameter of the marble). This scheme is used for all the primitives with the exception of the **Corner** primitive because the the marble moves only small amount, or not at all, during the execution of the **Corner** primitive.

4.2.3 Observing the Task Performed on the Hardware Implementation

The data collected from the hardware version of the marble maze is noisy and does not include the wall contact status and velocity information provided by the simulator. To remove the noise in the data it is filtered forward and backward through a butterworth filter. The velocities are then

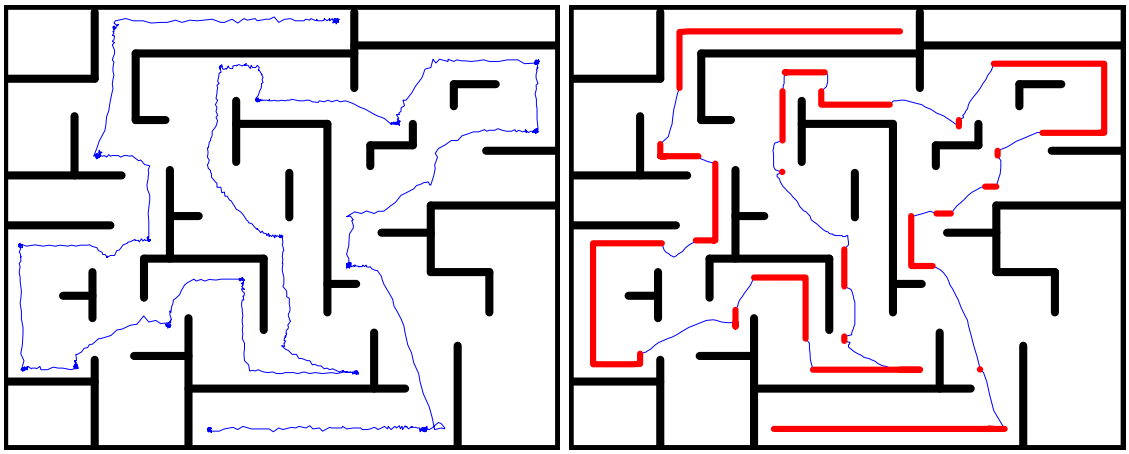

Figure 20: Left: Path of the marble collected while observing a human teacher. Right: The processed path with the thick line representing a marble-wall contact.

computed using the frame rate of $60Hz$. The wall contacts must be inferred from the collected data and this is done in two ways. One way is to look at the position of the marble and if it is very close to a wall for which positions are known, set the status to the appropriate wall contact. The other way is to observe the tilt of the board and the velocity of the marble. If the board is tilted in such a way to cause the marble to roll, but the marble velocity is zero, it may be held in position by a wall. In the hardware implementation the marble's initial momentum often prevents it from rolling until there is a significant amount of board tilt. Therefore if only board tilt and marble velocity were considered, it would appear that a wall is preventing the marble from rolling. For this reason the knowledge of wall positions and board tilt and marble velocity information are combined to process the hardware data and set the wall contact flags appropriately. After the observed hardware data is processed and the appropriate status flags set, it is applied to the recognition algorithms described in the previous sections. Figure 20 shows the raw collected data, left, of an observed game played by a human teacher. The right side of Figure 20 shows the processed data.

4.2.4 Results of Recognizing Marble Maze Primitives

The information shown in Figure 20 was used as the input data to the primitive recognition module as described in the previous section. The primitives recognized in this data are shown in Figure 21. The "$\times$" symbol shows the end, or goal, position of each recognized primitive. The "o" symbol marks the recorded start positions for each recognized and recorded primitive. The line connecting

Table 1: Twenty datapoints in the primitive database.

$Type$	SM_x	SM_y	$\dot{SM}_x$	$\dot{SM}_y$	SB_x	SB_y	EM_x	EM_y	$\dot{EM}_x$	$\dot{EM}_y$	EB_x	EB_y
CORNERTL	1.00	11.00	0.00	0.00	-0.09	0.11	1.00	11.00	0.00	-7.04	-0.10	0.11
CORNERBL	1.00	4.50	0.00	0.00	0.15	0.11	1.00	4.50	7.77	0.00	0.15	-0.09
CORNERTR	25.83	20.70	29.25	0.00	-0.07	-0.11	27.00	20.70	0.00	-13.69	0.11	-0.12
CORNERBR	27.00	17.00	0.00	0.00	0.14	-0.11	27.00	17.00	-4.66	0.00	0.16	0.07
GUIDE	4.02	5.41	8.62	7.59	-0.05	-0.12	4.75	5.87	13.60	4.8	-0.05	-0.12
GUIDE	12.78	20.30	2.55	0.00	0.05	0.02	12.85	19.47	-4.22	-11.97	0.09	0.04
GUIDE	20.02	18.31	3.80	10.53	-0.07	-0.06	21.25	20.36	9.00	10.03	-0.08	-0.06
GUIDE	20.94	14.10	-12.11	0.00	0.16	0.10	20.07	13.59	-15.78	16.27	0.16	0.10
RTCT	9.04	22.5	-17.93	0.00	-0.02	0.07	5.40	22.5	-18.71	0.00	-0.01	0.07
RTCT	21.25	20.36	9.00	10.03	-0.08	-0.06	27.00	20.07	9.54	0.00	-0.07	-0.11
RTCB	13.43	4.20	13.71	0.00	0.08	-0.11	17.60	4.20	21.16	0.00	0.07	-0.11
RTCB	18.21	1.00	-8.71	0.00	0.15	0.05	10.20	1.00	-14.17	0.00	0.15	0.05
RTCR	27.00	19.21	0.00	-1504	0.13	-0.12	27.00	17.00	0.00	-13.80	0.14	-0.11
RTCL	1.00	7.97	0.00	-15.47	0.15	0.11	1.00	4.5	0.00	-26.35	0.15	0.11
RFWL	14.10	8.51	-3.52	9.42	-0.03	0.04	13.97	11.11	-0.56	11.09	-0.02	0.04
RFWT	10.78	20.30	-0.35	0.00	0.00	-0.01	12.91	20.10	0.70	-7.18	0.07	0.05
ROWR	11.80	7.29	0.00	-10.95	0.07	-0.12	11.80	5.86	0.00	-14.96	0.08	-0.12
ROWB	4.71	15.70	-0.07	0.00	0.13	0.06	6.31	15.70	9.06	0.00	0.15	-0.05
ROWB	14.06	18.50	7.66	0.00	0.11	-0.05	16.09	18.50	10.40	0.00	0.12	-0.05

these points is the processed path of the marble. 116 primitives were recognized and recorded in the primitive database for this observed game as follows: **Roll To Corner**-34**, Roll Off Wall**-16, **Guide**-34, **Roll From Wall**-2, **Corner**-30.

The lower right graph in Figure 21 shows all the recognized primitives. This graph shows that there are small parts of the path that have not been classified by any of the primitive types. This omission is not a big concern because of environment dynamics and multiple task completions will be observed. In most areas where the path has not been classified, the dynamics of the environment has the marble in a state where it is moving toward the starting point of the next classified primitive. Therefore if the next primitive is chosen from this location, or if no action is taken during this time, the marble should continue to make progress through the maze. Also, because multiple task performances will be observed to create the primitive database, it is likely that there will exist a set of primitives in the database that will create a full path from the start to the end location.

Table 1 shows 20 of the 116 data points that were created from the observed data shown in Figure 20. The primitive type information specifies the primitive type that was observed and, if needed, the orientation of the primitive. The primitive type is specified by the following abbreviations: **Roll**

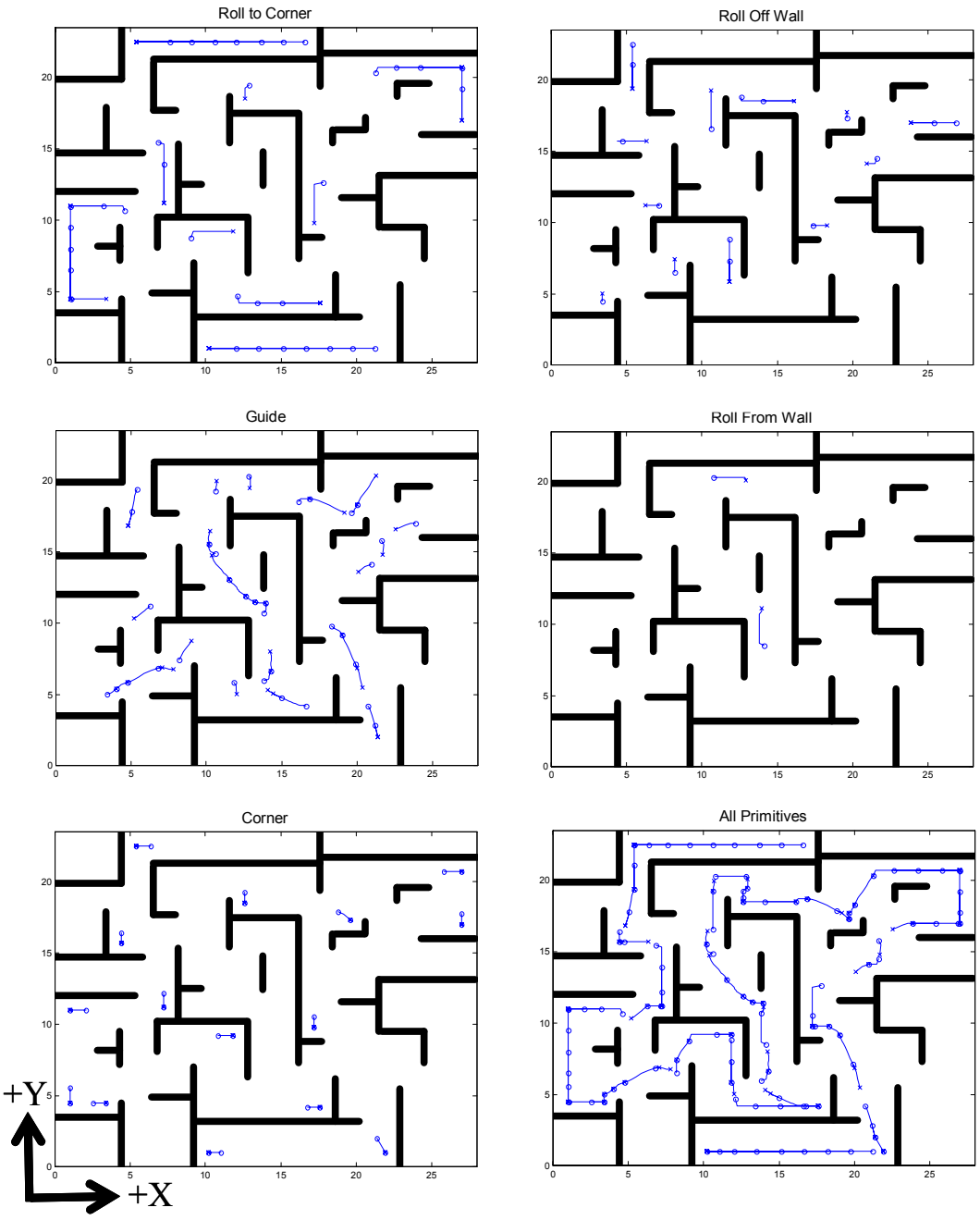

Figure 21: Results of recognizing the primitives performed by a human teacher.

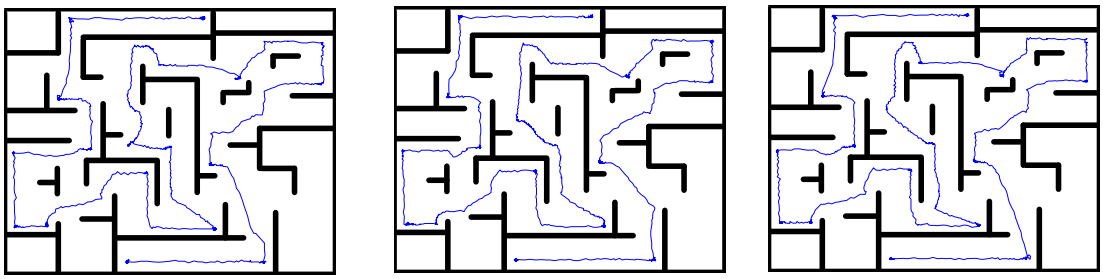

Figure 22: Three observed games played by a human teacher.

To Corner-RTC**, Roll Off Wall**-ROW, **Guide**-GUIDE, **Roll From Wall**-RFW, **Corner**-CORNER. The orientation is specified by T, B, L, and R (top, bottom, left and right respectfully). The orientation information is in reference to the $(0, 0)$ position, the bottom-left corner, with positive y going up and positive x going to the right as shown in Figure 21. SM_x, SM_y, SM_x, SM_y, SB_x, and SB_y specifies the state of the environment, $(Mx, My, \dot{M}x, \dot{M}y, Bx, By)$, at the start of the primitive. EM_x, EM_y, $E\dot{M}_x$, $E\dot{M}_y$, EB_x, and EB_y specifies the state of the environment at the end of the primitive. Distances are in centimeters, velocities are in centimeters per second, and rotations are in radians.

To create a primitive database four games performed by a human teacher were observed. During these observed games the teacher continues to make progress toward the goal location and never falls into a hole. The observed game shown in Figure 20 plus the three observed games with the paths shown in Figure 22 were used to create a primitive database. The database consists of 483 primitives data points; **Roll To Corner**-139**, Roll Off Wall**-69, **Guide**-147, **Roll From Wall**-13, **Corner**-115.

CHAPTER V

LEARNING ONLY FROM OBSERVING

I applied my heart to what I observed
and learned a lesson from what I saw:
- Proverbs 24:32 (NIV) [96]

and observe what the LORD your God requires: Walk in his ways, and keep his decrees and commands, his laws and requirements, as written in the Law of Moses, so that you may prosper in all you do and wherever you go,
- 1 Kings 2:3 (NIV) [96]

When robots use the "Learning from Observation using Primitives" framework they are told *what* actions are possible and have the ability to learn *where*, *when*, and *how* those actions are performed from observing the task being performed by a human. The robots then use this information to operate in the task environment. The robots presented in this chapter use the three modules of the framework, Figure 23, to attempt to perform the task as the observed teacher did in the following manner:

1. Primitive Selection: Observe the state of the environment and select a primitive type to use.
2. Subgoal Generation: Compute a subgoal for the selected primitive type.
3. Action Generation: Perform the appropriate actions in an attempt to reach the computed subgoal. When the action ends, the environment will be in a new state.
4. Return to step 1.

This chapter shows how the modules in this part of the framework can be instantiated to perform the air hockey and marble maze tasks. These modules use information obtained from observing the task as presented in the previous chapter. The operations of the primitive selection, subgoal generation, and action generation modules are explained. The algorithms used in these modules are designed to take advantage of data obtained from observing others and from the robot observing its own performance. This chapter does not include the use of the learning from execution module and

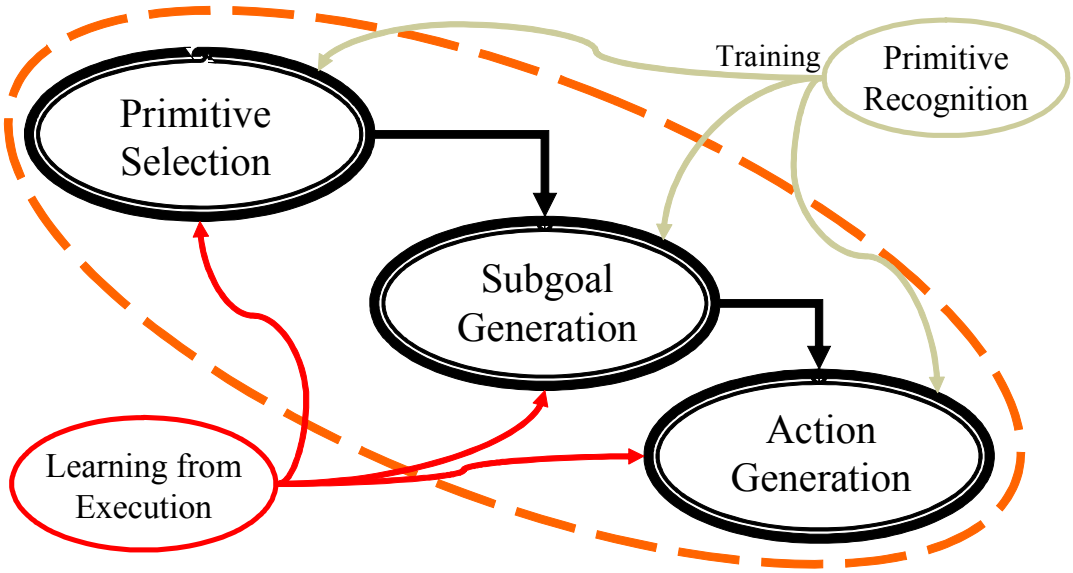

Figure 23: Robots use the primitive selection, subgoal generation, and action generation modules to perform the task in the same way as observed teacher.

therefore the robots described in this chapter learn only from the observed data and do not receive further gain by operating in the environment. The next chapter will present methods that are used by these robots to improve their performance as they perform the tasks.

5.0.5 Choosing a Primitive to Use and Generating Subgoals

In both the air hockey and marble maze environments, primitives are selected and subgoals generated in a similar way. This section presents the basic algorithms used in the primitive selection and subgoal generation modules. The following sections give specific details of how these algorithms are used by robots that operate in the testing environments.

robots that perform tasks using the "Learning from Observation using Primitives" framework must first select a primitive to use. The primitive selection module accomplishes this by first finding the data points in the primitive database that are closest to the observed environment state. By comparing the distance of each data point from the observed environment state, or query point, the distance is $d(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_j w_j \cdot (\mathbf{x}_j - \mathbf{q}_j)^2}$, where $\mathbf{x}$ and $\mathbf{q}$ are the locations of the data point and the query point in state space, and w allows each dimension to be weighted differently. When only trying to act like the teacher, w is a global parameter and remains constant. Section 6.1.5 shows

how the robot learns through practice to compensate for errors in these weights.

The primitive type (a discrete choice) can be chosen by selecting the primitive indicated by the closest data point returned from the nearest neighbor lookup. An alternate approach is to use several nearby points, and implement a voting scheme such as selecting the primitive type that occurs most often within the closest data points. We have found that using the single closest point to determine the primitive type is the easiest and most efficient method.

Once the primitive type has been chosen the subgoal can be computed. It is important to first choose the primitive type because the subgoals of different primitive types may not be compatible. For example, in the marble maze, it would not make sense to use the subgoal of the **Roll Off Wall** primitive with the **Roll Into Corner** primitive. The **Roll Into Corner** primitive will be expecting a corner for the marble to land in as a subgoal location and the **Roll Off Wall** primitive will be specifying a subgoal location at the end of a wall.

The algorithm used in the primitive selection module identifies multiple data points that are in the vicinity of the current state and their distance to the query point. These data point also contains the observed outcome of the human's performance of that primitive. The closest data point information can be used as the desired subgoal. A more robust approach is to use the N closest points that indicate the selected primitive type to compute the subgoal. The outcomes of the returned points are used to compute the subgoal using a locally weighted learning (LWL) approach [10]:

$$\hat{y}(\mathbf{q}) = \frac{\sum_{i=1}^{N} y_i \cdot K(d(\mathbf{x}_i, \mathbf{q}))}{\sum_{i=1}^{N} K(d(\mathbf{x}_i, \mathbf{q}))}$$

$K(d)$ is a kernel function and is typically $e^{-d^2/\alpha}$. The estimate for $\hat{y}$ depends on the location of the query point, $\mathbf{q}$. If N is chosen as 1, it will have the effect of performing the action indicated by the data point closest to the query point. [10] discusses the effect of other kernel functions on the weighting of the data points.

5.0.6 Performing the Primitive

The action generation modules contain a policy to control the robot. There is a separate action generation module for each type of primitive. These modules can use any algorithm that provides the

needed actuator commands to obtain the desired subgoal. We explored the use of various methods including idealized models based on physics, neural networks, and kernel regression techniques [10].

The model based on idealized physics contains a simulation algorithm and computes the required movements to hit the puck or control the marble to a desired location with the desired output velocity. The computed movement is the minimum movement needed to obtain the correct output. In air hockey, paddle velocity that is perpendicular to the normal of the paddle-puck collision does not affect the puck's movement. This method ignores puck spin. Using this model produces extremely accurate results but does not take into account the information obtained from the observation. The accuracy of the model largely determines the results of this method. If a model based on domain knowledge is not available, some other method must be used. For the neural network and kernel regression methods information is extracted from the captured data so as to have the virtual player move the way that the human moved to produce a desired outcome.

The policy used within the action generation module can be locally referenced and generalizable. This will allow the module to be used at multiple locations within the task domain and, if properly formatted, can also be used in similar domains. The state and the policy's specified actions must be transformed as needed. Information needed by the action generation modules is broken into small units that encode state or action outcome information. This chapter shows how this method of breaking the policy into small pieces increases the reuse of learned information.

5.1 Air Hockey

This section shows how robots operate in the air hockey environment using the primitive selection, subgoal generation and action generation modules of the "Learning from Observation Using Primitives" framework. These modules are supplied with training data as described in Section 4.1. The **Bank Shot** and **Straight Shot** primitives are the most difficult. These primitives require future state predictions, precise timing, and accurate movements. The details of the action generation models for shot primitives are described in the next section.

5.1.1 Primitive Selection and Subgoal Generation

Section 4.1.2 described how primitive databases are created from observing a human player. These databases encode the actions taken by the human player when the puck is quickly moving toward the player, the puck is moving slowly within reach, and when the puck is on the side of the board opposite of the player. robots operating in the air hockey environment must use these databases at the appropriate time to select primitives and generate subgoals. This section describes how each of these databases are used by the robot while operating in the air hockey environment. Queries are made to these databases and information is retrieved from them as described above in section 5.0.5.

Figure 24 shows an overview of the actions that a robot takes while playing air hockey. When the puck is on the opposite side and has not yet crossed the decision line, the robot uses the idle database to choose a subgoal location for resting the paddle. The defend and shot primitives are selected in a manner to give the robot time to move its paddle to the shot or defence position at the proper time. After the puck is hit by the opponent, it will take some time for the puck to come within reach of the robot. Having the robot select an action shortly after the puck is hit by the opponent gives the robot time to plan and then take the action. The decision line, Figure 25, specifies one of the times that a shot primitive is considered. If the puck quickly crosses the decision line heading toward the robot, the robot will select a shot type or a blocking maneuver using the fast-shot database. A shot primitive is also selected when the puck is moving slowly on the robot's side. In this situation the robot will choose a shot type using the slow-shot database. Since the puck is moving slowly, the robot will have plenty of time to set up and make a shot.

The simplest behavior is that of the **Idle** primitive. This primitive gives the robot an indication of where it should rest its paddle when the puck is on the other side of the board and no other action is being taken. The database contains the following information:

- Input: The position and velocity of the puck.
- Output: The position of the player's paddle.

A query to this database is made when the robot observes the puck on the opposite side and no other action is taking place. The subgoal information returned from the query is where the robot should move its paddle to.

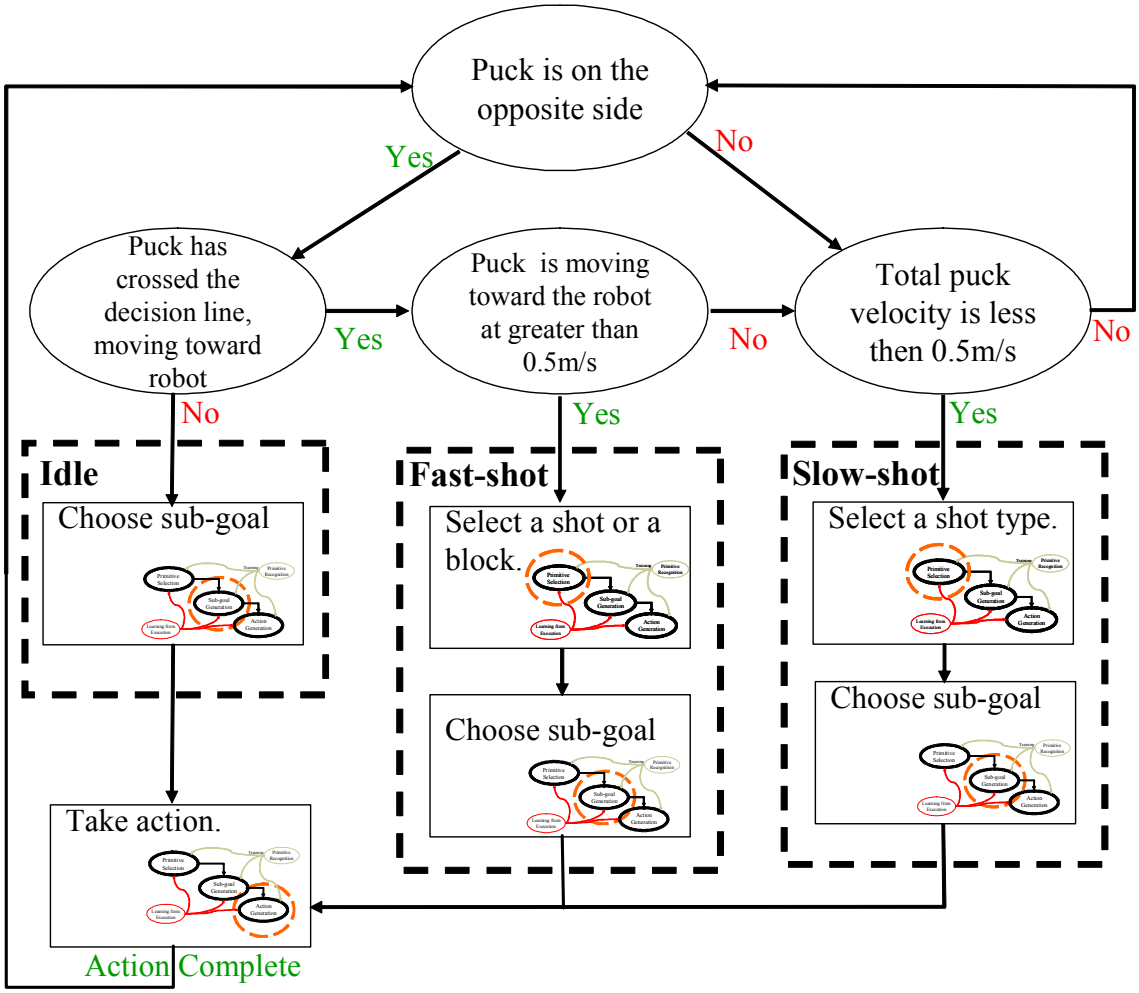

Figure 24: Actions taken by a robot while operating in the air hockey environment.

When the puck is observed crossing the decision line the robot uses a database to select an action. The puck's position and velocity when it crosses the decision line is used as the index for a lookup. This database encodes the primitive type and subgoals for defence and shot primitives. These primitives have different subgoal definitions and therefore the output varies depending on the primitive type selected. The database contains the following information, Figure 25:

- Input:
  - The position and velocity of the puck when it crosses the decision line.
- Output (Shot):
  - The line where the puck will be hit.
  - The absolute post-hit puck velocity.
  - The target location that the puck should go to.
- Output (Block):
  - The position the paddle should be to block the puck.

The slow-shot database is queried when the puck is moving slowly within reach of the robot. This database will tell the robot the type of shot it should take for the observed situation. This database contains the following information:

- Input:
  - Position and velocity of the puck.
- Output:
  - The absolute post-hit puck velocity.
  - The target location that the puck should go to.

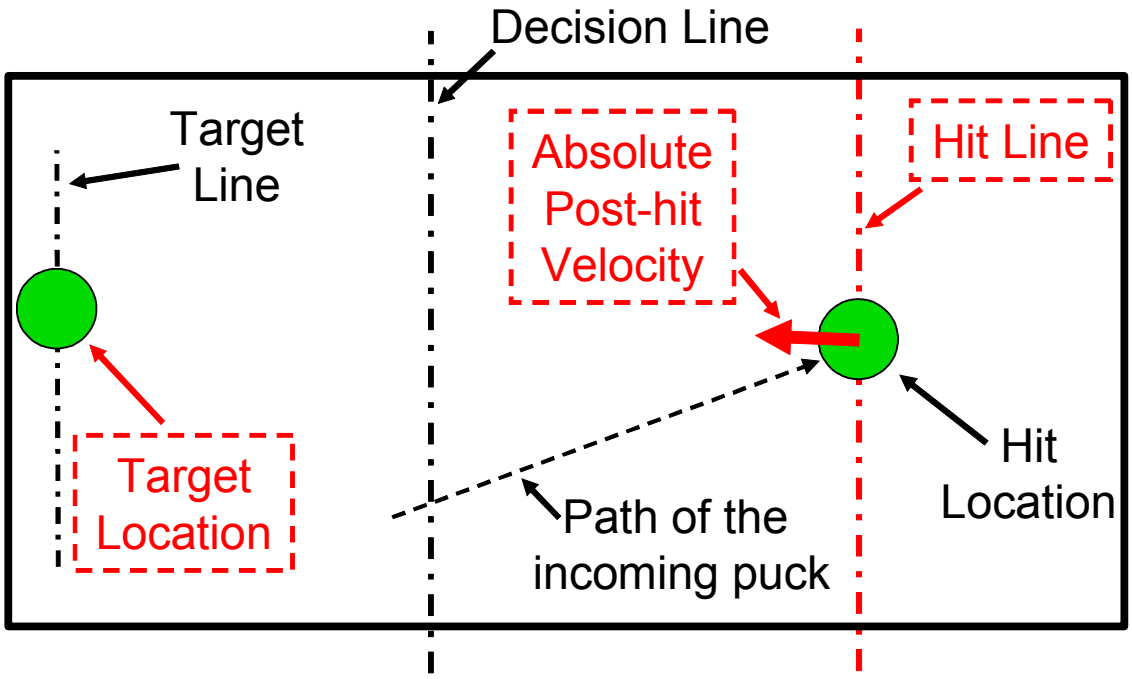

Figure 25: The information returned by the fast-puck database.

5.1.2 Generating Actions

There are four action generation modules that encode the actions needed for each type of primitive; **Idle**, **Block**, **Bank Shot**, and **Straight Shot**. This section describes how the subgoal information is used by these modules to have the robot make the needed movements.

When a shot primitive is chosen by the primitive selection module, the subgoal generation module, using the previously observed information, will specify a line at which the hit should take place, the target location to shoot the puck at, and the absolute desired post-hit puck velocity (Figure 25). The line at which the hit should take place tells the action generation module *where* the action should occur. Because the puck is moving and will only be at the hit line for an instant, it also provides an indication of *when* the action should occur. The target location tells *what* the desired outcome of performing this action is. The target location is not fixed at the center of the opponent's goal but can vary along a line across the back wall as shown in Figure 25. Shooting the puck to a target location can be done at a variety of velocities. How fast (post-hit puck velocity) a player makes shots will have a large effect on their performance at the game. Therefore the absolute desired post-hit

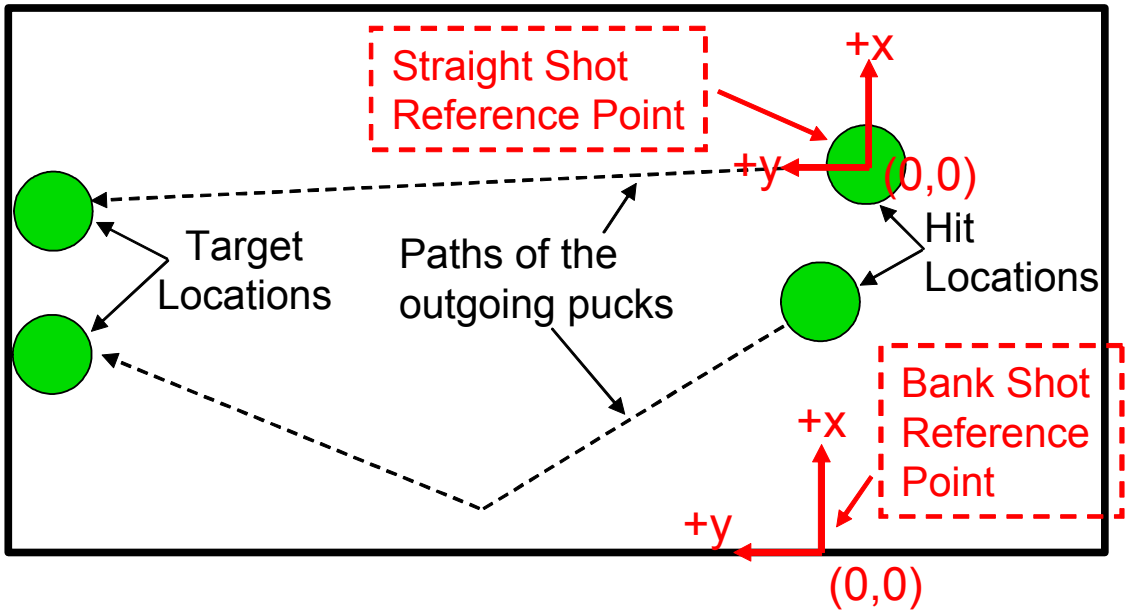

Figure 26: Bank shot and straight shot coordinate frames.

puck velocity is supplied by the subgoal generation module and gives further information of *what* the desired outcome of performing this action is. This information is then input to the appropriate shot primitive action generation module that will control the robot to perform the primitive. The next section shows how we have structured the learning system to take advantage of the observed information as much as possible.

5.1.3 Making the Most of the Observed Data

To increase the usefulness of the observed information, the hockey shot data used for the **Bank Shot** and **Straight Shot** primitives are represented in a local coordinate frame. Using a local representation allows a single **Bank Shot** model to be created from observing both left and right bank shots. The action generation module encodes the robot commands needed to make a specified bank shot in the local coordinate frame. The local coordinate frame for the **Bank Shot** action generation module uses the wall that the puck will hit as the x reference. The reference point $(0, 0)$, shown in Figure 26, is where the puck hit location lines up with the side wall. The x coordinate is positive in the direction from the wall toward the puck and the y coordinate is positive in the direction of the opponent's goal. When a left or right bank shot is observed, puck and paddle parameters are

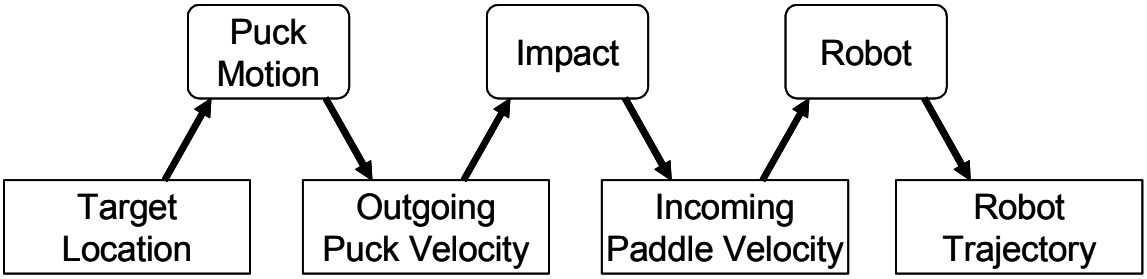

Figure 27: The transformations involved in action generation.

transformed into the local coordinate frame within the action generation module.

In the action generation module for the **Straight Shot** primitive the observed information is transformed to a local coordinate system that is centered on the position at which the puck is hit, also shown in Figure 26. Positive y is toward the opponent's goal and positive x is to the right. This will generalize shots that have approximately the same incoming velocity vector and a similar target displacement and post-hit puck velocity.

5.2 Air Hockey Shot Action Generation

A shot is made by moving the paddle so that it makes contact with the puck. The puck's movement will be influenced at this point of contact [99] and can therefore be controlled by the paddle. A robot trajectory leads to the paddle hitting the puck, and subsequent puck motion that causes the puck to go to the target location. An action generation module must invert this process, and find a robot trajectory that causes the puck to hit the target location (Figure 27). Methods to learn the *Puck Motion* and *Impact* models for the **Straight Shot** and **Bank Shot** primitive action generation modules are presented in this section. The construction of the *Robot* model is also described.

We use a locally weighted learning technique, Locally Weighted Projection Regression (LWPR) [129], to represent these learned models. The *Puck Motion* and *Impact* models are implemented as LWPR models that are trained from the observed information. These models are used during action generation in the **Straight Shot** and **Bank Shot** primitives.

5.2.1 Learning the *Puck Motion* Model

Section 5.1.2 and Figure 25 describe the information supplied by the subgoal generation module when a shot primitive has been selected. This information provides a line at which the puck hit should occur, but the point at which the puck will be hit must be computed using the puck's incoming trajectory. The puck's state shortly after it is hit by the opponent is observed and the location of the puck when it crosses the hit line is predicted (the *Puck Motion* model described in this section does not refer to this prediction). With the addition of this predicted information, the location of the puck when it is to be hit (hit location), the desired absolute post-hit velocity, and the location on the back wall (target location) that the puck should hit is known. What is needed is the direction of the puck's outgoing velocity vector that will accomplish this action. To obtain this information we used the data obtained from observation to train one LWPR model for the **Straight Shot** primitive and another for the **Bank Shot** primitive. The inputs to these models are as follows:

- **Straight Shot** *Puck Motion* model:
  - Target x position.
  - Target y position.
  - Absolute desired post hit velocity of the puck.
- **Bank Shot** *Puck Motion* model:
  - Target x position.
  - Target y position.
  - Puck x position at the time it is hit.
  - Absolute desired post hit velocity of the puck.

The specified locations are in the local coordinate frames of the primitives as discussed in Section 5.1.3. Therefore the puck for the straight shot always starts at the origin and the target is the desired x and y displacement. For the bank shot the puck always starts at the line $y = 0$ and the y value in the target is the desired y displacement of the puck. The output of both models is the

direction in which the puck should travel to reach the given target point from the location the puck will be hit at. This information, along with the desired absolute post-hit velocity provided by the subgoal generation module, is then used by the *impact* model to generate the paddle parameters needed to hit the puck to cause it to have the correct post hit velocity (magnitude and direction).

5.2.2 Learning the *Impact* Model

The *puck motion* model supplies the needed post-hit velocity direction for the puck to reach the target location from the hit location. The *impact* model must specify where the paddle should be relative to the puck and the paddle's velocity at the time of the hit. One way to compute the needed paddle state at the hit time is to use a pre-specified model of the interaction such as the one presented by Partridge and Spong [99]. If the model parameters cannot be precisely defined and determined, the model will not be accurate. Partridge and Spong assume that the mass of the paddle is much higher than that of the puck and therefore the paddle's velocity is unchanged by the collision. This does not seem to be the case in our hardware air hockey environment where small changes in the paddle's trajectory can sometimes be observed at the point where it hits the puck. Therefore, in our implementation, the stiffness of the robot is an additional item that has an effect on the outgoing puck's velocity. Defining an accurate parametric model for our humanoid robot that includes all the items that have an effect on the puck's outgoing velocity would be difficult, as is evaluating the significance of each item to determine which items can be ignored. For these reasons we would like to have the robot learn the *impact* model in the same way it learns the *puck motion* model.

The input and outputs of the *impact* LWPR model are as follows:

- Input:

  The x velocity of the puck when it is hit.

  The y velocity of the puck when it is hit.

  The desired puck's absolute velocity after it is hit.

  The desired movement direction of the puck.

- Output:

  The angle between the puck and the paddle at the hit time.

  The movement direction of the paddle when it hits the puck.

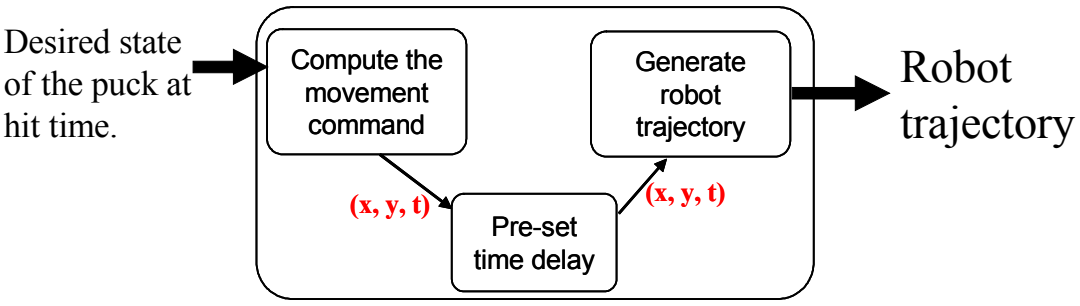

Figure 28: The *Robot* model used to control the movement of the paddle.

The velocity of the paddle when it hits the puck.

The output of the *Impact* model along with the puck's hit position is used to compute the needed state of the paddle at hit time. The position of the paddle at hit time is computed using the puck's hit position and the angle returned from the *Impact* model. The paddle movement direction and velocity returned from the *Impact* model specifies how the paddle should be moving at the time of the hit. The paddle must then be moved at the proper time so that it is at the correct hit position with the correct velocity at the correct time. It is the function of the *Robot* model to achieve this state.

5.2.3 The *Robot* Model

The previous sections shows how the puck state for the desired shot is computed. The paddle is moved by specifying a trajectory from a starting position to an end position and the time to make the movement. The paddle has zero velocity and acceleration at the beginning and end of the trajectory. This trajectory definition gives the paddle a maximum velocity at the center of the trajectory and this is where the hit should occur. In the software environment the paddle is controlled directly and the rendered robot moves as needed to appear natural. But in the hardware implementation the robot must be moved to position the paddle. Section 3.1.3 describes how the humanoid robot makes the needed movements within the air hockey environment using six set configurations.

Figure 28 shows the robot model that is used to control the movement of the paddle. Using the desired state of the paddle at hit time and current paddle position, a paddle movement command is computed. The paddle must be moved at the proper time to be in the hit state when the puck reaches

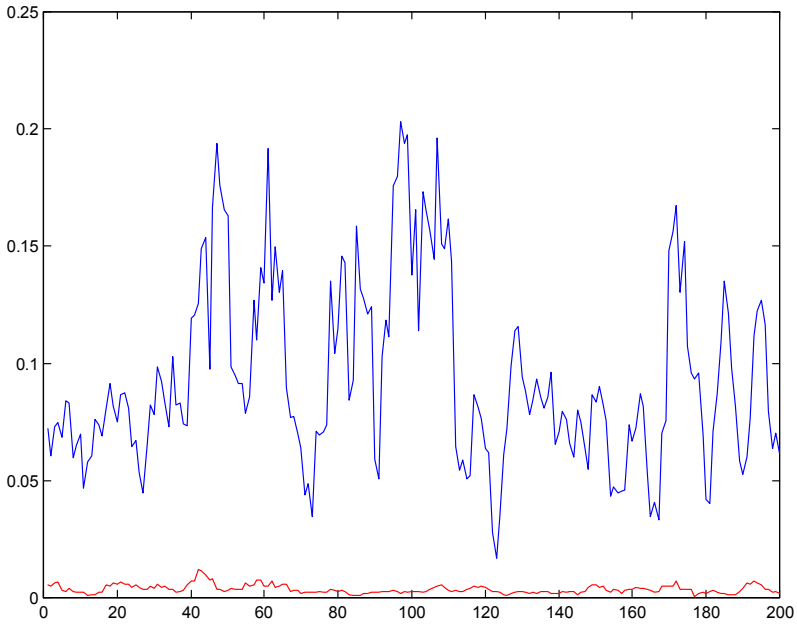

Figure 29: This graph shows the absolute error in reaching the target location during 200 straight shots made by the robot in the software air hockey environment. The solid (top blue) line shows the result of the robot making 200 shots using the LWPR model trained from observing 44 straight shots performed by the human. The dotted (bottom red) line is the result of an robot making straight shots using an exact model of the environment. The graph shows the running average of 5 shots.

the hit position. The robot model also computes this time using knowledge of the trajectory and puck prediction information. With the robot hardware, there are delays in sensing the puck position, computing the puck state, and sending commands to the robot. The total delay is approximately $80 - 110ms$ and the command is sent to the robot earlier to make up for this delay. This delay is not present in the software environment.

5.2.4 Model Learning in the Simulator

The robot in simulation observed 44 straight shots and 108 bank shots while watching a human play simulated air hockey for approximately ten minutes. This information was used to create *impact* and *puck motion* models that are used by the action generation modules of the **Bank Shot** and **Straight Shot** software air hockey playing robot.

The solid (top) line in Figure 29 shows the result of the robot making 200 straight shots in the simulator using models created from observing the human's shots. Figure 29 plots the average

absolute error in hitting the target location, the distance between the target location and the location where the puck actually hit the back wall. The values plotted are the running average of 5 shots. The dotted line at the bottom of the graph shows the results of the robot performing the action using an exact model of the simulator. The error in the exact model is due to the noise introduced into the simulator and this is effectively the best the robot can perform.

The width of the goal is $20cm$ and from Figure 29 it can be seen that if the robot was targeting the center of the goal it would be in range to enter the goal most of the time. But by comparing these LWPR results with the results of performing the action using an exact model of the simulator, we see that it could perform better. Section 6.2.1 describes methods that are used to allow the robots to increase performance as they operate in the environment.

5.2.5 Hardware Air Hockey Performance

Figures 30 and 31 show the performance of the humanoid robot while playing air hockey. Figure 30 shows the path of the puck and the paddles during three intervals of two seconds of game play. The humanoid robot is on the left and the human player is on the right. Figure 31 shows the position of the puck and paddles plotted against time for 10 seconds of game play. The top graph plots the y position of the object on the same graph and most puck-paddle collisions can easily be seen on this graph. The bottom three graphs in Figure 31 show the x position of the paddles and puck during the same time period. The level of performance in Figures 30 and 31 is based on 20 minutes of observed human play and 10 minutes of practice. The regression parameters for selecting primitives and computing subgoals are as follows: the number of data points used in the regression $(N) = 5$, the dimensions are scaled to ± 1.0 and the weight of each dimension is 1.0, and the kernel function is $e^{-d^2/\alpha}$ where $\alpha = 1.0$.

5.3 Marble Maze

This section shows how robots operate in the marble maze environment using the primitive selection, subgoal generation, and action generation modules of the "Learning from Observation Using Primitives" framework. These modules are supplied with training data as described in Section 4.2.

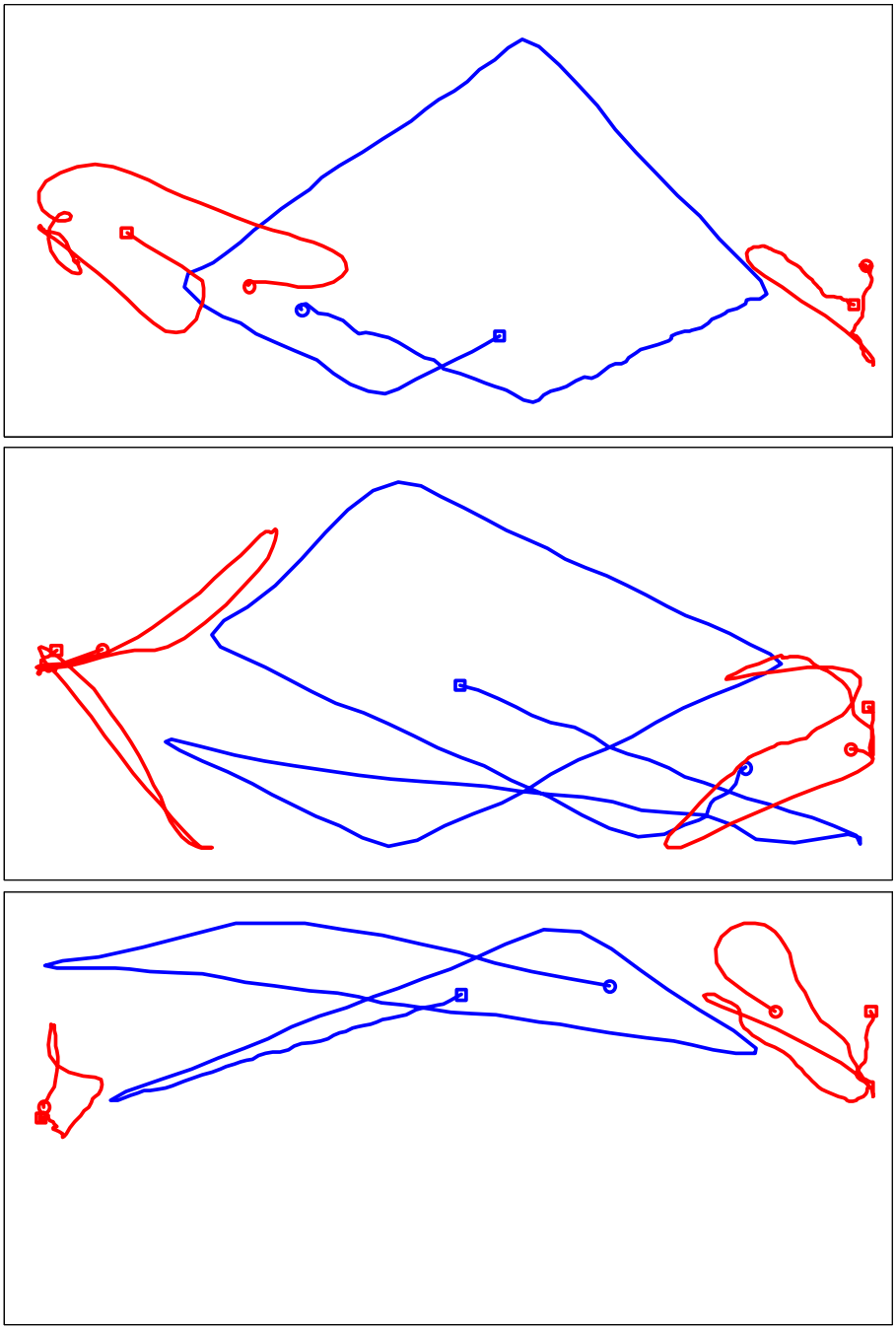

Figure 30: The paths of the puck and paddles during three 2-second intervals of game play with a humanoid robot (left) and a human (right). The "◯" symbol denotes the start of the path and the "□" denotes the end of the path.

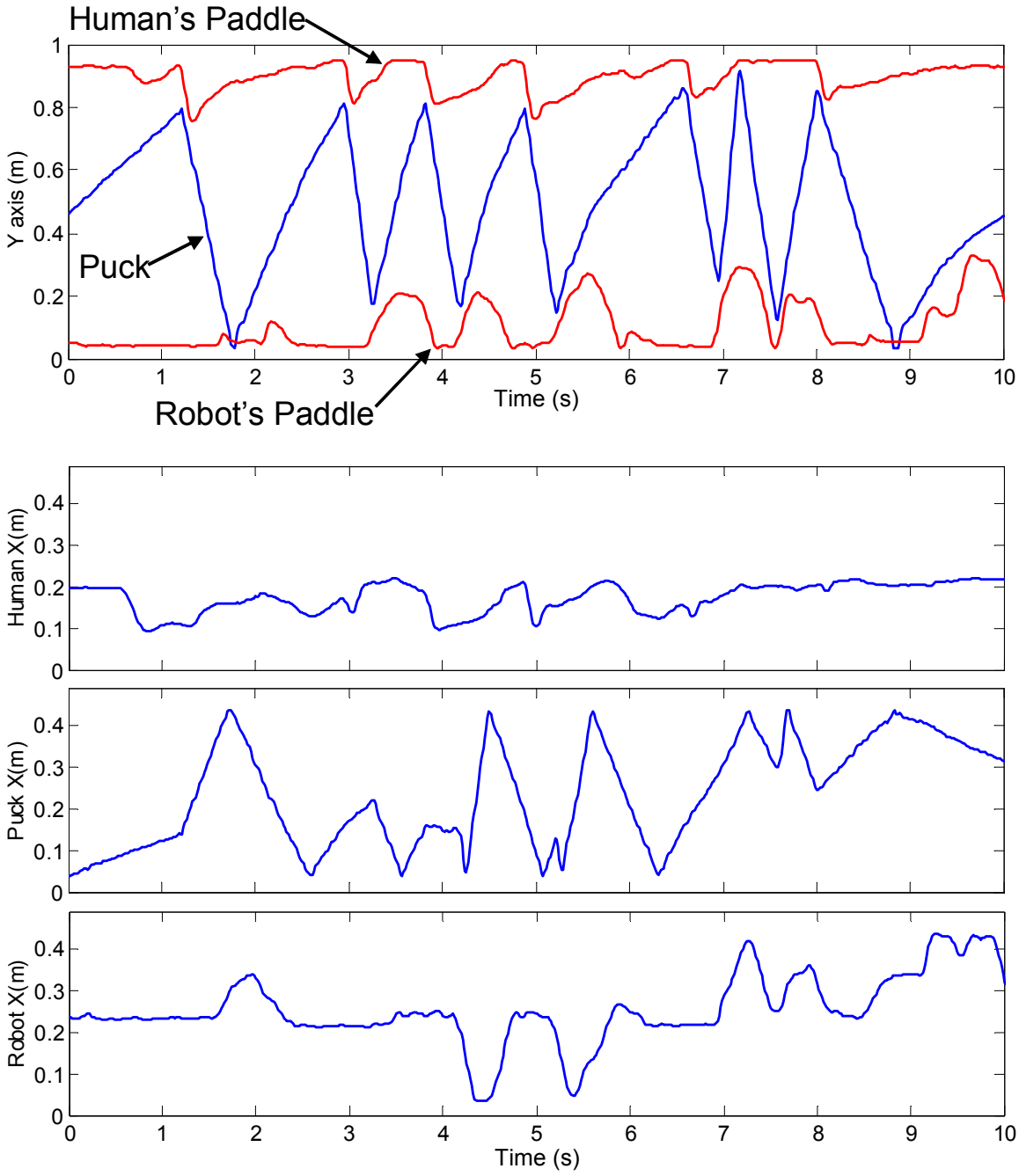

Figure 31: The position of objects in air hockey plotted against time. The top graph shows the y position of all objects on the same graph. The bottom three graphs show the x position of the objects during the same time.

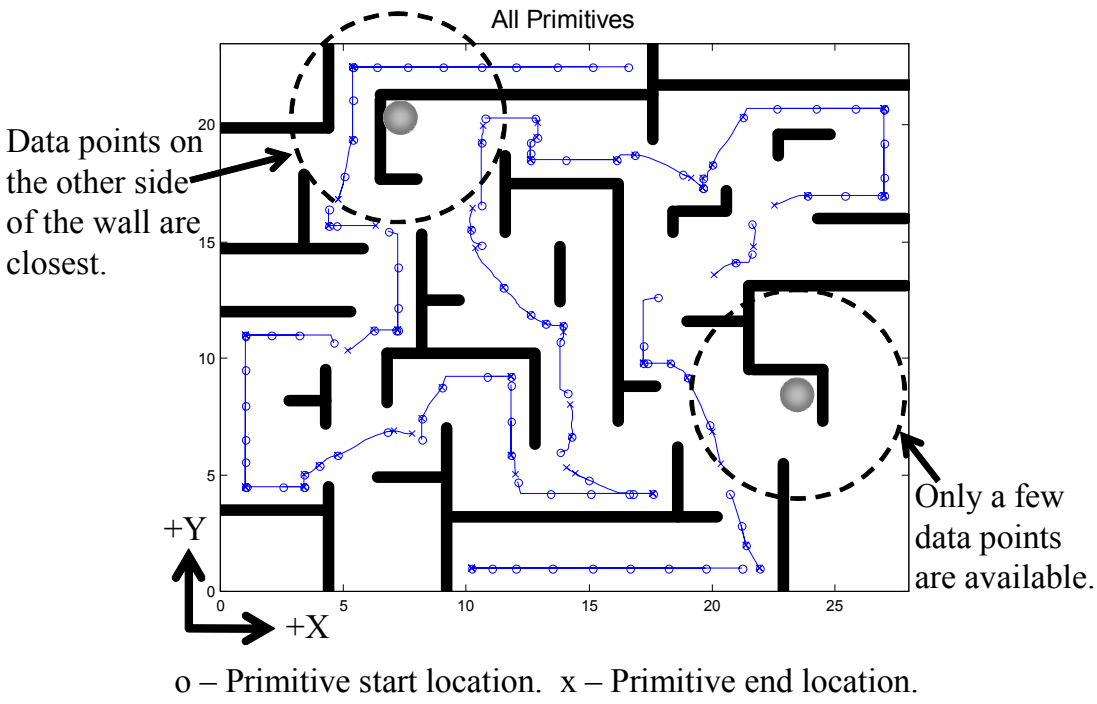

Figure 32: Primitives recognized from observing the task performed by a teacher.

5.3.1 Choosing the Primitive Type

The primitive database for the marble maze consists of the actions that an observed teacher performed while operating in the task environment. This information indexed by the state of the environment at the start of the action; marble's position (Mx, My) and velocity $(\dot{Mx}, \dot{My})$, and board tilt angles (Bx, By). This database is used by the primitive selection and subgoal generation modules to compute the information needed by the robot to perform the task (Figure 23).

In the marble maze task, we consider only actions that the observed teacher performed in the area near the marble's current location. When selecting the closest data points, those that are farther then $4cm$ from the lookup position, (Mx, My), are ignored. Figure 32 shows the primitives found while observing a single game. The number of data points available within a $4cm$ range varies throughout the board. If the robot does not perform the task in the same way as the teacher, the marble may go to locations that are far from the actions performed by the teacher. The location $(23, 8)$ is an example of this problem. If the marble is in this location, the robot has only one or two data points to choose from. In some locations the closest data points are on the other side of

the wall from the marble (location $(7, 20)$). In this situation, the robot will choose actions that were performed in an unrelated part of the maze and the performance will likely not be good.

Multiple task performances are observed to give the robots more information to choose from. The natural variability in the task performances of the human help to provide the robot with a variety of possible actions. The next section describes the rules used by the robot for selecting primitives while performing the task.

5.3.1.1 Primitive Selection rules.

The primitive selection module selects which primitive type to perform for the observed state using the primitive database. Section 5.0.5 showed how Euclidean distance can be computed for each of the data points and how the primitive type to be used can be selected using this information. If only this simple selection scheme is used, the robot can get into trouble. For example, the selection scheme may pick the **Corner** primitive even though this primitive has just been performed resulting in no progress. The robot may also choose actions that were performed in areas that the marble has already passed through. The marble will go backwards through the maze or fall into a hole.

How much information about the task should be given to the robot to help it make intelligent selections? Some possible pieces of information include: Do not choose actions that have occurred on the opposite side of a wall. Only choose actions that make progress through the maze. Do not choose actions in which the subgoal has already been reached. Additional context information will be needed such as the direction in which the marble should travel to make progress and knowledge of the spatial relationship between the query point, walls, and relevant data points.

The primitive selection module for the marble maze uses only two pieces of information to help it select which primitive type to use. One piece comes from the description of the **Corner** primitive which states "the marble is captured in a corner and then the board is positioned in preparation to move the marble from the corner location." This description specifies that a different action should be performed after the **Corner** primitive has been performed. Therefore at the completion of a **Corner** primitive, the primitive selection module must choose some other type of action. The other piece of information used is encoded in the data points. The data points have a start and end location. This information can be used in many ways to help decide if the selected action should

be performed. The robot's objective is to try to obtain the goal state, or end location, specified by the chosen data points. The robot uses the marble's current location along with the data point's end location information to compute how far the marble is from completing the primitive being considered. The primitive selection module will not choose actions in which the marble is already very close to the end location (less then one half the marble radius). Using the computed distance and the primitive selection rules, the primitive selection module provides a list of possible data points that can be used to select the primitive type and compute the subgoal as described in the next section.

5.3.2 Computing the Desired Subgoal

As described in section 5.0.5, N closest points of the same primitive type are used to compute the subgoal using a locally weighted learning (LWL) model[10]. N should be constant because there may be only a few data points available for the robot to choose from. If N is set too high it will greatly limit the actions that are available to be selected because it cannot select actions that have been observed less than N times. It may also be that the correct action to perform for the observed environment state may have only been performed once by the observed teacher. Therefore only one or two data points may be available to the robot. For this implementation, N specifies the maximum number of data points to use. The number of data points used will therefore vary between 1 and N, where N has been set to the number of games observed.

5.3.3 Generating Actions

We now know the primitive and desired subgoal. The robot must now perform the primitive to move the environment from the current state to the desired subgoal state. These actions are performed by the action generation modules which are specific to each primitive type. They are created using the primitive definitions and other provided domain knowledge. This section will describe the creation and operation of these modules for the marble maze primitives. It is important to note that these are independent control policies. The primitive selection process does not take into consideration all the conditions that may cause a primitive to fail. It is also not fully aware of the capabilities of the action generation modules. Therefore the primitive type selected or subgoals generated may not actually be able to be performed by the action generation module and the next chapter presents

methods the robot uses to handle this situation.

5.3.3.1 Action Generation Rules

The function of the action generation modules is to take the environment from one state to another for a constrained context. For example the **Roll Off Wall** primitive can only be used if there is a wall present for the marble to roll off of. Therefore rules have been created that determine when these primitives can be initiated and when they should end. This section discusses the primitive initialization and ending rules summarized in Table 2. For the **Roll Into Corner**, **Roll Off Wall**, and **Roll from Wall** primitives to begin the marble must be within $1.5 \times d_m$(diameter of the marble) of the wall. The wall's position is known by a combination of the orientation of the selected primitive type and the subgoal location.

The marble must be within a distance of $2.0 \times d_m$ of the subgoal location for the **Corner** primitive to be initialized. The **Guide** primitive uses the desired subgoal velocity to decide if it should be performed. The subgoal velocity information is compared with the movement vector, the vector from the current marble position to the subgoal position. If there is more then 45° between them, the **Guide** primitive will not be initialized.

All the primitives, with the exception of the **Corner** primitive, move the marble through the maze. The chosen data points show the general direction that the marble moved when the primitive was performed by the observed teacher. This direction can be compared with the desired direction of movement computed using the current and subgoal locations. If these directions are not within 60° of each other, the primitive cannot be initialized. This check helps to ensure that the robot does not choose primitives that will cause the marble to move to subgoals that have already been passed.

With the exception of the **Corner** primitive, all the primitives will end when the marble is within $0.5 \times r_m$ of the subgoal location or will reach the subgoal location within the next time cycle. The **Corner** primitive must also have the board in the subgoal orientation to end. When a primitive is initialized, the marble's current location and the subgoal location are used to create a bounding box and the primitive is terminated if the marble goes outside this box more than r_m. The objective of the **Roll from Wall** primitive is to guide the marble along a wall and guide it off the wall at the specified subgoal location. Therefore this primitive will end if the marble leaves the wall, even if

Table 2: Summary of primitive type initialization and end reasons.

	Primitive Type				
Initialize Conditions	**Roll to Corner**	**Roll Off Wall**	**Guide**	**Roll from Wall**	**Leave Corner**
Marble is near a wall	X	X		X	
Marble is near the corner					X
Movement vector is close to the sub-goal velocity vector			X		
Movement vector is close to the teacher's movement vector	X	X	X	X	
End Reason					
Marble at sub-goal position	X	X	X	X	
Out of bounding box	X	X	X	X	X
Marble and board in sub-goal position					X
Marble has left the wall				X	

the marble has not reached the subgoal location.

The information needed to compute the initial conditions and end reasons can quickly be computed using the current environment state and the desired primitive type and subgoal information. It is important that this calculation occur quickly because if a lot of time is used, there will be less time available to compute the actions needed. If a primitive cannot be initialized, or the end conditions have been met, the action generation module will return NULL (no action will be taken) and the action generation process will end.

5.3.3.2 Obtaining and Formatting Observed Information for Use by the Action Generation Modules

As discussed in Chapter 2, many methods can be used to create action generation modules. This research seeks to obtain and use information from observing task performances to increase the learning rate of the robot. Section 4.2 shows how the observed data is segmented into the defined primitive types and how information is retrieved from the segmented data and formatted for use

by the primitive selection and subgoal generation modules. This segmented data is also used to provide information about how to learn to perform actions in the environment. The actions the human performed and the environment state can be computed from the raw data collected during the observation. By observing the changes in the environment state and the corresponding actions taken by the human, the robot can learn how to make desired environment state changes.

The segmented data identifies the state of the environment when the primitive is started, S_s, and when it is completed, S_e. The observed actions taken during the performance of these primitives moved the marble from the initial state, S_s, to the end state, S_e. Data points are created to encode the actions that were taken by the observed player during this time. The **Roll Into Corner**, **Roll Off Wall**, and **Roll From Wall** primitives all control the marble while it rolls along a wall. Rolling along a wall should be a local activity that has the wall that the marble is rolling along as a reference. In this local reference frame, these primitives move the marble from one location on the wall to another location. They also change the velocity of the marble along the wall. The data points created for a policy for rolling the marble along a wall uses the end location as the reference point, Figure 33. A data point is created for each observation occurrence and contains the velocity of the marble along the wall, the board orientation, the distance to the end point, the velocity of the marble when it reached the end point, and the change in the board rotation that occurs between this and the next observation. Only the board rotation that affects the marble's velocity along the wall is saved. This ignores the small changes in rolling friction against the wall due to the angle of the board that causes the marble to stay on the wall. This scheme also assumes that all walls are approximately the same and rolling the marble in all directions is approximately the same. Using this scheme allows data points to be created whenever the **Roll Into Corner**, **Roll Off Wall**, or **Roll From Wall** primitive is observed.

Data points are created for a policy that is used by the **Guide** primitive in a similar way. The reference point for the **Guide** primitive, Figure 34, is again the end location. For this primitive it is assumed that rolling the marble in the x direction is the same as rolling it in the y direction and that rolling the marble in a positive direction is the same as rolling it in a negative direction. Using these assumptions a policy can be created for displacing the marble in one dimension perpendicular to the rotation axis. For the situation shown in Figure 34, two data points can be created for each

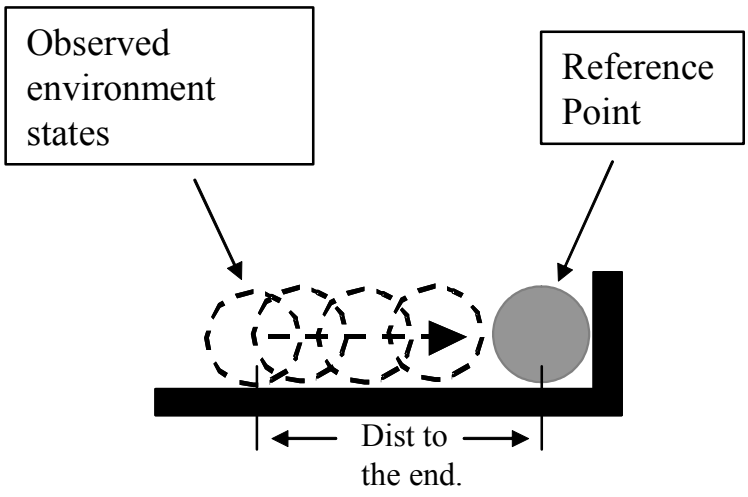

Figure 33: Using observed information to create a database that encodes the actions taken while maneuvering the marble along a wall.

observation occurrence from the start of the primitive to the end. The **Guide** database contains the following information of the marble along a single axis; the velocity of the marble, the board orientation, the distance to the end point, the velocity of the marble when it reached the end point, and the change in the board rotation that occurs between this and the next observation. This method also assumes that the marble rolling in the x (y) direction has no effect on the velocity in the y (x) direction.

The **Roll Off Wall** and **Roll from Wall** primitives must also control the board orientation to cause the marble to roll off the wall with the proper velocity or roll from the wall at the desired location. A database is created for each of these actions in the same way that the guide and roll along wall databases were created. These databases contain the following information; the velocity of the marble along the wall, the board orientation against the wall, the distance to the end point, the velocity of the marble when it leaves the wall, and the change in the board rotation that occurs between this and the next observation.

5.3.3.3 Creating Action Generation Modules

To make the desired changes to the environment, the action generation modules must compute the commands needed to control the robot. Each action generation module, a separate module for each primitive type, contains one or more policies to move the environment from the current state to the

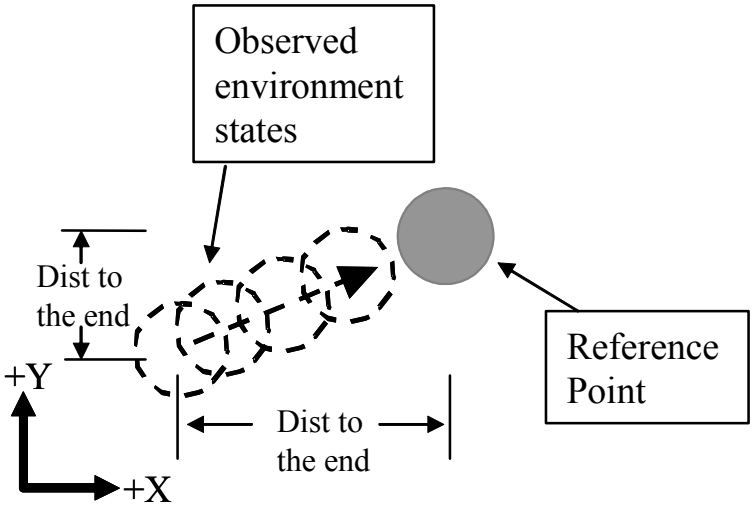

Figure 34: Using observed information to create a database that encodes the actions taken while guiding the marble.

desired subgoal state. These modules perform the following steps:

1. Transform the observed environment state to the local reference frame.
2. If this is the first call of this action generation module, verify that the initialization conditions are met.
3. Check for reasons to end this action.
4. Compute the needed actions in the local reference frame.
5. Transform the action from the local to the global reference frame.
6. Control the robot.

If the module cannot be initialized or any of the end conditions are met, the action generation process will end, NULL will be returned, and no action will be taken. The modules use a variety of methods to compute the needed actions. The **Corner** primitive uses a simple specified policy. The board is given a large tilt to cause the marble to be trapped in the corner. Once the marble is stable in the corner the board is moved close to level and the primitive ends. The other primitives make use of policies created from the observed information to compute the needed actions.

A policy that controls a marble rolling along a wall is created from the database described in the previous section. This policy is used by the **Roll Into Corner**, **Roll Off Wall**, and **Roll from Wall** primitives. This database is used to train an LWPR model to control the robot. The input and outputs to this module are as follows:

- Input:

  The velocity of the marble parallel to the wall.

  The distance to the end of the primitive.

  The tilt of the board perpendicular to the wall.

  The desired velocity of the marble at the end of the primitive.

- Output:

  The perpendicular board movement that should occur during the next time cycle.

Figure 35 shows the performance this LWPR model as it exposed to various amounts of training data. The figure plots the error in obtaining the desired subgoal velocity when the primitive ends and shows how the model's performance increases as it is initialized with more observation data. Each bar represents 30 trials where the robot makes 100 random roll along wall actions with starting conditions and subgoals typical of what is observed when a human operates in the environment. At the beginning of each trial the LWPR model is initialized with data from random observed games.

The LWPR model will provide an output if the its activation level is greater than 0.05. If the activation is not greater than 0.05, model cannot be used and the primitive uses a set policy. If the marble is moving toward the subgoal location with a velocity less than $1.0cm/sec$, the set policy will change the tilt the board slightly ($0.001rads$) to cause the marble to increase its velocity toward the subgoal location. If the marble is moving toward the subgoal location with a velocity of greater then $1.0cm/sec$, no action will be taken. The results of using only the set policy can be seen in the left most bar in Figure 35.

An LWPR model for guiding the marble, used by the **Guide** primitive, is also created in a similar manner. Models are also created to control the parallel board rotation during the execution of the **Roll Off Wall**, and **Roll from Wall** primitives. The **Roll Into Corner** primitive uses a set parallel wall rotation of 0.02 radians to keep the marble against the wall during the primitive.

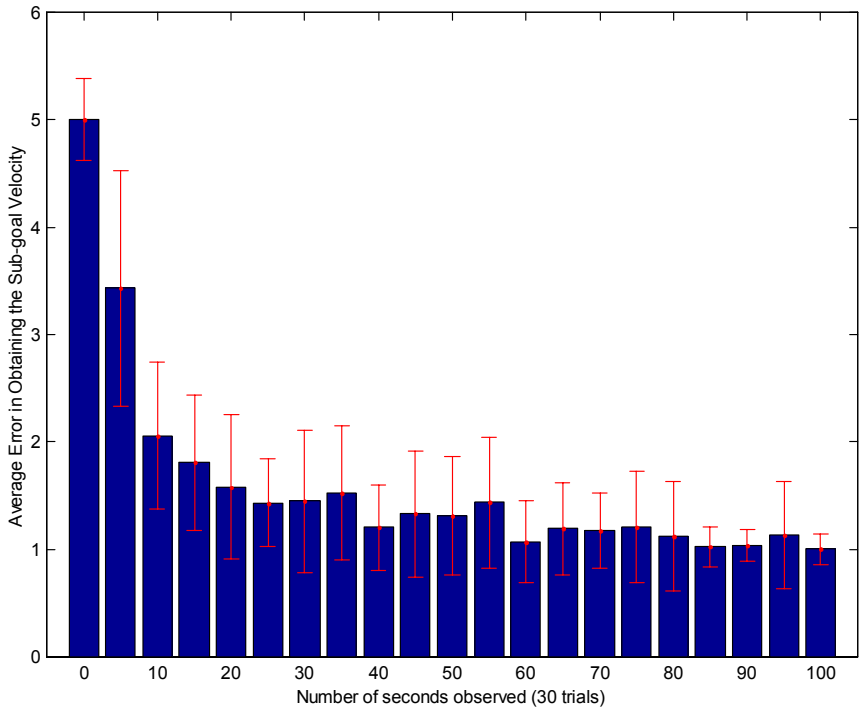

Figure 35: The error of the roll along wall LWPR in obtaining the desired subgoal velocity as it is initiallized with more training data.

5.3.4 Marble Maze Performance Evaluation

Performance evaluation criterion can be used in the marble maze task to compare the performance of different players. Progress through the maze from the start position to the end position is one metric that can be used but this would only evaluate the section of the maze until the marble falls into a hole. Instead we test by placing the marble on the board ahead of the last failure until the board is completed. The players in the software marble maze are given a 10 second penalty for each failure. There are 16 holes on the board but it is possible to fall into the same hole from different locations on the maze so a player has 21 opportunities to fall into a hole. If the marble does not make progress for 15 seconds the player is also penalized by 10 seconds. Using this technique, a player's performance can be evaluated over the entire maze using such metrics as the time it takes to move the marble to the end point or the number of failures that occur during a game.

5.3.5 Results of Learning from Observation

Robots using only the primitive selection, subgoal generation, and action generation modules, along with the training data obtained from the primitive recognition module, have the ability to operate in the observed environment. The results of robots operating in the marble maze environment using these modules are presented in this section. These robots do not use the learning from execution module in the performance of the task. Therefore they do not learn beyond what was observed from the human and the policy used by each module remains fixed. These robots followed four basic steps:

1. Observe the state of the environment.
2. Decide what primitive type to perform; primitive selection.
3. Compute the parameters to use with the selected primitive type; subgoal generation.
4. Perform the primitive until it has terminated; action generation.

The robot's intention is to act as the human did, or as the human would, for the observed state. But if the robot incorrectly predicts the human's action, or cannot correctly perform the chosen action, it has no way of knowing if the outcome is desirable for completing the task.

We programmed a robot to operate in the hardware marble maze using the four observed games shown in Section 4.2. Figure 36 shows the path of the marble while the robot played 30 consecutive games. In each game the robot has 45 seconds to maneuver the marble from the start location to the goal position. The play ends when the time has expired, the marble falls into a hole, or the marble reaches the goal location. Figure 36 shows six graphs each showing five consecutive games, the locations where the marble falls into a hole, and the locations where the marble got stuck. The marble is considered stuck when time runs out. The board is leveled at the start of each game.

Because the policy remains fixed, each game can be considered the robot's first attempt at the game. In other words, the robot has never operated in the environment and has no experience of the effect its actions will have on the environment when it first plays the game. Therefore the difference in the performance from one game to the next is due to the noise within the environment.

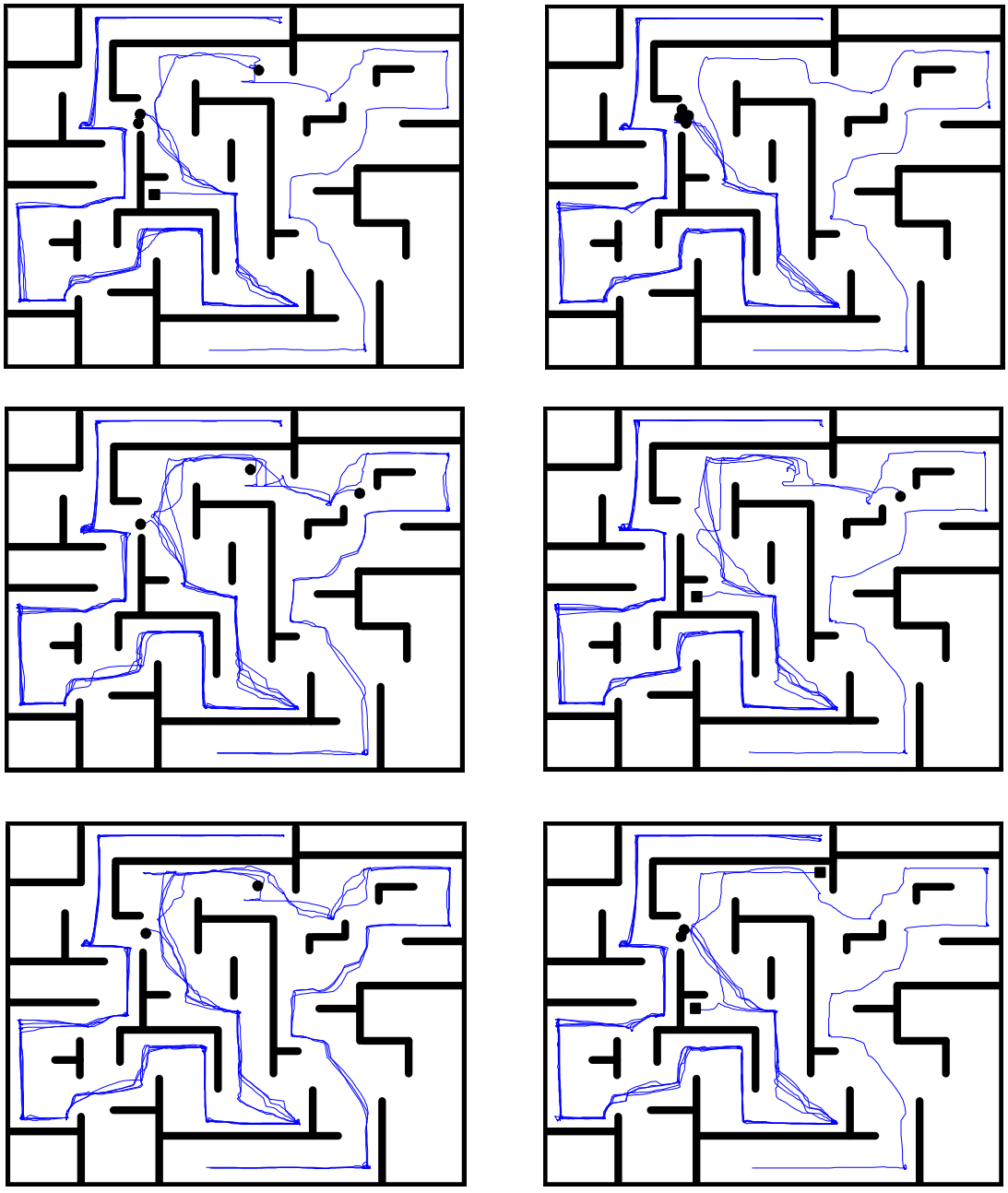

Figure 36: The path of the marble in the hardware marble maze during 30 consecutive games played by the robot using only observed data. Each board shows 5 paths.

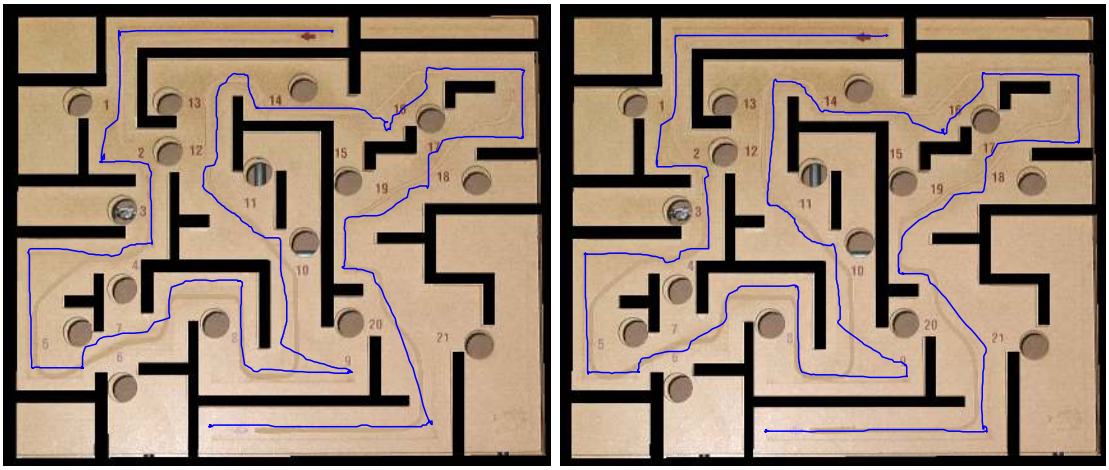

Figure 37: Two of the four paths presented in Section 4.2.4 taken by a human while performing the hardware marble maze task.

The paths shown in Figure 36 show the robot can complete the maze about 3 out of 10 attempts and often falls into the same hole or gets stuck in the same location. By comparing the paths the robot has taken to that of the observed human in Figure 37, we see that the robot has the most trouble when it deviates from the path taken by the human. In these areas the observed actions are all far from the marble's observed position. The robot will only choose the actions that were observed closest to the state the marble is in, but in this situation the chosen actions are not sufficient to get the marble back on track and lead to failure.

The robot has also found a new path that allows the maze to be completed. When the human plays the game they always guide the marble below hole 14 (Figure 37). The robot, on the other hand, only makes progress past this hole by guiding the marble around the top of this hole. Before we observed this action taken by the agent, we did not know this path was possible. In this situation imperfect control has lead to an unexpected discovery.

The robot may deviate from the observed human's path for many reasons. The walls in the environment provide a constraint that helps the robot to maintain the desired path. When the walls are not followed the robot has more difficulty in performing the actions in precisely the same way as the human. The marble may go off the path and into environment states that have not been observed in the teacher's performance. The marble may also deviate due to the differences in the marble's state at the beginning of the primitive. Errors can also occur because the robot may not

be skilled enough to move the marble from its current state to the desired subgoal state. This in turn may leave the marble in a deviated state for the next chosen action. This cycle will continue until the marble falls into a hole or returns to the area of the state space that has been covered by the observed teacher.

We conducted closer analysis of the actions selected by the robot to provide a more in-depth understanding of the inner workings of the learning algorithms. Figure 38 shows the actions and corresponding subgoal locations chosen by the robot during one game in which the marble reached the goal location. The "o" symbol is the lookup or query location. The solid line connects the query point with the associated desired subgoal location indicated by the "$\times$" symbol. The dotted line is the observed path that the marble takes while the primitive was active. Looking at the top left graph of Figure 38 for example, the robot chooses the **Roll to Corner** primitive at the location $(16.6, 1.0)$ and has a computed subgoal position of $(5.4, 1.0)$. During the performance of this primitive the robot observes the movement of the marble from the location $(16.7, 0.5)$ to $(6.9, 0.5)$. The lookup position and the starting position of the marble may not be the same because a predicted future marble state is used for the lookup. The predicted state of the marble three vision cycles into the future is computed using a model of the maze to compensate for the delays in the vision system. By following the path of the marble through the maze in the graphs in Figure 38 for the individual primitives, the sequence of chosen primitives can be seen. For example, from the starting location, the first ten actions chosen by the robot are **Roll to Corner**, **Roll Off Wall**, **Guide**, **Guide**, **Corner**, **Guide**, **Roll to Corner**, **Corner**, **Roll Off Wall**, and then **Guide**. The bottom right graph shows all the chosen primitives combined. This graph is also shown in Figure 39 so the details can more easily be seen.

Figure 39 shows the trouble the robot has in the area of $(9, 16)$ where it usually falls into the hole. The robot is constantly choosing primitives for which the initialization conditions cannot be satisfied. The primitives fail to initialize mostly due to the marble being too far from the wall or having a desired subgoal velocity vector that cannot be obtained from the lookup location. Therefore no actions are taken during this time because the selected action generation modules return NULL when the initialization conditions are not met. As the robot continues to try to find a usable action, the marble is moving in the environment. In the game shown in Figure 39, the marble finally

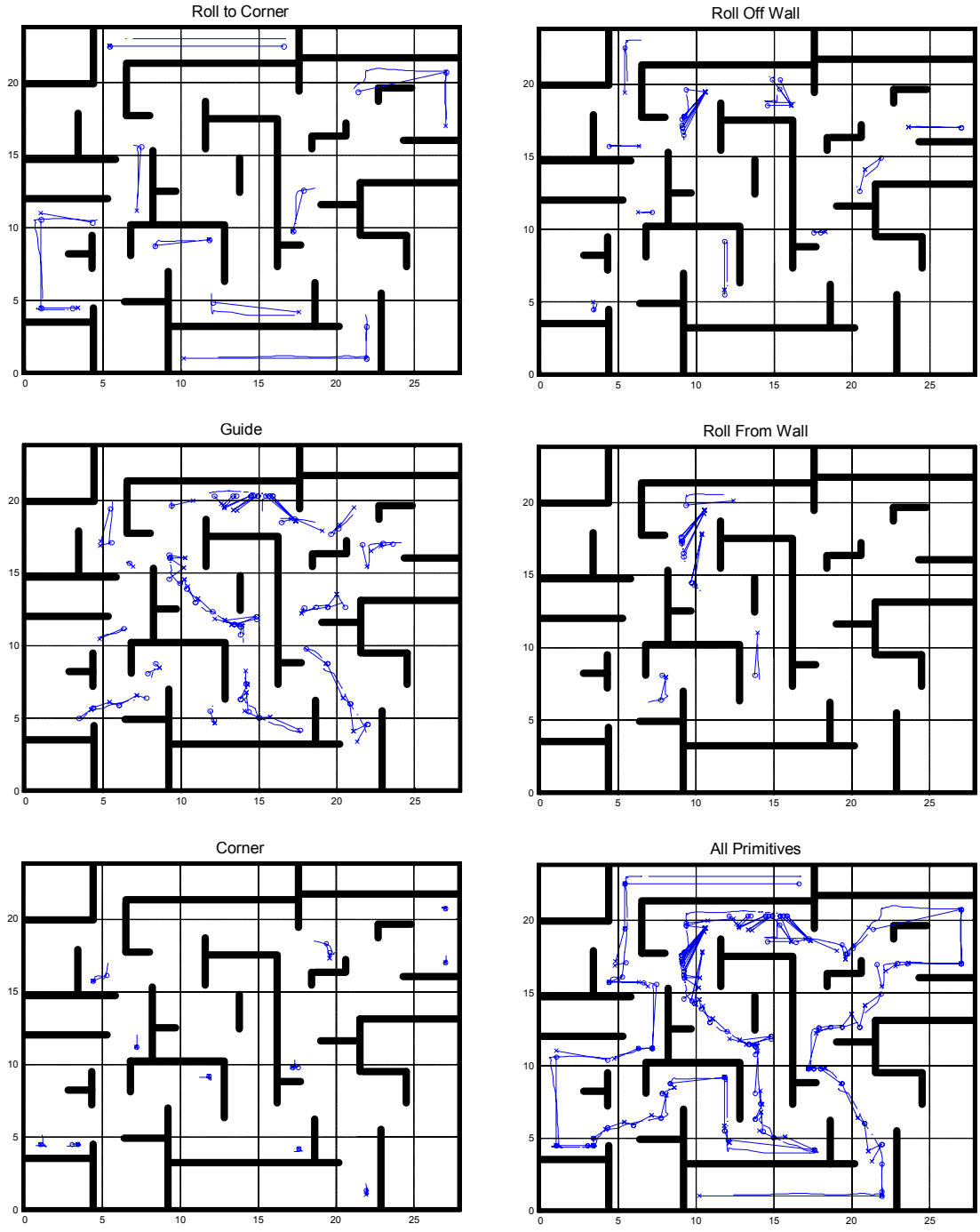

Figure 38: The primitives selected by a robot while traversing the hardware marble maze. The solid line connects the lookup point with the desired subgoal. The dotted line is the path of the marble while the displayed primitives were selected.

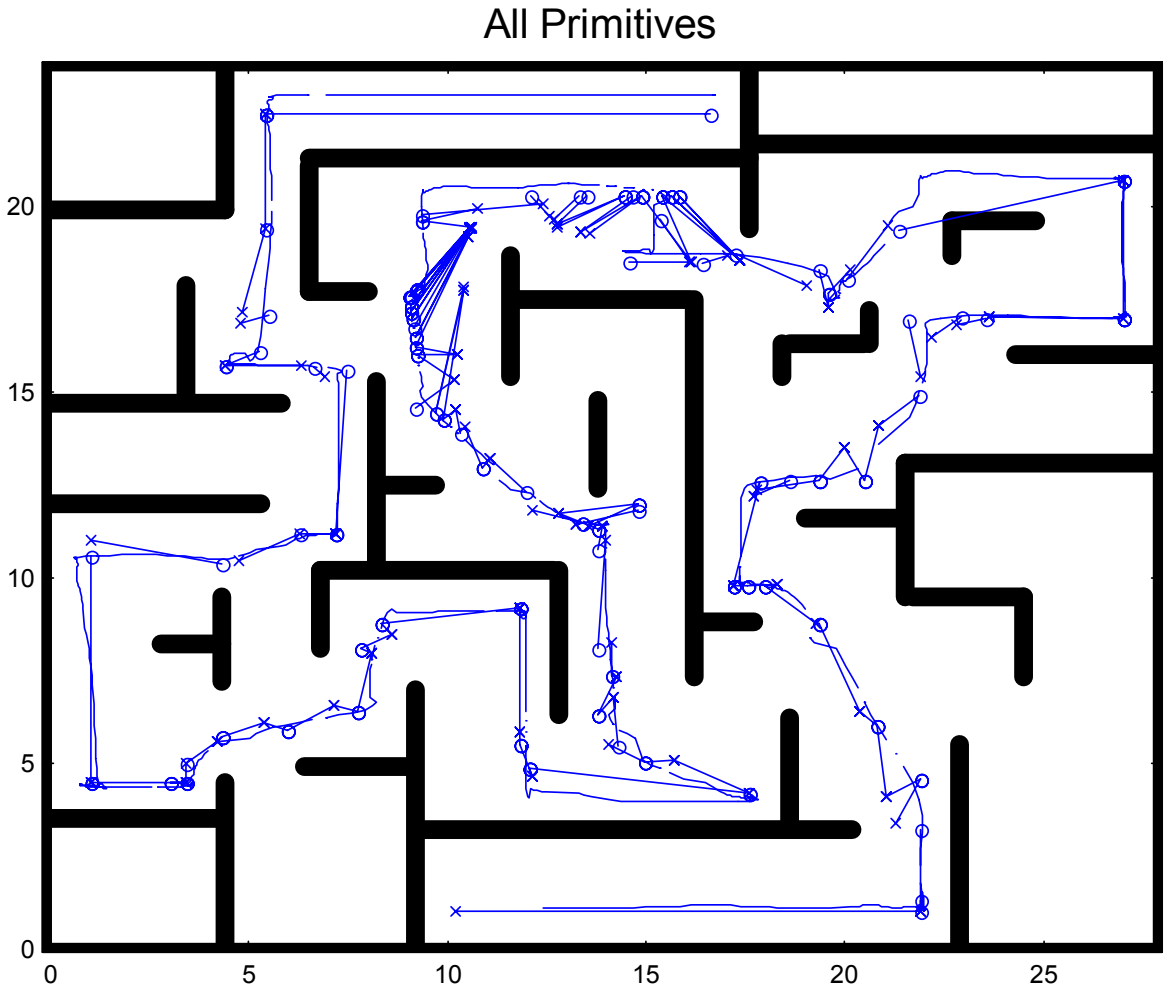

Figure 39: The primitives selected by the robot while manuvering the marble from the start to goal location.

reaches a location from which a usable action can take control. Figure 38 shows that during many of the games the marble reaches the hole before a primitive can be selected to maneuver the marble as needed to make progress in the maze.

A similar situation occurs in the area of the hole at position $(10, 11)$. Before it reaches this area the robot has control of the marble and, during this trial, performs the **Roll From Wall** primitive. The **Guide** primitive then takes over but soon ends because the marble rolls outside the initialized bounding box. The **Guide** primitive failure is mostly due the failure of the previous **Roll From Wall** primitive action to put the environment into the proper state when it completed. As a result, the marble has too high a velocity in the x direction when the **Guide** primitive is initialized. The robot then has problems finding a usable action as the marble again enters states that the observed human was not in. Once again the marble is not controlled as it moves through the maze until a usable action can be selected. The ability of the marble to make it past this area is mostly determined by the state the marble was in when it was last controlled. If the marble has enough velocity in the positive x direction, it has a good chance of making it past this area. However in some games, the marble makes contact with the edge of the hole and is deflected in the negative y direction.

During these trials the marble got stuck four times, Figure 36. The robot fails to make progress past these areas because the chosen primitive action cannot be initialized. In the areas discussed above, when the primitive could not be initialized the marble continues to move and therefore the environment state changes as the robot continues to try to find a usable action. But in these stuck locations the environment state remains the same and the robot continues to select the same primitive and fails to initialize it.

In the software maze environment we can easily conduct many trials. Figure 40 shows the result of a robot playing in simulation after observing three games played by a human in the software environment. The graph on the left shows the number of failures made by the robot during 300 meter runs. A failure is when the marble falls into a hole or does not make progress for 15 seconds. When a failure occurs the marble is placed on the board past the failure location and 10 seconds are added to the time it takes to complete the maze. The run then continues from the new location. The graph on the right shows the overall average time to complete the maze during 5000 second runs. During the observed games the human did not have any failures and had an average completion time

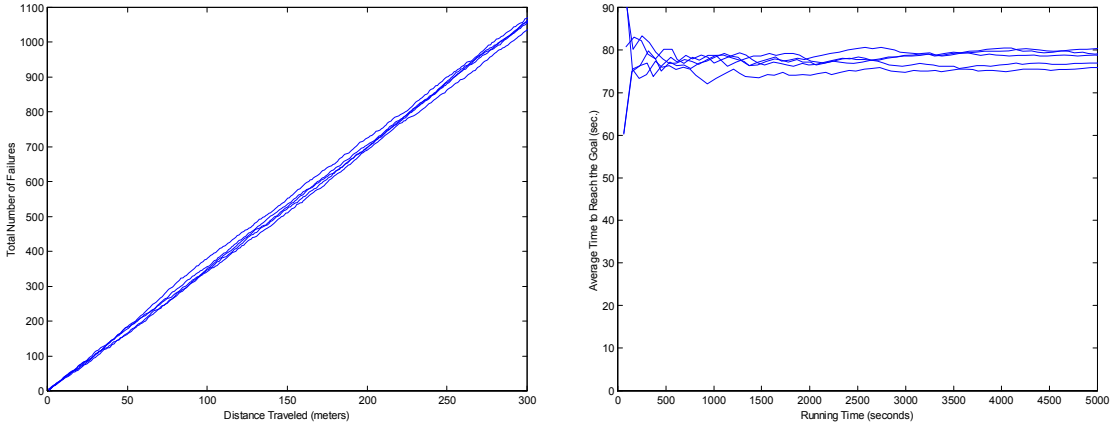

Figure 40: Software marble maze results during five runs. Left: The total number of failures while making 300 meter runs. Right: The average time to complete the maze during runs of 5000 seconds.

of 25.0 seconds (25.0, 25.0, 25.1).

5.4 The Usefulness and Limitations of Only Observing Others

The combination of the domain knowledge and information obtained from observing the task speeds up learning, but also gives the robot clues about when and where to use particular primitives. Due to the physical limitations of air hockey and marble maze, there is a limited set of shots or moves which can actually be made. In air hockey, if the puck arrives in the hit area very close to the right wall for example, a left bank shot would not be possible. Similarly, in the marble maze task the performance of a **Roll to Corner** primitive would fail if there is no corner available to roll into.

Because the robots presented in this chapter do not use the learning from execution module, they do not receive feedback on their performance. But the information provided to the robot up to this point guides it to perform actions and obtain subgoals in the way the observed teacher did. If the teacher performed actions and subgoals that lead to the overall task objective, the robot will attempt to do the same even though, at this point, it has no explicit knowledge of the overall task objectives. In this regard the robot is similar to a child that imitates the actions of others but may not know what the overall goal of performing those actions are.

The result of this limitation in the approach described so far can be seen. The robots quickly

learned how to perform in many parts of the maze from only observing the human and have approximately the same performance from one trial to the next. Differences between trials are due to the noise that is inherent in the physical environment, or introduced into virtual environment. A common error made by the robot playing marble maze is choosing a primitive that it cannot perform from the current state of the environment. This creates a subgoal location that cannot be obtained from the current environment state. The state of the marble evolves without explicit control and the marble would usually just sits in a corner or falls into a nearby hole. The robots playing air hockey have difficulty making good shots because their knowledge of interactions in the environment is limited by the small number of observations. The humanoid robot's ability to accurately place the paddle is affected by many things such as the friction between the paddle and the board and the physical limitations of its mechanical system. Because the robot cannot determine these parameters from observing others, it cannot always produce the desired trajectory.

The results presented in this chapter show that the robots can perform adequately at tasks after only observing a teacher perform the task for a short time. But the robot's performance does not match that of the human and the robots could perform better if they had the ability to change their policy while playing the game. The marble maze playing robots allow the marble to fall into holes more often than the teacher and only rarely complete the maze in a time close to that of the teacher. Comparing the performance of the robot using only observed information and that of the robot using a model of the air hockey environment shows that the robot using learning from observation should be capable of better performance (Figure 29). But to change policies, the robots must have knowledge of the overall task objectives, a way to evaluate its performance with respect to those objectives, and the ability to change its behavior. The next chapter presents methods that give the robots the ability to improve their performance while practicing the tasks.

CHAPTER VI

INCREASING PERFORMANCE THROUGH PRACTICE

"Therefore everyone who hears these words of mine and puts them into practice is like a wise man who built his house on the rock. The rain came down, the streams rose, and the winds blew and beat against that house; yet it did not fall, because it had its foundation on the rock. But everyone who hears these words of mine and does not put them into practice is like a foolish man who built his house on sand. The rain came down, the streams rose, and the winds blew and beat against that house, and it fell with a great crash."
- Matthew 7:24-27 (NIV) [96]

There are many ways that the robots can learn to increase their performance while operating in the task environments. They can become more proficient at performing the primitives. Some examples of robots that learn to increase their performance of primitives through practice include a robot that learns to balance a pole [11] and a robot that learns to grasp objects [64] through practice. The robots can also learn to select more appropriate primitives and primitive parameters for the observed environment state. This chapter presents methods that have been implemented for improving primitive selection, subgoal generation, and action generation through practice.

6.1 Learning to Select Primitives and Parameters

Chapter 5 shows the performance of robots operating in the air hockey and marble maze environments. Using only observed information to create a set policy, these robots often fail at the task because they choose the wrong primitive or generate bad subgoals for the current state of the environment. If these robots had the ability to change their primitive selection and subgoal generation policy while practicing the task, they could learn to select actions that result in improved performance. In this section we present a method that can be used by robots that encode the observed primitive actions in a database as described in Chapter 4, select data points from this database using nearest neighbor techniques, and generate subgoals using locally weighted learning (LWL) techniques. The marble maze task will be used as a test domain to explain the operation of this algorithm.

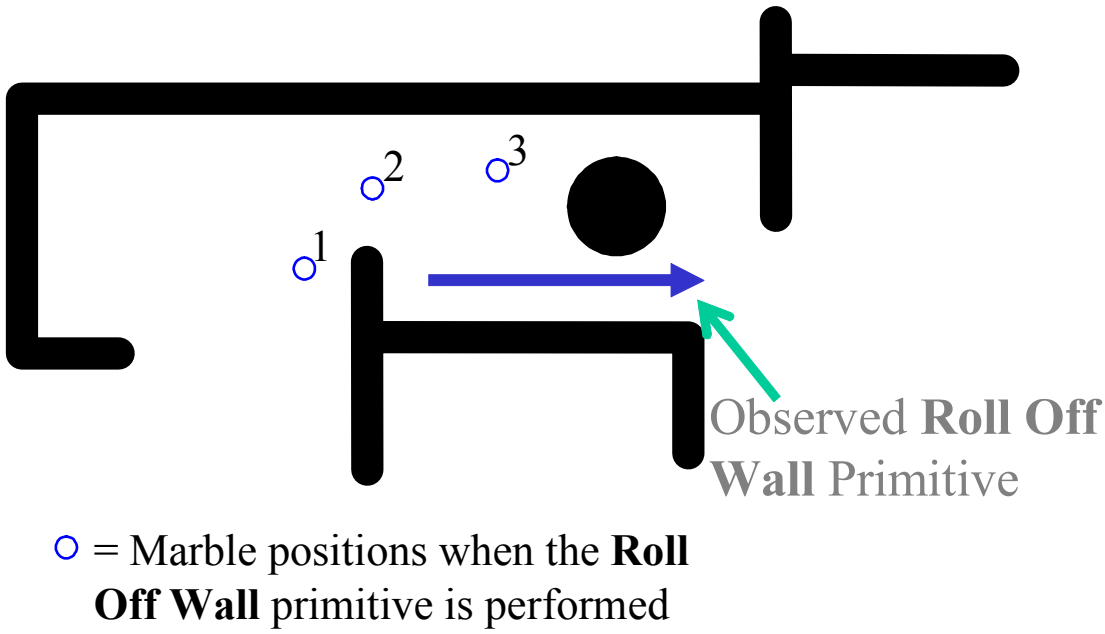

Figure 41: Choosing a **Roll Off Wall** primitive from various nearby locations.

Figure 41 shows an example of choosing a primitive action to perform from various locations nearby an observed data point. The line along the wall shows an observed **Roll Off Wall** primitive action that is recorded in the database. The small circles are various marble positions and the large black circle is a hole. For location 2, choosing this primitive will have a high probability of successfully moving the marble to the subgoal location (the location at the end of the wall). But if this same action is chosen for the two marble locations 1 and 3, the marble is unlikely to make it to the subgoal location. If this primitive is chosen for the marble at location 3, the marble will almost certainly fall into the hole. The **Roll Off Wall** primitive shown expects that there is a wall on the bottom to roll off of. Therefore if this primitive is chosen at location 1 there is no wall to catch the marble and it will move backwards through the maze. The basic primitive selection scheme shown in Chapter 5 cannot discriminate between these situations based solely on the computed distance between the data point and the query points. If the robot had the ability to encode this information into its primitive selection method, when chosen action leads to a failure, a different action can be chosen in the future from this same state.

To obtain this functionality, we combined the LWL algorithm with a reinforcement learning algorithm [122] to give the robot an indication of the value of using data points from an observed

state. The basics of the algorithm are to incorporate a multiplier into the distance function $d(\mathbf{x}_i, \mathbf{q})$. Incorporating a multiplier into $d(\mathbf{x}_i, \mathbf{q})$ has the effect of moving the data point in relation to the query point. A multiplier greater than 1.0 will have the effect of moving the data point further away from the query point and a multiplier less than 1.0 will have the effect of moving the data point closer to the query point. For example, if the marble falls into a hole after a selected primitive is performed, the multipliers associated with the set of data points that were used to decide on that primitive can be increased. The next time the robot finds itself in the same state, those data points will appear farther away and will therefore have less or no effect on the newly chosen action.

6.1.1 Associating Multipliers With Data Points

The naive approach of only associating one multiplier to each data point has a problem because although the chosen data points may be inappropriate for one query point, they may not work very well for another of equal distance (Figure 41). If the same multiplier is used for all query points, the system will not be able to distinguish between the success and failure situations. For example, from query point 1 the robot may choose a more desirable action, but from query point 2, the robot may now choose a less desirable action.

To overcome this limitation our algorithm provides the ability to use a different multiplier for different query points around the data point's origin. Obviously there cannot be an infinite number of multipliers so there must be some way to encode the multipliers. This section presents a simple example to explain the use of these multipliers and an encoding scheme and Section 6.1.4 describes function approximator approaches that have been investigated to encode the multipliers.

Figure 42 shows a simple one-dimensional example where multipliers are associated with two stored data points, P1 and P2. The numbers above the data points are the multipliers that are initialized to 1.0 in the top figure. For each data point, the number to the right (left) of the data point is used when the query point is to the right (left) of the data point. For the two query points shown the distance QP2-P1 is $1.0 * 5 = 5$, QP2-P2 is $1.0 * 20 = 20$, QP1-P1 is $1.0 * 5 = 5$, and QP1-P2 is $1.0 * 10 = 10$. For the situation shown, the data point P1 would be selected for both query points. Assume a robot makes the query QP1 and chooses data point P1 and performs the primitive with the parameters specified by P1. The robot's progress is then evaluated when the primitive

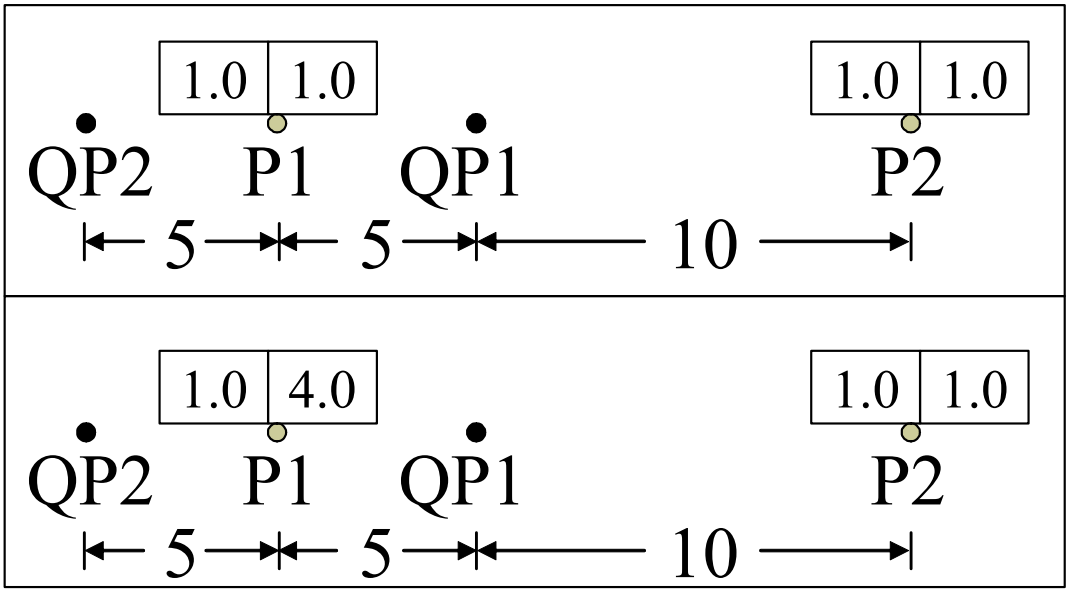

Figure 42: One dimensional example of associating multipliers with data points. The multipliers are initialized to 1.0, top figure. Then the multiplier for P1 is changed to reflect the effect of using this point in query QP1, bottom figure.

ends. If the robot has performed poorly, the multiplier can be changed as shown in the bottom of Figure 42. For this new situation the distance QP2-P1 is $1.0 * 5 = 5$, QP2-P2 is $1.0 * 20 = 20$, QP1-P1 is $4.0 * 5 = 20$, and QP1-P2 is $1.0 * 10 = 10$. The distance between QP1 and P1 now appears much larger and P2 will now be chosen for this query. However, a query at QP2 will remain unchanged. The next section describes how the multipliers are used and updated and Section 6.1.4 shows methods for associating multipliers with a data point.

6.1.2 Using the Multipliers When Selecting Primitives and Generating Subgoals

The multipliers provide an indication of the value of using that data point for the given query. Multipliers that are greater than 1.0 indicate that the robot failed when that action was chosen, and there is a low value, or chance of success, if that data point is used for the state the environment is currently in. Multipliers less than one, on the other hand, indicate that the chosen action worked well for the current environment state. In our implementation the multipliers have been replaced with a number that represents the value of using that data point for the current environment state. A multiplier is then computed using the stored number.

We now use the following equation to compute the distance between the data point $\mathbf{x}_i$ and the query point $\mathbf{q}$: $\hat{d} = d(\mathbf{x}_i, \mathbf{q}) \cdot f(\mathbf{x}_i, \mathbf{q})$. The function $f(\mathbf{x}_i, \mathbf{q})$ retrieves the value of using data point $\mathbf{x}_i$ associated with the query point $\mathbf{q}$ and uses that number to compute a multiplier. As the cell value is increased, $f(\mathbf{x}_i, \mathbf{q})$ must decrease so that the data point's apparent distance is reduced. We tried using C/V where C is the value that the cells are initialized to and V is the current value in the cell. V must then be prevented from going to 0. To eliminate this restriction, we also explored the function $e^{(C-V)/\beta}$. Within this function β controls the amount of discrimination between different values. If C is chosen as 0, the function reduces to $e^{-V/\beta}$.

As seen in the example in Figure 42 above, the multiplier returned by the function $f(\mathbf{x}_i, \mathbf{q})$ has a direct impact on the apparent distance of that data point in relation to the query point. Section 5.3.2 showed how multiple data points of the same type could be combined to compute the subgoal parameters using the following equation:

$$\hat{y}(\mathbf{q}) = \frac{\sum_{i=1}^{N} y_i \cdot K(d(\mathbf{x}_i, \mathbf{q}))}{\sum_{i=1}^{N} K(d(\mathbf{x}_i, \mathbf{q}))}$$

The new distance $\hat{d}$ is now used to compute the parameters in place of $d(\mathbf{x}_i, \mathbf{q})$.

6.1.3 Updating Multiplier Values

When the results of performing a primitive are observed, the values associated with the data points used to select the primitive type and subgoal must be adjusted to reflect the performance. If the values are set in isolation for each primitive performed, the robot will not consider longer term effects - three consecutive primitives that lead to a hole, for example. For the marble at location 3 in Figure 41 there may be no action that can prevent the marble from going into the hole. Therefore the actions that led the marble to this state are at fault and there must be some way to propagate this information back to these primitive selections so they are no longer performed.

The Q-learning algorithm [122] lends itself very well to updating the values while taking into account the results of subsequent actions. The state-action values, or Q values, in the Q-learning algorithm, $Q(s, a)$, provide the expected future reward that can be obtained by taking action a from state s. The results are observed when an action is taken and the values are updated. Rewards are given to guide the values to provide the desired outcome. In our implementation, the Q values,

$Q(\mathbf{q}, \mathbf{x}_m)$, are stored with each data point and provide an indication of the expected reward that can be obtained by selecting data point $\mathbf{x}_m$ for query the point $\mathbf{q}$. The value V, described in the previous section, becomes the Q value. Since the selected data points and their distance from the query point determine the action that is taken, the primitive type to perform and the desired subgoal, the Q values have a direct impact on manipulating the action that is chosen from query point $\mathbf{q}$.

We initialized the Q values with a constant, C, and update them using a modified version of the Q-learning function. Because there are N data points selected by the nearest neighbor search, N Q values must be updated at each time step. For each data point chosen, $\mathbf{x}_m$, $m = 1, N$, the Q values at time t, $Q_t(\mathbf{q}_t, \mathbf{x}_m)$, are updated as follows:

$$Q(\mathbf{q}_t, \mathbf{x}_m) \leftarrow Q(\mathbf{q}_t, \mathbf{x}_m) + \alpha \cdot [r_{t+1} + Q(\hat{\mathbf{q}}, \hat{\mathbf{x}}) - Q(\mathbf{q}_t, \mathbf{x}_m)]$$

- α is the learning rate. Since multiple points are used, the weighting given by $\frac{K(d(\mathbf{x}_i, \mathbf{q}_t))}{\sum_{j=1}^{N} K(d(\mathbf{x}_j, \mathbf{q}_t))}$ is used as the learning rate. This weighting has the effect of giving points that contributed the most toward selecting the primitive the highest learning rate.

- r_{t+1} is the reward observed after the primitive has been performed.

- $Q(\hat{\mathbf{q}}, \hat{\mathbf{x}})$ is the future reward that can be expected from the new state $\hat{\mathbf{q}}$ and selecting the data points $\hat{\mathbf{x}}$ at the next time step. This value is given by:

$$\sum_{i=1}^{N} \left[Q(\hat{\mathbf{q}}, \hat{\mathbf{x}}_i) \cdot \frac{K(d(\hat{\mathbf{x}}_i, \hat{\mathbf{q}}))}{\sum_{j=1}^{N} K(d(\hat{\mathbf{x}}_j, \hat{\mathbf{q}}))} \right]$$

6.1.4 Encoding the Q-values in a Function Approximator

The research of Santamaria et al. [111], Smart and Kaelbling [118], and Forbes and Andre [44] give some insight on various methods that can be used to encode the Q value in a function approximator. Their research also proves the usefulness of using function approximators for the Q value in various domains. We explored the use of two function approximation methods: tables and LWPR models.

We created a table for each data point by quantizing the state space in a small area in the vicinity of the data point. In the marble maze environment, for example, the state space has six dimensions,

the marble's position (M_x, M_y) and velocity $(\dot{M}_x, \dot{M}_y)$ and the board rotation (B_x, B_y). We quantized each dimension into five cells and therefore each data point in the database has a table of size 5^6. For any query point in the state space, its position relative to the data point is used to find the cell that is associated with that query point. Since only a small fraction of the cells are visited, the tables are stored as sparse arrays and only when the value in a cell is initially updated is the cell actually created. For example, if a set of chosen data points provides a good result, those data points will be chosen every time for that environment state. Other data points in the area will not be visited and therefore all the cells that are associated with those data points will not be created.

In order to achieve satisfactory performance increases, effort must be invested into selecting the size of the cells in each dimension. We chose the size of the cells associated with a data point manually through trial and error with the cells near the center being smaller than those further away. For example the difference in x and y position between the query point and the data point is divided into the following areas: less than -0.4cm, -0.4cm to -0.1cm, -0.1cm to 0.1cm, 0.1cm to 0.4cm, and greater than 0.4cm. This system assumes that the data points will be chosen close to their origin in the state space and that making the closer cells smaller will allow a finer distinction of the data point's effect on the outcome. But if the data point is chosen far, in state space, from its origin, it will not be possible to discriminate its effectiveness. Therefore choosing the most effective discretization for each of the dimensions that will work in all areas of the environment may not be possible. To overcome these limitations we also explored another method that encodes the Q value in an LWPR model. The table for each data point is replaced with an LWPR model. As the robot operates in the environment and adds information to the LWPR, the LWPR adjusts to discriminate between query points as needed. The characteristics of the LWPR models allow data points to be added continuously. The LWPR parameters that control the initial size of the receptive fields can more easily be chosen and adjusted. Unlike the fixed cell sizes in the table, the receptive fields within the LWPR will adjust their sizes as necessary to conform to the training data.

Lookups and updates to the tables and LWPR models are performed in the same manner. There are six inputs; the difference between the query point and data point in position, velocity, and board orientation. Using this lookup scheme, values can be retrieved or new values can be input to the function approximators. If there are no receptive fields within the LWPR with an activation of at

least 0.1, the default Q value is used for this environment state.

The Q values for the environment states can change greatly during learning. Therefore the initial and final forgetting factors are set to cause the model to maintain a high weight on the most current data. To further help shape the model more quickly, when updates are performed on the models the new data is fed to the model multiple times, up to 30. The next section shows the performance and inner workings of robots using tables and LWPRs to encode the learned Q values.

6.1.5 Results of Learning to Select Primitives and Parameters in the Marble Maze

We programmed a robot to play the hardware marble maze game using the "Learning from Observation using Primitives" framework. The top picture in Figure 43 shows the training data that was provided to the robot. The robot first performs the task without using the learning from execution module. The middle picture shows ten consecutive paths taken by the robot while performing the task. During this time the robot completes the maze five times and falls into a hole five times. The robot is then given the ability to change the primitive selection and subgoal generation behavior as described in the previous section. In this implementation, the initial cell value, C, is 100,000 and the robot uses the function C/Q to compute the multiplier. The reward strategy is as follows:

- Moving through the maze: $100\times$ the distance in meters from the beginning of the primitive to the end location. Movement along the path toward the goal is positive distance and movement away from the goal is negative distance.
- Taking up time: $-10\times$ the amount of time in seconds from the time the primitive started to the time it ended.
- Not making any progress during the execution of the primitive: $-50{,}000$.
- Not completing the execution of the primitive within the given time of 4 seconds: $-20{,}000$.
- Rolling into, and being stable in, a corner: 30,000. When a corner is reached, learning stops and restarts from the corner location. The robot only gets a reward the first time it goes to a corner. Subsequent visits to the same corner will also reset learning, but will result in no reward. This prevents the robot from returning to corners just to get high rewards.

- Falling into a hole: $-50{,}000$.
- Reaching the goal location in the maze: 10,000.

The bottom picture in Figure 43 shows the path of the marble for ten games played by the hardware robot after it has practiced for 30 games. This robot has completed the maze ten consecutive times without falling into a hole.

We also tested robots in the simulator with and without the use of the learning from execution module to update the primitive and parameter selection policy. This implementation was used to obtain a more thorough look into the performance of our algorithm because it is much easier to run multiple trials. This section first presents the results of using this strategy with the Q values stored in LWPR models. The performance of the table method is then compared with the LWPR function approximator method. Both methods use the same initial cell value, $C = 0$, and $f(\mathbf{x}_i, \mathbf{q}) = e^{-Q/\beta}$ where β =20,000. The reward strategy is as follows:

- Moving forward through the maze: $200\times$ (distance in meters).
- Moving backward through the maze: $-1{,}000\times$ (the distance in meters).
- Taking up time: $-1{,}000\times$ the amount of time in seconds from the time the primitive started to the time it ended.
- Not being able to initialize the primitive: $-30{,}000$.
- Not completing the execution of the primitive within the given time of 4 seconds: $-30{,}000$.
- Rolling into, and being stable in, a corner: 0. When a corner is reached, learning stops and restarts from the corner location.
- Falling into a hole: $-50{,}000$. When the marble falls into a hole, learning is stopped and restarted from the location the marble is placed at when the game begins again.
- Reaching the goal location in the maze: 10,000.

This reward strategy leads the robot to choose actions that make large forward progress in the least amount of time and to not choose actions that cannot be initialized or cause the marble to fall

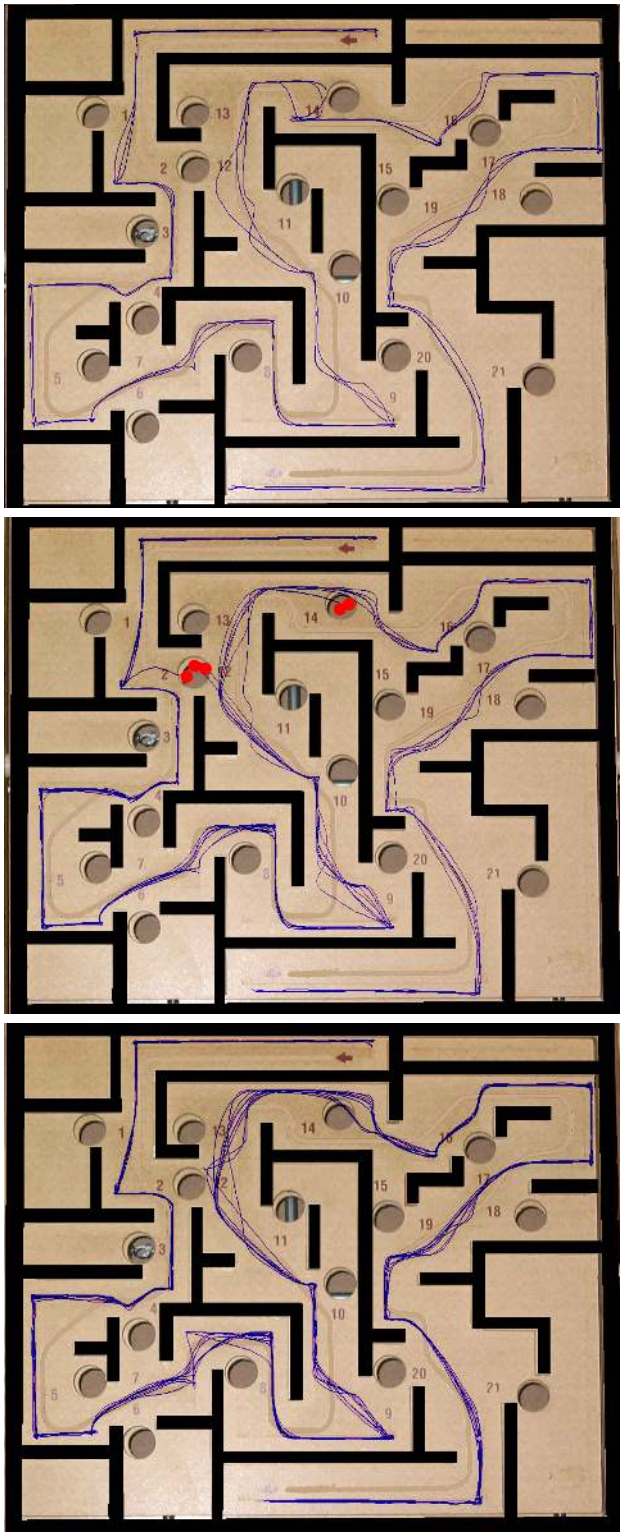

Figure 43: Top: The 3 observed games played by the human teacher. Middle: Performance on 10 games based on learning from observation using the 3 training games. The maze was successfully completed 5 times, and the red circles mark where the ball fell into the holes. Bottom: Performance on consecutive 10 games based on learning from practice after 30 practice games. There were no failures.

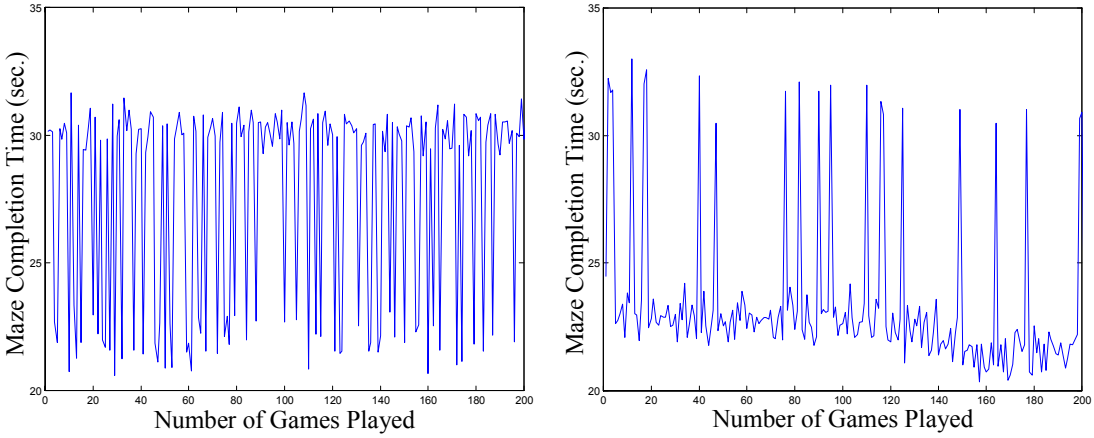

Figure 44: Raw game times of a robot performing 200 trials in the software marble maze environment using only observed information (left) and given the ability to learn through practice (right).

into a hole. The robot was initialized with three observed games played by the human. During the observed games the human did not make any failures and had an average completion time of 25.0 seconds $(25.0, 25.0, 25.1)$. Figure 44 shows the raw game times of a robot performing 200 trials in the software marble maze environment using only observed information (left) and given the ability to learn through practice (right). Figure 45 shows number of failures made by a robot performing 200 trials in the software marble maze environment using only observed information (left) and given the ability to learn through practice (right). A failure is an instance of the marble falling into a hole or not making progress through the maze. These robots use the evaluation strategy described in Chapter 5.3.4 and are placed back on the board after a failure.

To obtain a more comprehensive evaluation of the algorithms, multiple trials have been run with a robot using only observed information and given the additional ability to learn from execution. Figures 46 and 47 show the performance of the robot during five runs using only observed information, solid (blue) line, and five runs with the ability to change its primitive selection and subgoal generation policy, dotted (red) line. This robot follows the evaluation policy described in Chapter 5.3.4 and the marble is replaced on the board after a failure. Figure 46 shows the robot's performance during ten $5,000$ second runs. The left graph shows the number of goals made per second over time. The right graph shows the total number of goals made over time.

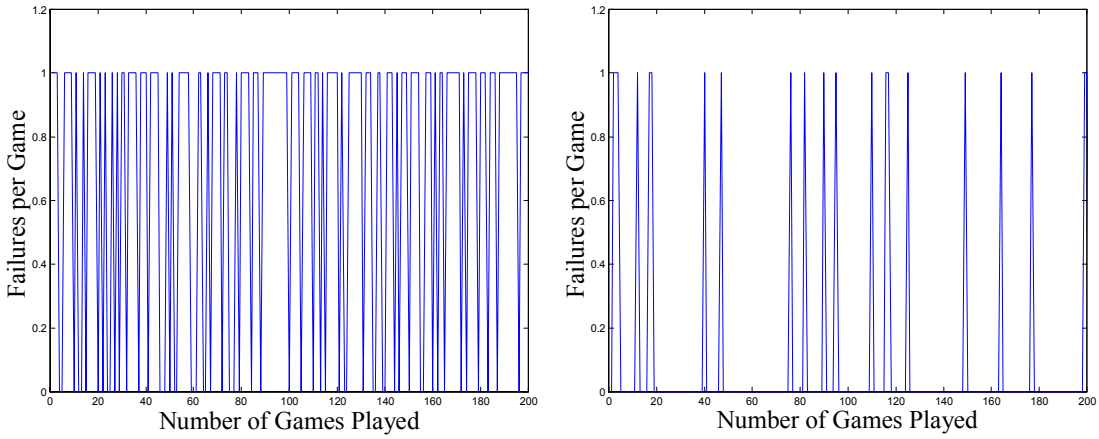

Figure 45: Failures made by a robot performing 200 trials in the software marble maze environment using only observed information (left) and given the ability to learn through practice (right).

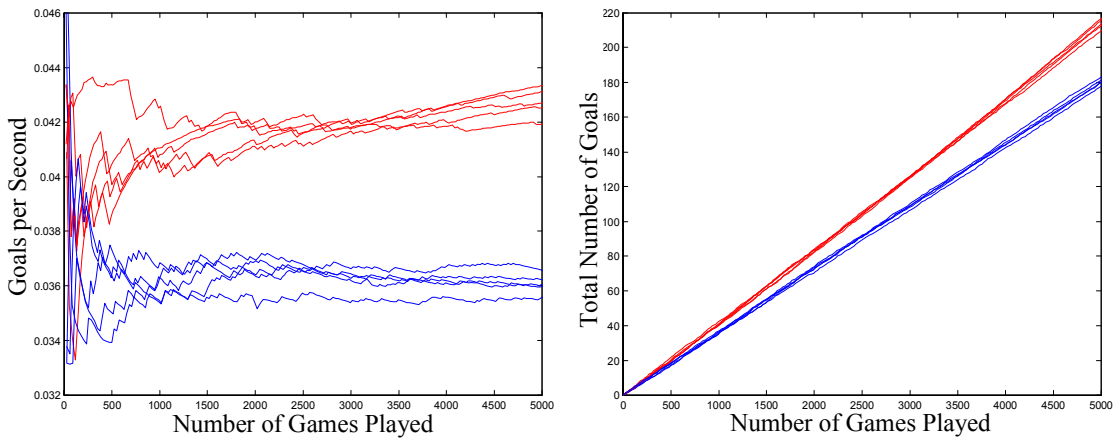

Figure 46: The performance of the robot during five 5,000 second runs using only observed information, solid (blue) line, and five 5,000 second runs with the ability to change its primitive selection and sub-goal generation policy, dotted (red) line. Left: number of goals made per second over time. Right: total number of goals made over time.

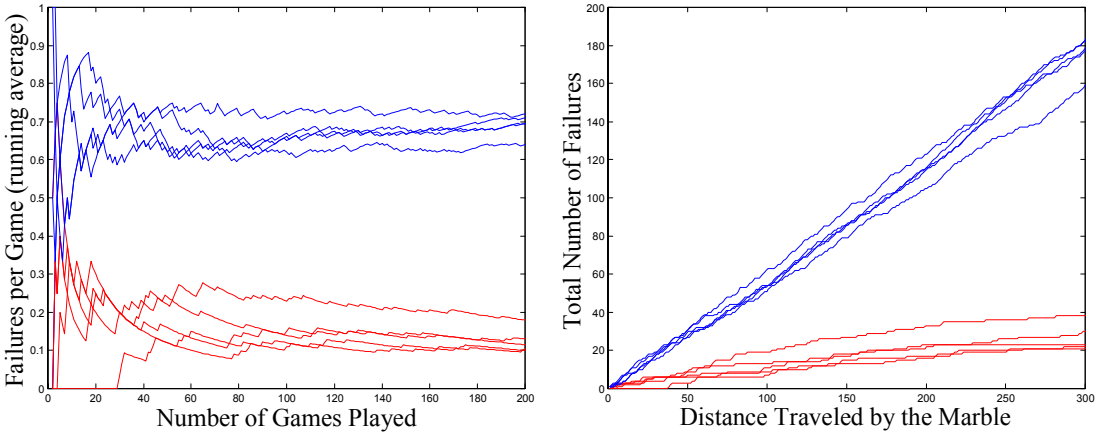

Figure 47: Failures made by the robot during five runs using only observed information, solid (blue) line, and five runs with the ability to change its primitive selection and sub-goal generation policy, dotted (red) line. Left: number of failures made per game over the course of 200 games. Right: total number of failures over 300 meters runs.

Figure 47 shows the failures made by the robot. The left graph shows the number of failures made per game over the course of making runs of 200 games each. For the right graph, the robot makes runs of 300 meters and the graph shows the total number of failures made during each trial. When the marble reaches the goal location it is placed at the start location and the run continues.

The LWPR models used by the above robots are replaced with tables to encode the Q values and the same tests have been performed. Figure 48 shows the performance of the robot using tables compared to a robot that uses LWPR models. The right graph shows the goals per second during 5000 second runs. The left graph shows the number of failures the robot makes during runs of 300 meters each. The parameters for each of the function approximators were chosen manually through trial and error to obtain the best performance for each system encoding scheme and the three observed games.

We also analyzed the performance of the robots in simulation as a function of the amount of observed data. We created a database of 50 observed games. The games were performed by a human and the time to complete the maze varies from 20.1 to 25.6 seconds. Only games that do not include failures are included in the database. Figure 49 shows the result of providing this information to the robot. Thirty runs of 320 games each were performed by the robot after observing one to five games. At the beginning of each run the robot chooses observed games randomly from

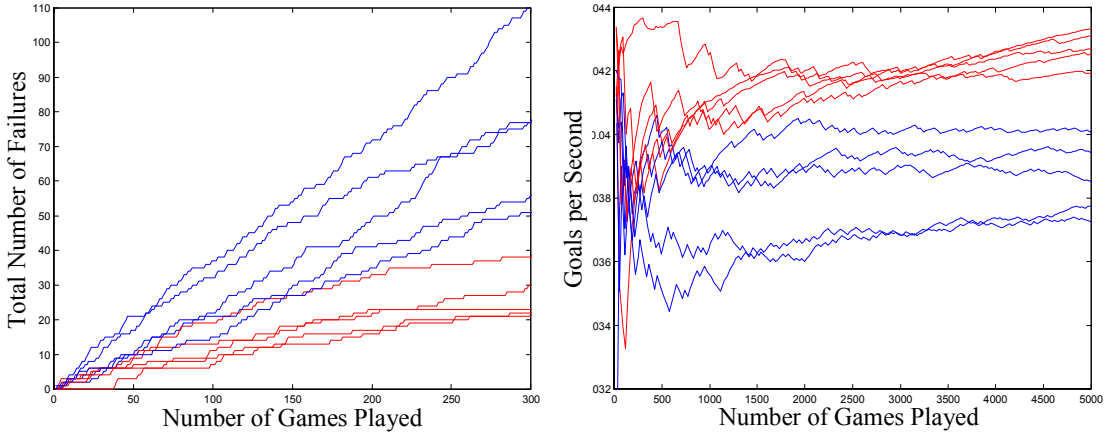

Figure 48: Comparing the performance of a robot encoding the Q value in tables, dashed (red) line, and encoding the Q value in LWPR models, solid (blue) line. Left: failures made during 300 meter runs. Right: goal per second during 5,000 second runs.

the entire database. The robot uses this information under three situations; 1) no learning from practice (Observation Only), 2) learning with Q values encoded in tables (TABLE), and 3) learning with Q values encoded in LWPR models (LWPR). This figure shows the average performance of the robot during games 301 to 320. The average and standard deviation is shown for the 30 runs.

Figure 49 shows that when the robot is not learning from practice, the bars marked "Observation Only," its performance varies more than when it is given the ability to learn from practice. Both encoding schemes, table and LWPR, used the hand-tuned parameters that were designed for the three games provided to the robot with the performance shown in Figures 46 and 47 above. The performance of the learning systems are approximately the same. The encoding system using tables is more consistent as would be expected since the cell sizes remain fixed. The LWPRs, on the other hand, are formed by the observed Q values and will vary depending on the ordering of the observed values.

The observed games that are include in the database were not performed consecutively by the human, but were chosen from among many performance attempts with games that include failures removed. The average of a run of 20 consecutive games, including games that have failures, played by the observe human was 29.6 seconds with a standard deviation of 7.4 seconds. The robots learning from practice exceed that performance after three observed games.

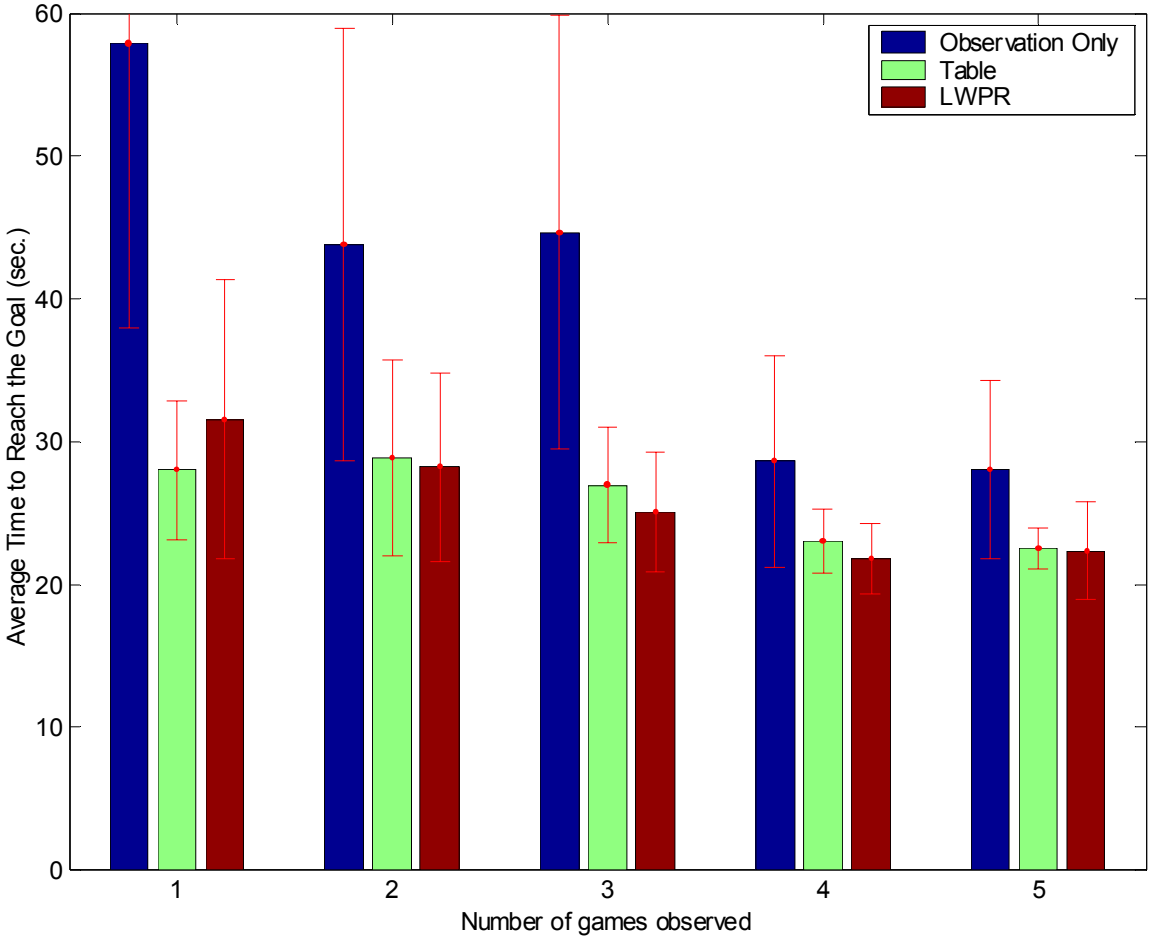

Figure 49: Observing one to five games while not learning from practice (left), learning from practice using tables (middle), and using LWPRs (right).

6.1.5.1 Looking Inside the Operation of the Algorithms

The previous graphs show that giving the robot the ability to learn while practicing improves performance in many ways. This section looks more deeply into the changes that are taking place in the algorithms. Figure 50 shows all the primitives selected by a robot during a game using the initial policy created from observation. This graph is similar to Figure 39 described in Chapter 5.3.5. The graph shows that in some parts of the maze, shaded areas, the robot is having difficulty choosing a primitive and generating subgoals that can be used. As described in Chapter 5.3.5, if an action cannot be performed another action can be selected on the next time cycle. Since the environment changes only slightly during this time, the robot may select a similar action that cannot be performed. During this time, the marble is moving in the environment but the board orientation is not being controlled by the robot. The robot cannot take control of the board until the marble moves to a position where a usable action is selected.

Figure 51 shows the primitives chosen by the robot after it has practiced for 100 trials. The graph shows that this robot makes fewer selections during the game and that, overall, a single primitive is active longer. Therefore the robot has learned which primitives are useful for completing the task and is in control of the board throughout most of the task performance. When not learning from practice it is possible for the robot to get stuck if the environment is not changing and a usable primitive cannot be selected. But when learning through practice, even when the environment is not changing, the robot will change its selection behavior reducing the chance of getting stuck.

After the robot has practiced the task many times the learned Q value function models contain information on the usefulness of choosing data points at various locations. We then take an individual model and query it to observe the data point's Q values when trying to select this data point at various locations in the state space. But the models cover six dimensions and only contain information about areas of the state space that have previously been visited. The following graphs show what has been learned by a typical LWPR model associated with a data point in the marble maze environment. The graphs are created by holding the board orientation and marble velocities constant and querying the model at different board positions. The model being tested is associated with the **Roll Off Wall** primitive action shown in Figure 52. This primitive was observed when the

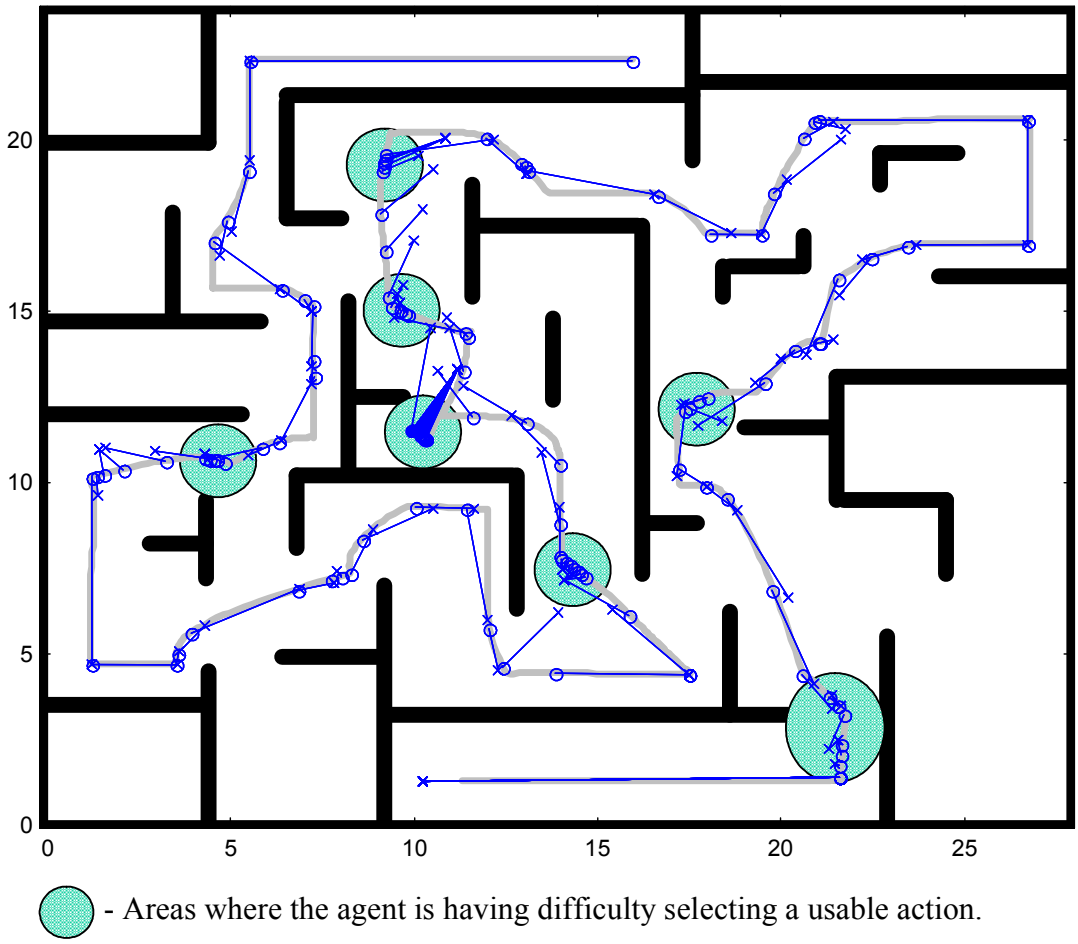

Figure 50: Primitives chosen by a robot while performing the software marble maze task. This robot is not learning while practicing. The gray line shows the path of the marble during this game. The solid line connects the primitive start location, "o", to its associated sub-goal location, "×".

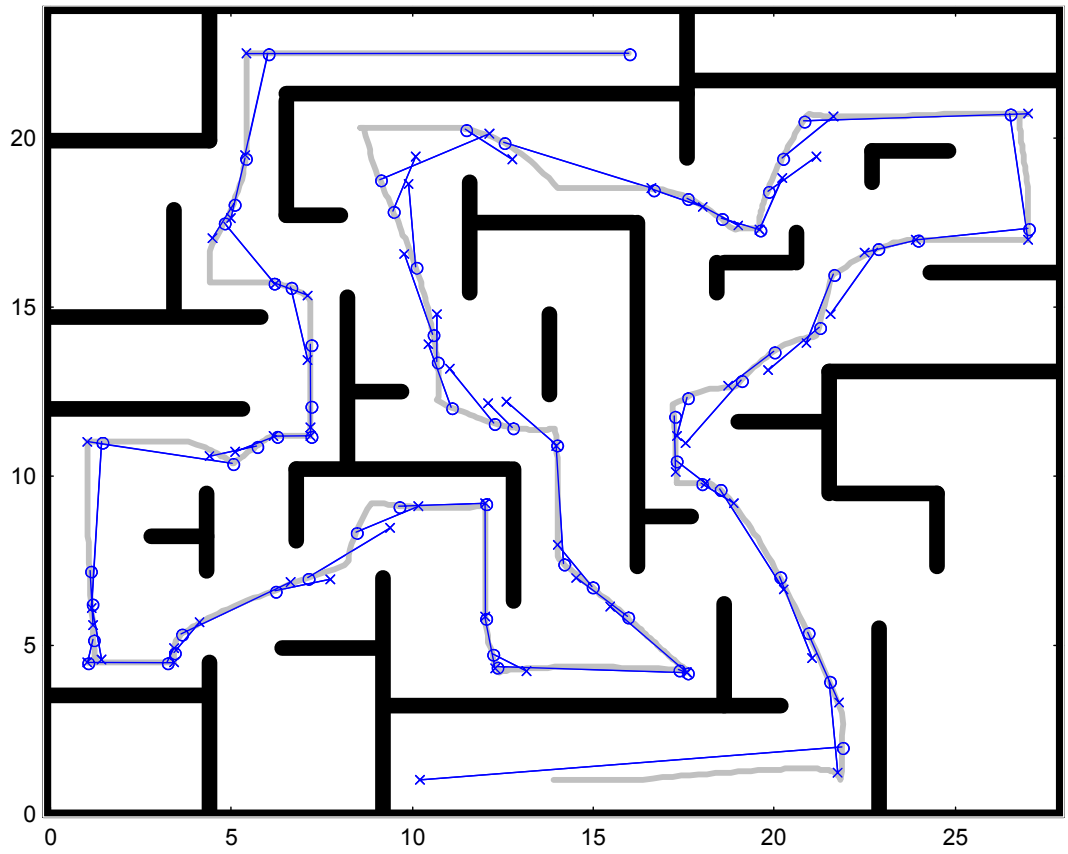

Figure 51: Primitives chosen by a robot while performing the software marble maze task after the robot has practiced by playing 100 games. The gray line shows the path of the marble during this game. The solid line connects the primitive start location, "o", to its associated sub-goal location, "×".

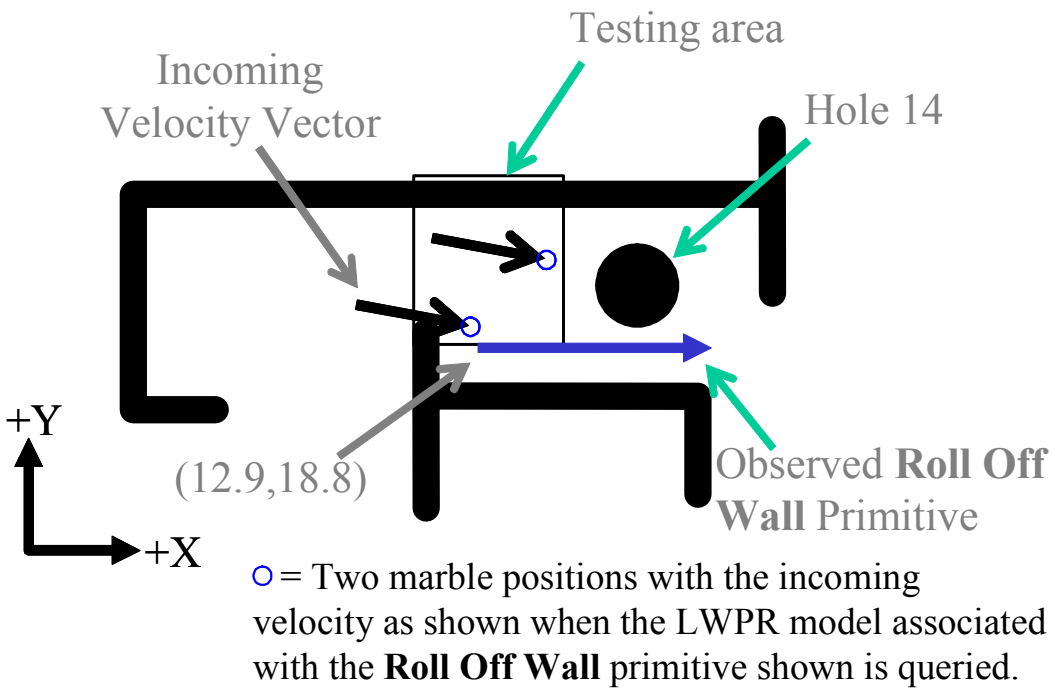

Figure 52: Testing the LWPR model associated with an observed primitive data point.

marble was at the location $(12.9, 18.8)$. The shaded area shows the approximate area being tested. Two possible marble positions along with a vector that represents the incoming velocity are also shown. Intuition says that choosing the shown action for the marble position near the origin of the observed primitive, $(12.9, 18.8)$, should lead to successfully maneuvering the marble to the subgoal location. But if this same action is chosen for the location near hole 14, the marble will fall into the hole with a high probability. If the LWPR has been exposed to this situation it should encode the knowledge that this should be used at one of these locations but not the other for the given marble velocity.

We created model for this data point by having the robot practice 400 games in the simulator. We first tested the model to see what the activation levels were in the testing area. The activation levels give an indication of the query point's distance from the center of the receptive fields within the model. If the maximum activation is not greater than 0.1, the model does not have information for the query point and the initial Q value is assumed. Figure 53 shows the model activations for the testing area. The graph shows that there is no activation in the area that overlaps the top wall. This

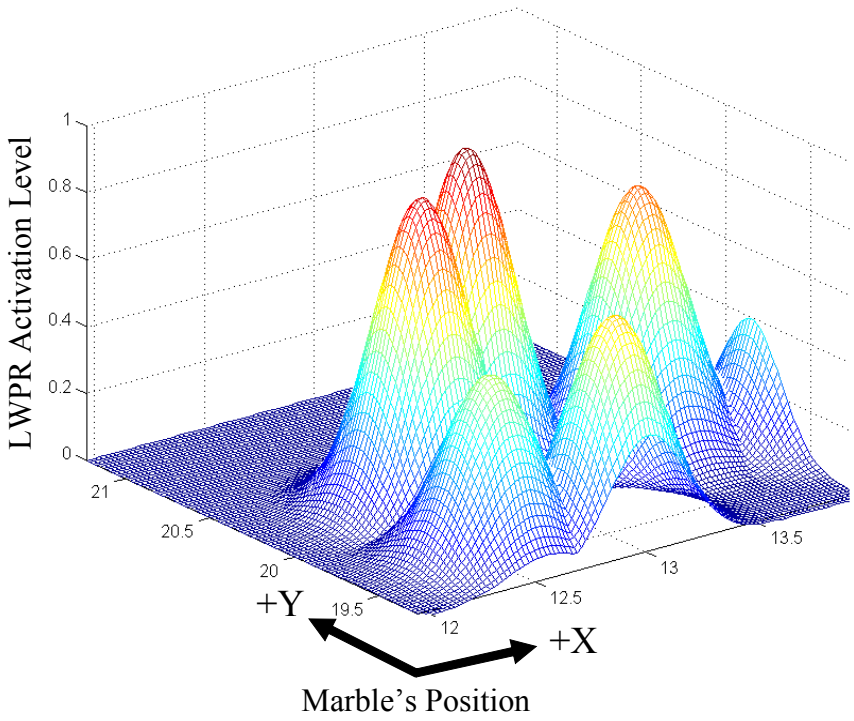

Figure 53: Activations of an LWPR model.

is to be expected because the robot should never have been in that environment state. The activation also drops off as it is queried further to the left of the data point origin. This may be that this primitive has never been tried from those locations. But remember that this is a six-dimensional space and this graph is only showing two dimensions. So the model may know about the area to the left, but not for the given velocities and board angles. Because the activations are greater than 0.1 in a large part of the testing area, it can be assumed the model has knowledge about using this data point in this area for the given velocities and board positions.

Figure 54 shows the Q values returned from querying this model. If the LWPR model does not return an activation of greater than 0.1 the initial Q value is used and this can be seen in the graph in the flat area at the $70,000$ level. The graph in Figure 54 shows that this action has a low Q value when the marble is close to the hole. But if the marble is a little farther from the hole, it is very good to use this data point and high rewards are received that have increased the Q value. The Q values near the data point's origin are also above the initial value. But in the testing area closest to the

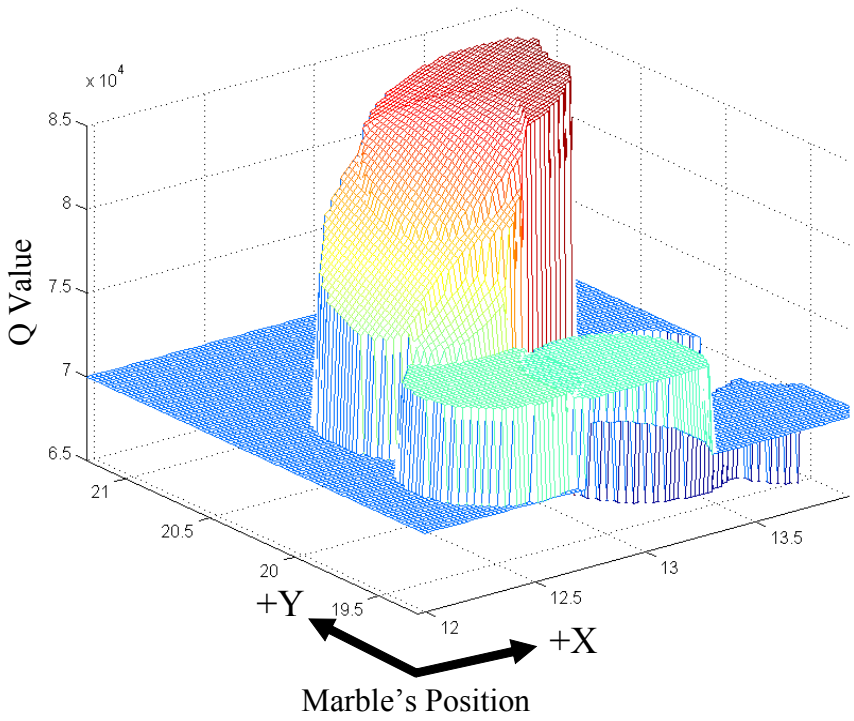

Figure 54: The Q-values returned when quering the LWPR model in the testing area and associated with the data point shown in Figure 52.

data point's subgoal location the model does not have information for the given velocity and board orientation. For the given environment state, this primitive has been selected in the area where the activations are greater than 0.1 and the marble either makes it to the subgoal location or falls into the hole. Therefore this primitive was not initialized in the area near the subgoal location.

The Q values are then used to compute a multiplier that is used in the regression as shown in Section 6.1.2. For this implementation the equation $e^{-(C-V)^2/\alpha}$ is used to convert the Q-value to a multiplier where C is the initial Q value of $70,000$ and α is $10,000$. Figure 55 shows the multipliers that would be computed for the situation shown in Figure 54. Again the flat area at the 1.0 level represents areas of the environment state where the model does not have knowledge and the initial Q value is used.

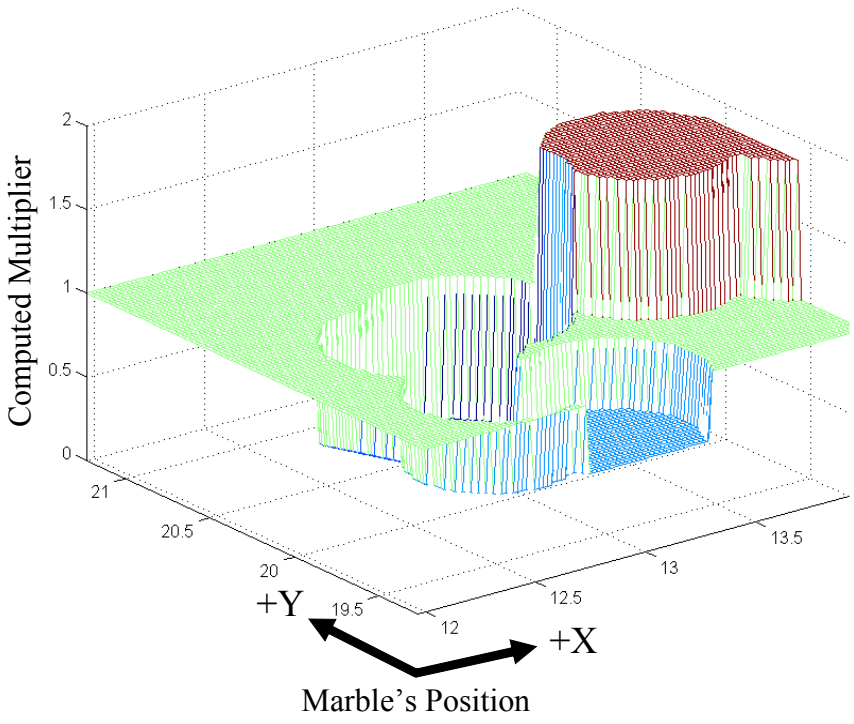

Figure 55: The multipliers computed for the Q-values shown in Figure 54.

6.1.6 Parameter Analysis

Our learning system has various parameters that were hand tuned. This section looks at the effect of those on the performance of the robot. The parameters that will be changed and their values for the creation of the graph in Figure 49 are as follows:

- Weights on the individual scaled dimensions in the regression (w).
  - Marble position (M_x, M_y):100.0
  - Marble velocity ($\dot{M}_x, \dot{M}_y$):1.0
  - Board rotation (B_x, B_y):1.0.
- The value of α in the kernel function $K(d) = e^{-d^2/\alpha}$:10.
- The number of data points used during the regression to compute the subgoal:4.

Each test is performed using five random observed games from the database. 30 trials are run for each situation and the bar graphs show the average of 20 games after the robot practices for 300 games.

The weights on the dimensions in the regression (w) control the significance of the individual dimensions on the result of the regression, after scaling each dimension to ± 1.0. Figures 56, 57, and 58 show the result of changing these weights. The graphs on the left show the performance of the robot after it has practiced for 300 games and the graph on the right shows the performance of the robot while practicing (using tables to encode the Q values) for three different weight values during the first 100 games. From these graphs it can be seen that board position has the most effect on the performance of the robot.

The value of α in the kernel function $K(d) = e^{-d^2/\alpha}$ controls how much points close to the query point have an effect on the regression results. If α is very low, the closest data point will have the most say in what outcome of the regression is. If α is large, all points that are included in the regression will have an almost equal vote. Figure 59 shows the performance of the robot as α is changed.

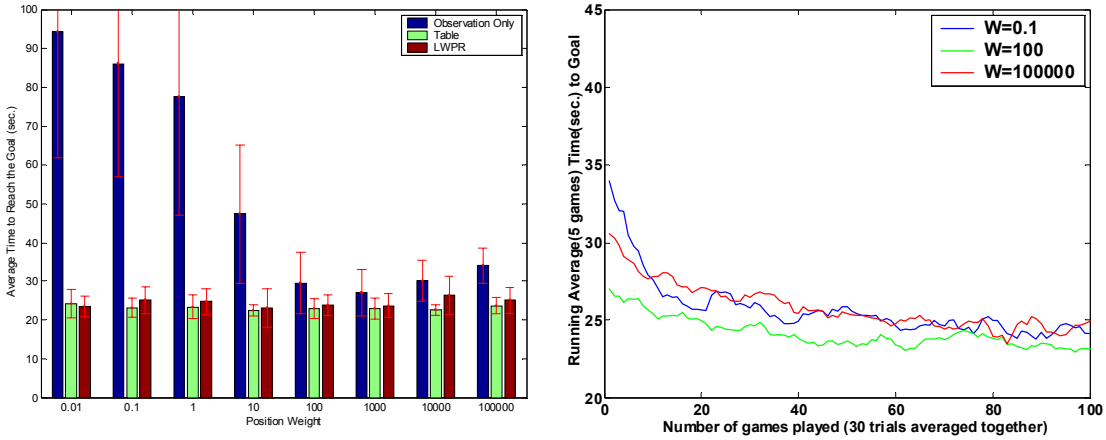

Figure 56: Changing the weight on the marble position dimension.

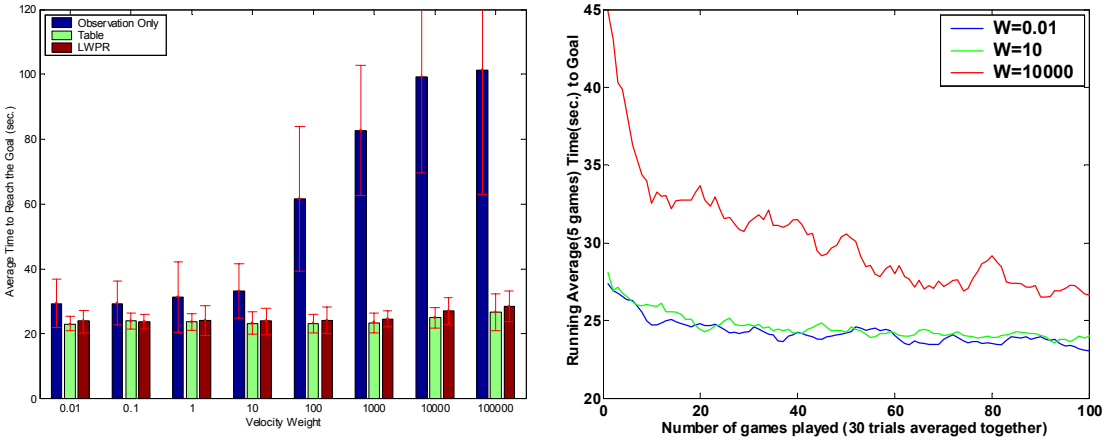

Figure 57: Changing the weight on the marble velocity dimension.

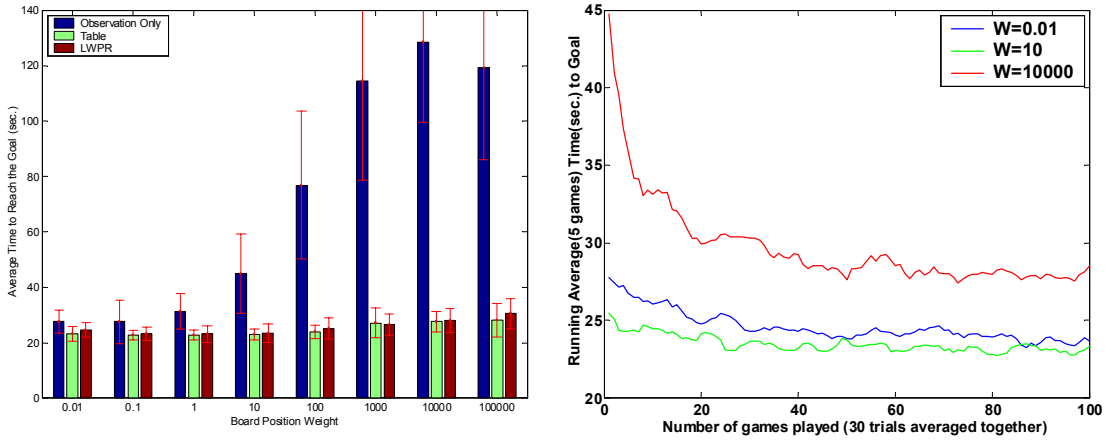

Figure 58: Changing the weight on the board angle dimension.

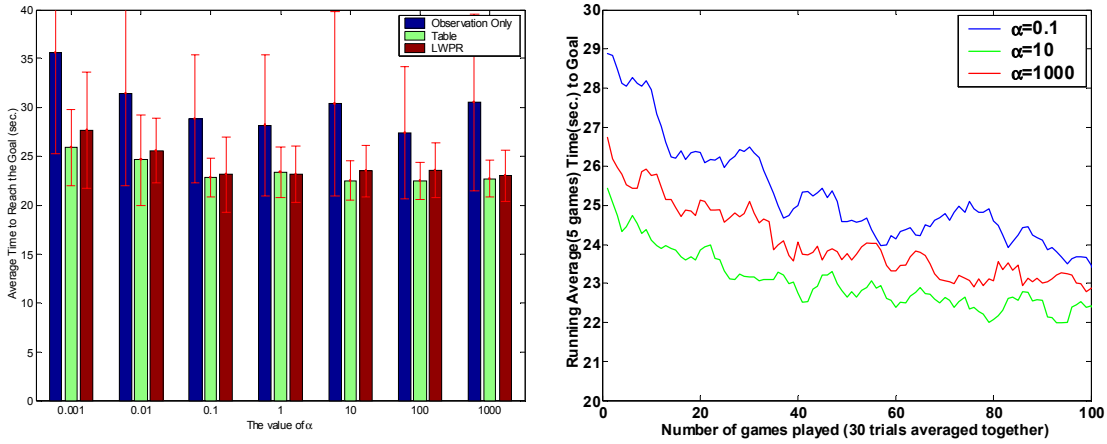

Figure 59: Varying the value of α in the kernel function.

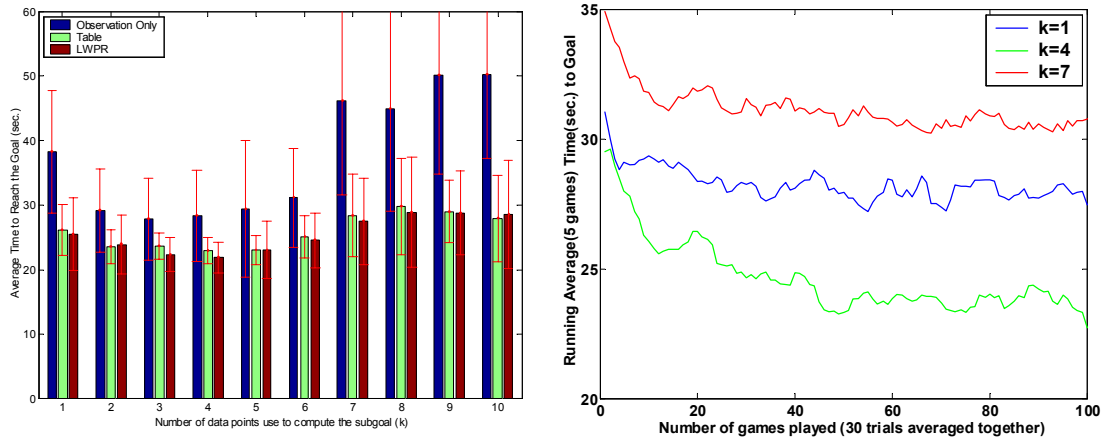

Figure 60: Changing the number of data points that are used in computing the subgoal.

The primitive type to use for a query point is decided upon by the closest data point. The subgoal specified by this single data point can also be used as the desired subgoal. The regression allows multiple data points to contribute towards the desired subgoal. Figure 60 shows the result of changing the number of nearby data points that are used in the regression.

From these graphs it can be seen that varying the parameters has the most effect on the robots that do not learn from practice. All the robots, when given the ability to learn from practice, significantly improve their performance. Our learning from practice system is very robust and in most situations the robots obtain approximately the same performance after practicing for 300 games.

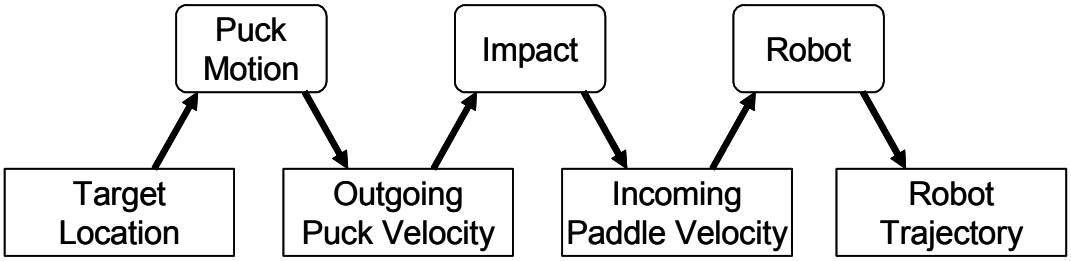

Figure 61: The models involved in action generation: *puck motion*, *impact*, and *robot*.

6.2 Learning to Perform Primitives

There is an extensive amount or research in having robots learn to perform a single primitive through practice as discussed in Chapter 2. This section presents methods that are being used by our air hockey learning robots to increase their performance of the shot primitives through practice. Chapter 5.2 describes how the shots are made in air hockey using the models shown in Figure 61.

The *puck motion* and *impact* models were initialized with data from observing a human as described in Chapter 5.2.4. The solid line in Figure 62 shows the result of the robot making 500 straight shots in the simulator. For the first 200 shots the robot is using models created from observing the human's shots and the graph plots the average absolute error in hitting the target location, the distance between the target location and the location where the puck actually hit the back wall. The values plotted are the running average of 5 shots. The dotted line at the bottom of the graph shows the results of the robot performing the action using an exact model of the simulator. The error in the exact model is due to the noise introduced into the simulator and this is effectively the best the robot can perform.

Based on this comparison, it appears there is room for improvement. One way to increase performance is to have it observe the human making more shots. But this can be time consuming as it took over ten minutes to see only 152 shots. A more practical alternative is to have the robot observe its own behavior and add that information to the models.

After making 200 straight shots using the models learned from observing the human, the robot then observed 100 of its own shots while practicing (shots 201 to 300 in Figure 62). Whenever the robot observes its own shot it calculates the parameters in the same way as if it were observing a human. This information is then immediately given to the models. Figure 62 shows the result of

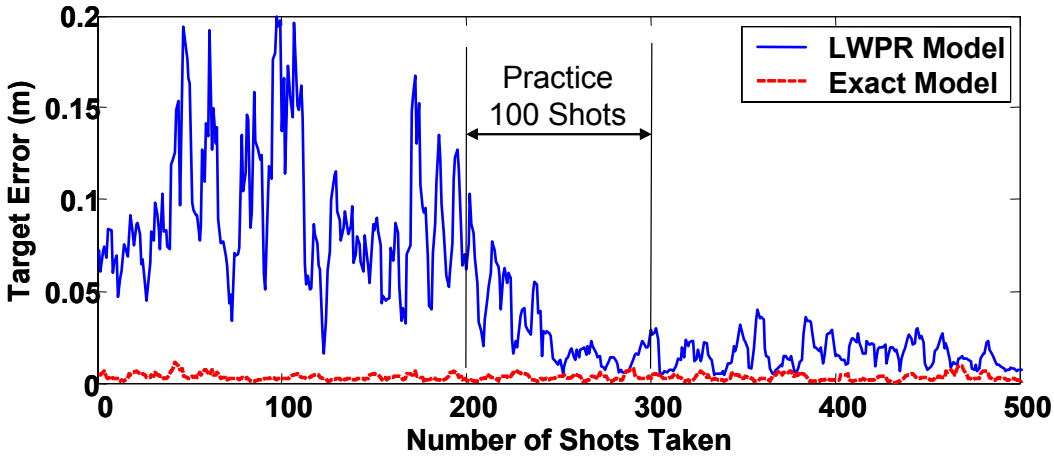

Figure 62: This graph shows the magnitude of the error in reaching the target location during 500 straight shots made by the robot in simulated air hockey. The solid line shows the result of the robot making 200 shots using the LWPR model trained from observing 44 straight shots performed by the human. It then observes 100 of its own shots while practicing and adds that information to the LWPR model. The dotted line is the result of a robot making straight shots using an exact model of the task. The graph shows the running average of 5 shots.

using these newly trained models for the shots from 301 to 500.

6.2.1 Action Learning in Physical Air Hockey

Learning on hardware provides a set of challenges that are not present in the simulator. The simulator can start in, or be set to, any given configuration. The movement of the items in the simulator can also be accurately controlled and sensed. This section presents a method that is used by the humanoid robot in the hardware air hockey environment to adapt to paddle movement and table placement errors. This method also allows the robot to learn the timing of the paddle movements.

The humanoid robot positions the paddle on the table using an interpolation method described in Chapter 3. This system is used to generate the robot trajectory in the robot model shown in Figure 63. The input to this system is the desired position of the paddle and the time in which it should make the movement. The movement is started after a fixed delay that represents the time delays in initializing the command and the time it will take for the paddle to reach the hit location.

This is a simple and useful method for positioning the paddle, but if the table is not accurately placed, or moved during the task, there will be an error in positioning the paddle on the table. We have found that paddle placement accuracy is also affected by the desired movement velocity. The

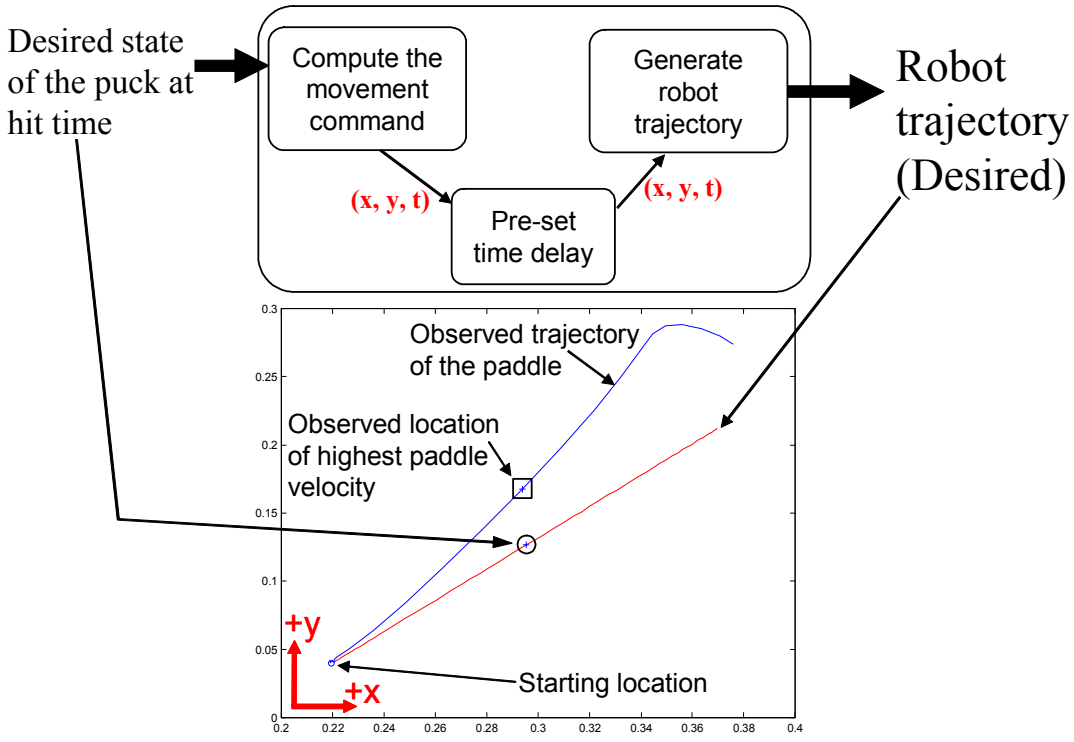

Figure 63: The *robot* model used to control the humanoid robot and errors in placing the paddle due to exceeding the design limits of the robot.

accuracy is much higher during slow shot maneuvers than during fast shot maneuvers. A reason for this is due to the design of the robot and the fact that some of the degrees-of-freedom (DOFs) are stronger and/or faster than others. One way to reduce the effect of this problem is to ensure that we keep the desired movement velocity lower then the slowest DOF. But this would severely limit the robot's performance. The robot is capable of moving the paddle at velocities close to those observed by a human player. The problem is that the paddle does not always correctly follow the specified trajectory at these high velocities. Figure 63 shows the deviation of the paddle from the desired trajectory during a hit maneuver.

For the task of making shots in the air hockey environment, there is only one important instant, and that is when the puck and paddle collide. It is at this instant that the paddle affects the movement of the puck. Therefore it is not the entire trajectory that is important, but the state of the paddle at the instant it hits the puck. If the robot can repeatedly control the paddle to be in the correct state at the correct time, it can make accurate shots in the air hockey environment irrespective of the path the paddle takes to arrive at that hit point. Figure 63 also shows the difference between the desired hit location and the location where the highest paddle velocity is observed during the hit maneuver.

6.2.1.1 Learning the Robot Model

The robot model must generate a trajectory which has the paddle arrive at the hit location at the correct time with the correct velocity. The robot command consists of a desired location and a time in which to reach that location. The movement follows a fifth order polynomial with zero start and end velocities and accelerations. It therefore should have its highest velocity in the middle of the movement and this is where the puck-paddle collision should occur. The graph in Figure 64 shows the path, the lines (blue) with the boxes on them, of three hit maneuvers. The lines (red) with the circles on them are the desired trajectories. The robot's initial position for the three maneuvers is approximately $(0.235, 0.05)$. The robot is commanded to move to the position $(0.31, 0.16)$ in $14.3ms$. The graph shows where the paddle-puck collision is expected to occur and where in the actual trajectory of the paddle the highest velocity is observed. From this graph it can be seen that there is a repeatable error for this given command and there may exist a set of commands that would place the puck at the desired hit location with the desired velocity vector.

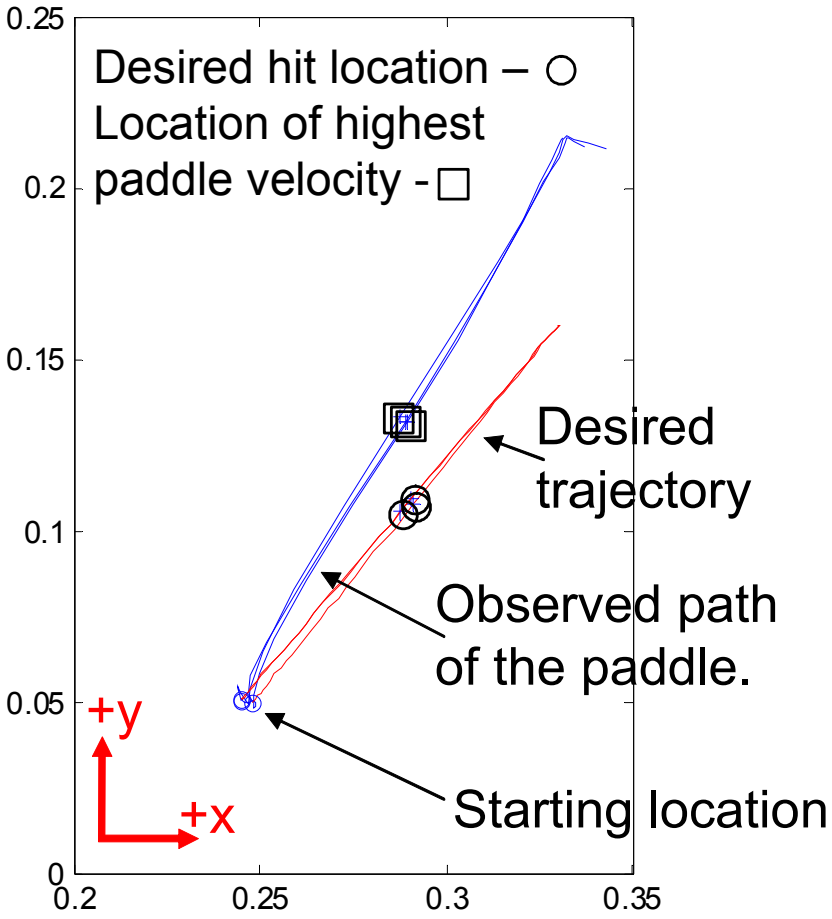

Figure 64: The errors in making three similar hit maneuvers.

It is the function of the *robot* model to learn the robot commands that will correctly place the paddle at the time of the hit. The robot learns this model when the board is placed in the environment and play is about to begin. At this time the robot makes a set sequence of 100 paddle movements that would be typical for making hockey shots. When a movement is commanded, the starting location of the paddle as seen by the robot, the desired movement command, and the time the command is sent to the robot are recorded. The movement of the paddle is then observed and the paddle velocity is computed whenever new vision data arrives (60Hz). The paddle's position and velocities, and the time of the observation, are recorded. When the paddle velocity returns to zero, a data point is created with the following information:

- Input:
  - Starting position of the paddle.
  - The state of the paddle, position and velocity, at the highest velocity location.
- Output:
  - The position and time command given to the robot.
  - The time from when the command is given to the time the paddle is observed at the highest velocity location.

These data points are used to train an LWPR model that is used in the *robot* model.

6.2.1.2 Using the Robot Model

The paddle's position and velocity needed at the time it hits the puck are computed using the puck's hit position and the information returned by the *impact* model. This information, along with the paddle's current location are used as inputs to the robot model. The model then provides the command that will place the paddle as desired.

The three graphs in Figure 65 show the results of using this trained LWPR model to make the same maneuver as the one shown in the graph on the left. Each graph shows one hit maneuver and the robot starts in approximately the same initial state. The initial paddle position and desired paddle hit state are input to the LWPR model and the values returned from the model are used as

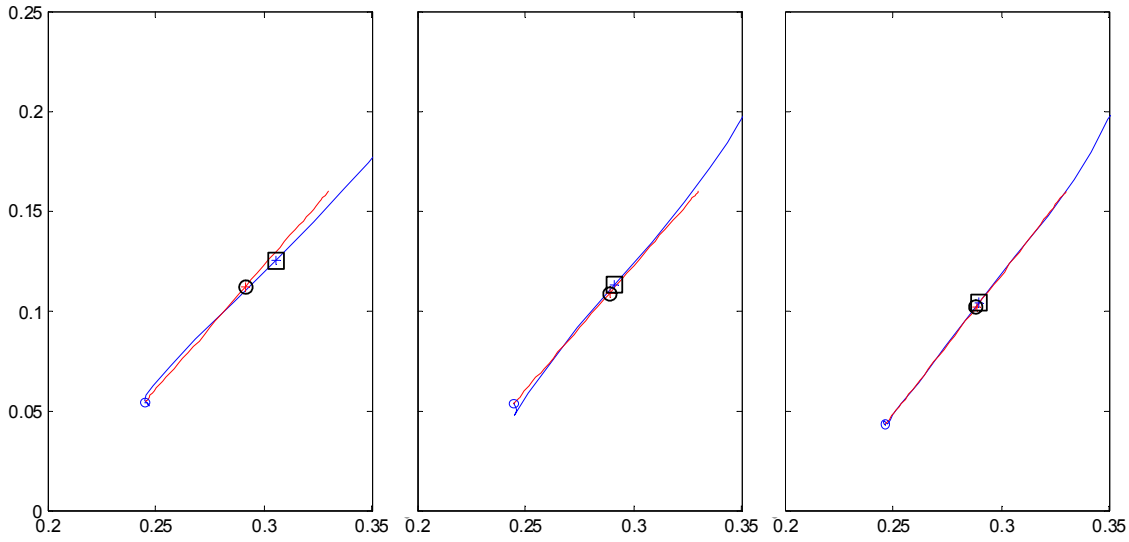

Figure 65: The lines (blue) with the boxes on them in these graphs show the path of the paddle during shot maneuvers. The lines (red) with the circles on them are the desired trajectories. The three graphs show the same shot maneuver being made using a trained LWPR model.

the movement command. The graphs show that the model has learned to more accurately place the paddle at the desired hit location with the velocity vector pointing in the correct direction and the location at which the highest paddle velocity is seen is now much closer to desired hit position.

6.2.1.3 Using the Timing Information

The *robot* model also learns the time that it will take for the vision system to observe the paddle at the desired hit location. In the hardware environment there are delays in observing the items in the environment and in commanding the robot. Because of the high velocities of the puck, a delay in sensing and computing the puck position means that the puck's position as reported by the vision system is not the real-time position of the puck. If the real-time position is required, a model of the environment must be used to predict the position of the puck into the future by an amount of time equal to the sensing delay.

The timing information provided by the *robot* model removes the need to know the vision and command delays. This timing information tells the time, from when the command is given, that the paddle will be observed at the desired hit location. This means that the robot can now work entirely in the vision system's frame of reference when computing the time that the movement command

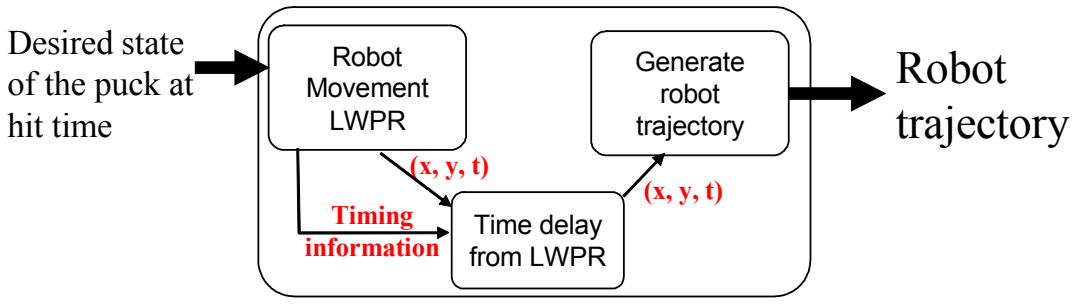

Figure 66: The updated *robot* model with the ability to learn from observing its own behavior.

should be given. When the puck's position and velocity are observed shortly after being hit by the opponent, the action generation module, using the hit line supplied by the subgoal generation module, predicts the time it will take, in the time frame of the vision system, for the puck to reach the hit line. In other words, this is the time the puck should be observed crossing the hit line. This predicted time, along with the timing information returned from the *robot* model, can now be used for determining when the hit should be commanded. For example, if the predicted time for the puck to cross the hit line is computed as $300ms$ and the time for the paddle to be observed at the hit position, returned from the *robot* model, is $250ms$, the robot's movement should be initiated in $50ms$. $250ms$ after the movement command is given, the vision system should observe the puck and paddle in the proper hit positions. Figure 66 shows the new *robot* model that has the ability to improve its performance from observing its own movements.

CHAPTER VII

DISCUSSION

Then Jesus came to them and said, "All authority in heaven and on earth has been given to me. Therefore go and make disciples of all nations, baptizing them in the name of the Father and of the Son and of the Holy Spirit, and teaching them to obey everything I have commanded you. And surely I am with you always, to the very end of the age."
- Matthew 28:18-20 (NIV) [96]

This Chapter explains the decisions that were made about the structure of the framework and discusses some of the issues related to this research. It also presents a comparison of the learning system presented in this thesis with a reinforcement learning system. The comparison highlights characteristics of the framework which are also discussed in this chapter.

7.1 Why Does the Framework Look the Way it Does?

While a framework offers structure and can help to modularize the problem, it also limits what can be done to only things provided for by the framework. Many decisions must go into creating a framework that affords an appropriate trade-off between modularization and flexibility. This section discusses some of the design decisions that went into creating the framework presented in this thesis. The ability to use observed data in a systematic way and to learn while practicing were two of the main concerns while creating the framework. The ability to generalize within the environment and across to other environments was also considered.

7.1.1 Combining Primitive Type and Parameter Selection

The presented framework first selects the primitive type. The selected primitive type then narrows down the search for subgoals. What if we do away with the primitive selection module and just have the subgoal generation module provide the next subgoal? To use that subgoal information there must be a higher level process that can select the primitive type that needs to be performed to use that subgoal in the robot's current state. We are now back to the original problem of selecting a

primitive type to use, but have different information in which to select it. For this situation it would be useful if a primitive could tell us if it is capable of taking the system from the current state to the goal state. The research of Faloutsos et al. [40] provides an example of a method to find the set of preconditions under which a policy will operate correctly and also shows how difficult this is. It is our belief that by first committing to a primitive type we are simplifying the learning problem by only having to learn the set of parameters appropriate to that primitive type. By separating these modules the method used to provide the needed information can be unique for each decision providing extra flexibility.

7.1.2 Combining Subgoal Generation and Primitive Execution

It may be considered that once a primitive type was selected the primitive execution policy can decide on the needed parameters thereby eliminating the subgoal generation module. By doing this there will be a loss in generality and flexibility. The primitive execution module contains the policy to bring the system from the current state to a new state within the constraints of the primitive type. The primitive execution policy maps the sensor readings to an action for each time step and is designed to operate under constraints of a local environment. This means that this same policy may be used in different parts of the state space. A policy for a primitive type, for example, can be performed in any location where the configuration matches that needed for that primitive type. The selected subgoal provides the arguments needed to communicate to the policy the desired outcome. This method also allows the subgoals to be generated using any type of algorithm and information needed. By separating these modules the subgoals can be generated using a coarse discretization of the environment state space and the policy can then use any method appropriate to control the system as seen in the research of Morimoto and Doya [88].

7.2 Primitive Design Considerations

Within this research a domain expert designed the primitives that would be explored in each task. It may be asked, "Are these the best sets of primitives for conducting these tasks?" One can certainly think of other possible primitives for these tasks. This research does not seek to find the optimal set of primitives or prove that we have created an optimal set. This research seeks methods that

robots can use to learn tasks quickly from observation and practice using a given, fixed, set of primitives. It also highlights some issues and challenges about using a fixed set of primitives in dynamic environments.

The set of primitives should provide the robot with an action for any possible state it may be in. This becomes more challenging in the hardware implementations. In the software implementations the simulator can perturb the environment if there is no movement. But in the hardware implementations, if the robot gets stuck, the human will need to intervene to allow the robot to continue the task. Within our marble maze environment we have found that the robot can take actions for all observed conditions. But this is not the case within the air hockey environment. In air hockey, the puck can be within reach of the robot, but not in a position where it can be properly hit. This is seen when the puck is very close to the back wall or when it is close to the center line and at the end of the robot's reach. In the simulator the puck is almost always moving and in most cases soon moves to a position where an action can be taken. The simulator will also reset the environment if no movement is occurring. On the hardware table the puck often stops in these locations and the provided set of primitives cannot be initialized. In our implementation we have given the robot set behaviors such as moving the paddle to the side to move the puck from near the robot and to push the puck to the opponent's side when the puck is near the center line. These behaviors can also be considered primitives but within this research they are not learned from observation or practice and their performance is not explored.

7.3 Perceptual Learning

Within dynamic environments, such as air hockey, the robot must initiate movements before the objects are at the intended interaction location. This is due to the fact that movements cannot be made at arbitrarily fast speeds and there are delays in sensing the objects in the environment. It is also important to note that the object will only be within a range of interaction for a brief period of time. If the robot's movements are not initiated quickly enough, the chance to interact will be lost. Therefore it is important for the robot to have the ability to predict the interaction location as soon and as accurately as possible, based on its perception of the object's current motion.

In our relatively small version of air hockey, the robot observes the puck when it crosses a line

that is just beyond the center-line from the robot and about $0.35m$ from where the puck will be hit. At this point the puck should be out of reach of the opponent and can no longer be influenced by the opponent's movements. The delay in the vision system can be up to approximately $35ms$ and a puck traveling toward the robot at $2.5m/s$ can be up to $0.0875m$ closer to the robot than where it is last seen. Since the robot needs to accurately predict the future location of the puck, it also filters the puck locations to more accurately compute the puck's velocity. This filtering process adds more delay to the puck's estimated state.

We currently use a parametric model with learned parameters to predict the future state of the puck. The accuracy of the future predicted puck state is determined by the accuracy at which the vision delays and environment parameters are known. Chapter 6.2.1 shows how we are initiating movements in air hockey irrespective of the vision delays. But if our model parameters are not correct, the puck will not be at the predicted hit position when the paddle arrives there. The model parameters that are currently being used are global and remain constant. It is likely that the friction is not constant across the playing surface and changes over time due to factors such as the fan motor wearing out. For these reasons the robot should have the ability to learn and update a model of the puck's movements from observing the environment. The next paragraph describes an implementation that we are currently exploring.

As mentioned in Chapter 6.2.1, the most important instant is when the paddle and puck collide. This collision occurs within a small range near the robot. Our continuing research is exploring a method in which the robot observes the position of the puck just beyond the center line for three to five vision cycles. The state of the puck when it crosses the desired hit line is then observed. It is our hope that this information can be used to train an LWPR model that will provide the robot with the ability to accurately predict the puck's state at the hit time. The research of Park et al. [97] shows that a neural net can be trained to provide this prediction on a larger air hockey table given the puck's position and velocity as input. But to accurately compute the puck's velocity, more than two observations will be needed. The increased size of the table used by Park provides them with more opportunities to observe the puck in locations that are out of reach of the players. Their model is also trained with 3,000 observations. We would like the humanoid robot player to learn from much less data and have the ability to update the model while it is operating in the environment.

7.4 Framework Limitations

This section discusses some of the limits of our learning system and task characteristics that support our framework and those that work against it. First a comparison of our learning system with a straight forward reinforcement learning implementation will be presented. This comparison will serve two purposes. First is to show how our learning system compares with a different numerical learning system that is well studied. The second purpose is to highlight some of the framework discussion items that will follow the comparison.

7.4.1 Reinforcement Learning Comparison

Another method that can be used by robots to learn the marble maze task is reinforcement learning (RL) [122]. We programmed a robot to play the marble maze game using a tabular Q learning approach. The most common method of encoding the Q values in a continuous state space is to discretized the state space into cells. The size of the cells are as follows; (M_x, M_y) distance: 1.0cm, $(\dot{M}_x, \dot{M}_y)$ velocity: 10cm/sec, (B_x, B_y) board rotation: 0.01rads. The action is to change the rotation angle of the board and is limited to three possible changes in each direction; $(-0.01rads, 0, 0.01rads)$. With cells of this size and a board of size $28cm \times 23.5cm$, a maximum velocity of $50cm/sec$ and maximum board rotation of 0.14 radians there are $(28 * 24) * (10 * 10) * (28 * 28) = 52,684,800$ states. This results in $52,684,800 * (3 * 3) = 474,163,200$ state-action pairs or Q values. Since many parts of the board are inaccessible due to walls and holes the actual number of possible Q values is smaller than this number. The robot, referred to as the DIRECT robot, uses the Q learning algorithm as described in Section 6.5 of [122]. The Q values are updated as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \left[r_{t+1} + \gamma \max_a Q(s_{\mathbf{t+1}}, a) - Q(s_t, a_t) \right]$$

α, the step-size parameter, is set to 0.2 and γ, the discount rate, is set to 0.999. r_{t+1} is the reward received by the robot when it moves from s_t to s_{t+1}. The reward function is $distTrav * 10 - 1$, where $distTrav$ is the distance in centimeters the marble traveled toward the goal. If the marble moves away from the goal, $distTrav$ will be negative. The robot receives a reward of -1000 if the marble falls into a hole and learning ends when the goal state is reached. The initial Q values,

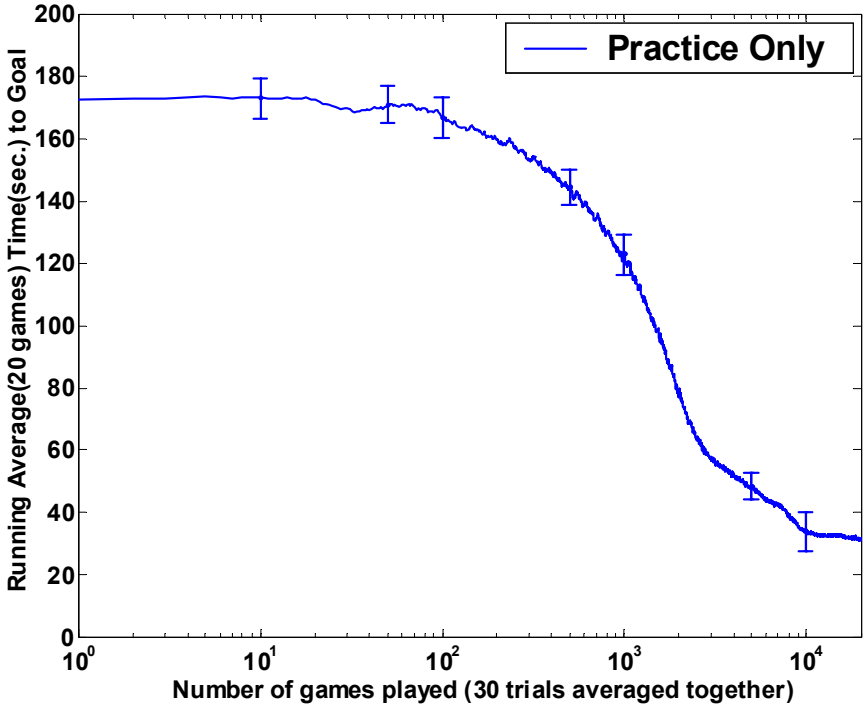

Figure 67: The performance of a robot using direct Q-learning.

$Qinit$, are set to -500.

The action-update cycle occurs at 60 times per second. The robot uses the soft-max function $\frac{e^{(Q_t(a)-Qinit)/\tau}}{\sum_{b=1}^{n} e^{(Q_t(b)-Qinit)/\tau}}$, where τ is set to 10.0, to select actions. The robot operates under the same conditions described previously. If the marble falls into a hole or does not make progress for 15 seconds it is moved forward in the maze and given a 10 second penalty. Figure 67 shows the performance of the DIRECT robot during 30 trials of $20,000$ games each. The graph shows the running average of the time to reach the goal for 20 games. The 30 trials are averaged together and the standard deviation is also shown. In our implementation the Q values are created as they are updated. Figure 68 shows the number of Q values created as the robot operates in the environment and it can be seen that the robot visits significantly less than the possible $474,163,200$ cells. The initial Q values are set to a pessimistic number. This means that as the robot operates in the environment and receives rewards the Q values tend to increase above the initial value. This results in the robot having less of a chance of visiting cells that have the initial value.

This robot is then given the ability to refine the Q values with observed games from the same

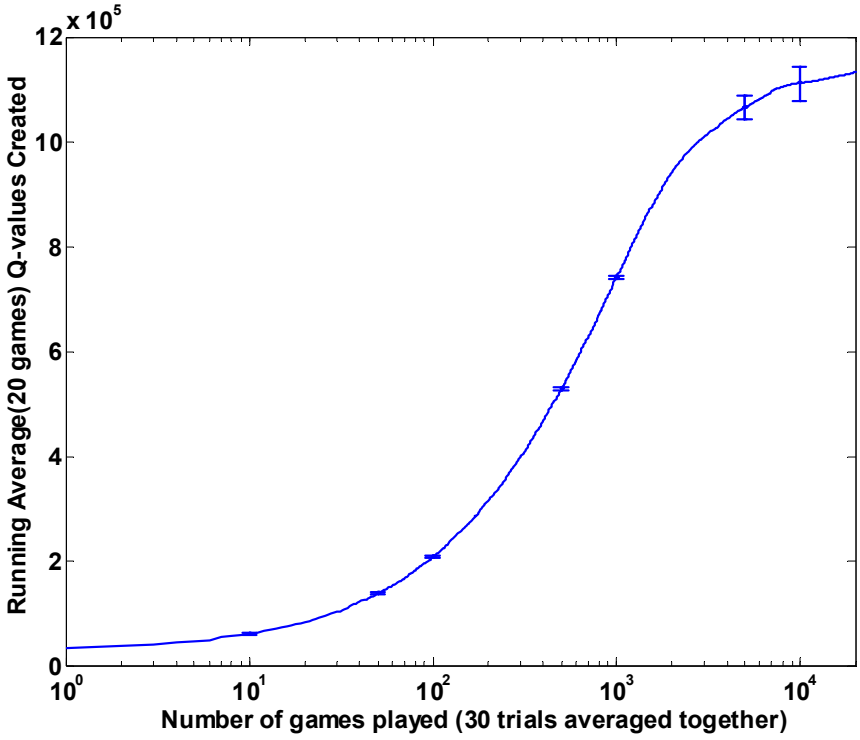

Figure 68: The number of cells created by the direct Q learning robot while performing the marble maze task.

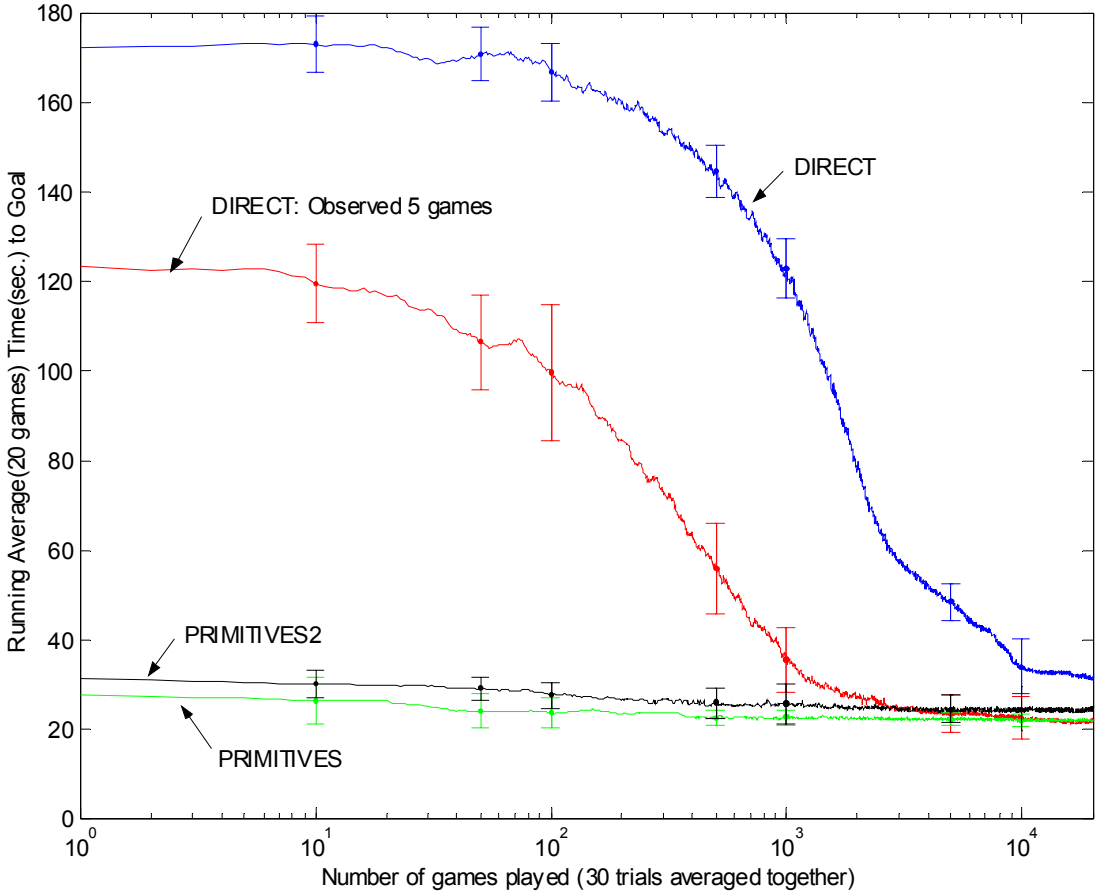

Figure 69: The performance of a robot using direct Q-learning (DIRECT) with and without observation and robots using primitives (PRIMITIVES and PRIMITIVES2).

game database described in the previous chapter. Figure 69 shows the performance after observing 5 random games from the database. The 5 randomly chosen games are fed to the system consecutively, 1 through 5, 100 times and the Q values are updated just as if the robot was playing the game. This figure shows how this robot learns a better policy sooner than the robot that does not use the observed data.

Figure 69 also shows the result of a robot learning using primitives, referred to as the PRIMITIVES robot, as described in this thesis and also using 5 random training games from the same database. The PRIMITIVES robot immediately performs better than the DIRECT robots. But given enough time to practice, the DIRECT robots can match the performance of this robot.

We then modified our PRIMITIVES robot to make it as close to the Q learning robot as possible. The cells that encode the Q values are the same size as the ones used by the DIRECT robot and the

reward is also the same; $10x$(distance (cm) traveled toward the goal) $- 60x$(time in seconds) and -1000 for falling into a hole. The new PRIMITIVES robot, referred to as PRIMITIVES2, no longer stops and restarts learning when a **Corner** primitive is performed but continues and the action receives a reward accordingly. The PRIMITIVES2 robot also uses a soft-max function to choose the primitive to perform.

The Q value is updated using the maximum value that could be expected from the current state using the following function:

$$Q(s_t, \mathbf{x}_m) \leftarrow Q(s_t, \mathbf{x}_m) + \alpha \cdot \frac{K(d(\mathbf{x}_i, s_t))}{\sum_{j=1}^{N} K(d(\mathbf{x}_j, s_t))} \cdot \left[r_{t+1} + \gamma \max_{\hat{\mathbf{x}}} Q(s_{\mathbf{t+1}}, \hat{\mathbf{x}}) - Q(s_t, \mathbf{x}_m) \right]$$

Just like the DIRECT robot's update, α, the step-size parameter, is set to 0.2, γ, the discount rate, is set to 0.999 and r_{t+1} is the reward received by the robot when it moves from s_t to s_{t+1}. In the previous implementation of the PRIMITIVES robot α was set to $\frac{K(d(\mathbf{x}_i, s_t))}{\sum_{j=1}^{N} K(d(\mathbf{x}_j, s_t))}$. This value is still used for the update with an additional step-size parameter that is set to 0.2. $Q(s_{t+1}, \hat{\mathbf{x}})$ is the future reward that can be expected from the new state s_{t+1} and selecting the data points $\hat{\mathbf{x}}$ at the next time step. The maximum value over the available data points is used in the update. Previously the robot would take the maximum value action, but now, since the robot is using a soft-max selection scheme, this is not guaranteed. The initial cell value, $C = -500$, and $f(\mathbf{x}_i, \mathbf{q}) = e^{-Q/\beta}$ where $\beta = 40$. Figure 69 also shows the performance of the PRIMITIVES2 robot and Figure 70 shows how this robot compares with the original PRIMITIVES robot.

When the PRIMITIVES robot finds actions that make progress toward the goal it receives a positive reward and continues to use the same action under the same conditions and does not explore other actions. But the PRIMITIVES2 robot has two opportunities to explore. One is explicitly due to the soft-max selection scheme and the other is due to the reward structure. For the PRIMITIVES2 robot to receive a positive reward it will have to make progress during the primitives at a rate greater than $6cm/second$. But this is often not the case and the robot receives negative rewards. This will cause the robot to try to find other actions in the database that will allow it to make progress faster than $6cm/second$. But in many parts of the maze the human did not perform actions that made progress at that rate. Therefore these negative rewards cause the robot to try many actions that may

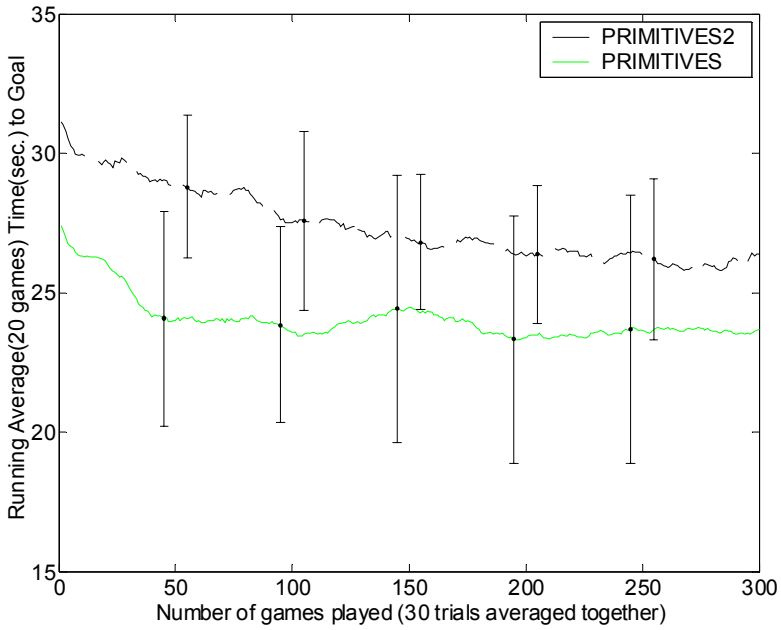

Figure 70: A comparison of the PRIMITIVES robot with the PRIMITIVES2 robot, a PRIMITIVES robot that has been modified to operate as close to the Q learning robot as possible.

not be as good as the one it has originally chosen. On the other hand, it may find an action better than the originally selected one.

The PRIMITIVES robot receives large negative rewards if a selected action could not be initialized and immediately the data points that determined that action are moved far from the query point. But the PRIMITIVES2 robot only receives the reward according to the time and distance traveled. Therefore since the PRIMITIVES2 robot will only make one selection per time cycle of $1/60$ seconds, if the environment is not changing, the robot would receive a reward of -1.0 for choosing an action that cannot be performed. The robot may now need to make this bad selection many times to move the data points sufficiently far from the query point.

A robot using our learning system but given no prior observed information cannot take any actions in the environment and therefore there are no results shown this type of robot in this section. This is a drawback of our system but from Figure 69 it can be seen that if this was a hardware implementation and the robot had to operate in real time, it would be unrealistic to learn this task using the direct Q learning algorithm.

7.4.2 Framework Discussion

The chosen testing domains were very effective in demonstrating the operation and benefits of our learning system. This section focuses on the characteristics of the testing domains that allowed them to work well with our learning system. This section will also discuss other domains that may work well and domains that may not work so well with our learning system. The title of the framework, "Learning from Observation Using Primitives," contains two key words that help to immediately identify tasks that will work well with the framework.

The first key word is "observation." The learning system must first be able to observe the actions that occur within a task to have the ability to learn those actions. If the robot cannot observe the actions of a task prior to performing the task, our framework is useless. The comparison of our learning system with a direct RL system highlights the value of providing the robots with observed information. Within our air hockey environment it is easy to observe the objects using a vision system. From these observations the dynamics such as object position and velocity can be computed. But from this information we cannot observe the force that is applied to the objects or the muscle activations that cause the arm to make the correct movements. Therefore in our framework the robot must have other methods to learn these unobservable items. Section 6.2.1 shows how the humanoid robot learned to position its arm and learned the timing of its movements while practicing the task. To fully perform a task the robot must also have previously observed the task in a majority of the possible task configurations. Chapter 6.1.5.1 shows how the marble maze playing robot has trouble when it goes to states that are not covered by previous observations.

The other key word is "primitives." Our framework is designed for tasks that can be broken down into smaller definable units. Tasks such as tennis, ping-pong, and soccer have been extensively studied and a variety of primitives have been defined such as **back hand** and **dribble**. Continuous tasks such as walking or riding a bicycle may contain primitives but it is not yet clear how these tasks can easily be divided into smaller units. Other tasks such as cleaning dishes and folding cloths have not been extensively studied but may contain primitives such as **clean a pot** and **fold a shirt**. The large variety of item configurations that a robot can be exposed to within these tasks will increase the difficulty of learning the primitives and stresses the importance of having the ability to encode

the primitive in a general way so that it will work even for objects that have not been seen prior to performing the task. Our learning framework requires the set of primitives for the task to be predefined. Our primitives were defined by a task expert. It is important that the primitives cover enough of the possible situations the robot may find itself in during the performance of the task to allow the robot to complete the task.

Our framework encodes actions in a database that is queried by the state. Therefore an observable task state representation is needed. A task such as folding cloths may have states such as "a shirt is observed" or "socks are observed". But if action selection is determined at a lower level, such as by the configuration of the item, it not clear how such a state space representation can be created. The actions in our framework are selected by comparing previously observed situations with the situation the robot is currently in. Therefore the state representation must support qualitative comparison so that it can be determined which of two previously observed situations is closest to the currently observed situation.

The performance of the learner is dependent on the performance of the teacher. The goal of this research is not to create robots that operate optimally in the environment, but to create robots that can learn to perform proficiently in a short time. Even though the robot's first attempt is to perform the same actions as the teacher, it is not a requirement to do so. If the robot, while practicing, performs an action that was not previously observed, but makes progress towards the goal, the robot will try to perform the same action in the future. This can be seen in the performance of the marble maze robot shown in Figure 43. In the three observed games the human maneuvers the marble below hole 14. During practice the marble falls into hole 14 and the robot learns that it can more easily maneuver the marble around the top of hole 14. The human player did not even know this action was possible until the player observed the action discovered by the robot.

CHAPTER VIII

SUMMARY

Surely goodness and love will follow me all the days of my life,
and I will dwell in the house of the LORD forever.
- Psalm 23:6 (NIV) [96]

This research presents an integrated system that combines a variety of learning and robot control methods. This thesis presented a framework for having robots learn from observation and practice using primitives. It showed how various learning mechanisms within the framework give our robots the abilities needed to operate in the testing domains. This chapter offers some suggestions of further research directions that can improve upon our learning system and presents the contributions of this research.

8.1 Future Directions

How to structure learning: Chapter 1 explains our motivation in using primitives in robot learning. This motivation leads us to seek out a structure for robot learning that supports fast learning from observation and practice and has the ability to quickly adapt to environment changes. This thesis uses the air hockey task as a case study in which we show how action generation has been structured to make the most use of the observed information, Section 5.1.3. The action generation module contains models that allow the robot to learn specific skills that may generalize to other tasks. This research presents the results of training these models from information obtained whenever a shot is observed. By structuring the problem in the manner that we have, the robot can learn and improve the performance of the models even when only a partial shot sequence is observed. The *impact* model, for example, can learn about the impact interaction whenever a paddle-puck collision is observed. The ultimate outcome of the shot is irrelevant to this model. The same is true for the *puck motion* and *robot* models. These models can be trained whenever the puck is seen moving from the hit area to the goal area or the robot attempts to make a shot.

Further subdivision of the models might be possible. There are two *puck motion* models that

provide the desired velocity vector; one for straight shots and one for bank shots. The straight shot *puck motion* model provides the velocity direction to reach a point without hitting a wall. The bank shot model includes the interaction of the puck with one wall. The bank shot has at least three clear segments: 1) the puck moves from the hit point to the wall, 2) undergoes a change in velocity during the wall-puck collision, and 3) travels to the target position. Segments 1 and 3 are the same as the straight shot model. Therefore it may be possible to have the system learn faster and be more accurate by structuring the problem to have a no-collision *puck motion* model and a *wall-puck collision* model that coordinate to provide the needed information for a bank shot.

Our structure is also organized so that the models are not specifically tied to a single task. It is absolutely necessary for humanoid robots to have the ability to collect and organize learned information in a way in which it can be reused. The impact model, for example, provides the robot with a lot of basic information on the effect a large moving object has on a smaller one. Perhaps this model could be used as a starting point for learning similar interactions such as the effect that a swinging bat has on a ball that is being hit.

When and What to generalize? Combining the left and right bank shots into one learning module provides the robot with more training data for bank shots. But should this be done? Can a left bank shot and a right bank shot be transformed to a standard shot? If the board is symmetric, this approach is successful as shown in Chapter 5.2.4. But what if the board is not symmetric or is tilted to the side? The robot should have methods to detect when information can be combined and when it cannot.

One such method would be for the robot to observe its own performance in the environment and evaluate the effectiveness of using information in all situations. In the air hockey environment, for example, it can compare the results of using the combined information for left and right bank shots verses using the information for left and right shots separately.

When To Forget: This research discusses and presents various items that robots can learn while they are operating in dynamic environments. We decided when information was added to models and when data is to be forgotten and replaced with new data. A more appealing approach would be for the robot to have control over its own learning and decide when models need to be trained further or replaced. If the robot has the ability to continually evaluate its movements and the outcome of its

actions, it can use this information to decide if previously learned models are no longer accurate and should be updated or replaced. In air hockey, for example, if the board is moved during the game, the robot should immediately notice an error in its paddle movements. It can then attempt to add new data to the *robot* model in an effort to have the model adapt to the new board position. When data is added to the current *robot* model, the data can also be use to train an entirely new, off-line, *robot* model. If the current model is not adapting to the changed environment, it can be replaced by the new model.

Automatically discovering primitives: In many research fields there is interest in methods robots can use to automatically define a library of primitives or actions from observing the performance of a task [43, 55, 84]. We have manually defined the library of primitives in this research but we also have a strong interest in automating this process. The implantations presented in this thesis gives insight into the type of information that a robot would need to know about and search for. Our method of breaking the learning problem into small independent models can assist in automatically discovering primitives. Wolpert and Kawato's [133] research of models that have the ability to learn forward and inverse models of the environment provides insight into how the models can be configured to predict the events occurring in the environment. Figure 27 showed how a primitive can be composed of a sequence of model activations. The environment state can be observed and recorded at the beginning and end of this sequence to provide all the information needed for a robot using our framework.

These forward models can be used is to describe which events are occurring in the environment as they are fed the observed data. The sequence of model activations can be recorded as the task is observed. The research of Kaminka et al. [63] on learning the sequential behavior of teams from observation gives some ideas into how the sequential list can be represented and used to discover primitives in the environment. For example, a sequence of model activations that are seen reoccurring many times may be an appropriate primitive. By structuring the models to be general, they may also be observing other environments. This has the effect that as the robot learns more models, it has an increased ability to discover new primitives.

Creating a Planner: The databases created from observing the task provide information about the outcome of actions performed during the task. Within our framework this information is used by

the robot to select an action for the current observed situation. But if the robot has confidence about performing the actions and obtaining the subgoals presented by the observed data, the robot can use this information to plan a sequence of actions. Since our robots operate in dynamic environments, they must make decisions quickly. Having the ability to plan ahead allows them to plan while a primitive is being performed and increases the time they have available for making decisions.

Manipulating the database: The size of the primitive databases used within this research was controlled by the amount of observed information. If the robots are presented continuously with data, their databases will continue to grow. Our primitive selection and subgoal generation methods require that this information is stored, so the robot will soon have difficulty computing actions due to the large number of data points that must be processed. Therefore the robots need methods for adding new data to the database. Many observations may be redundant and the robot can safely ignore them. There may be data points in the database that encode actions that are no longer appropriate or have proven in the past to be ineffective and are therefore no longer used. The robot also needs methods to remove information such as this.

Create local representations: The robots presented in this thesis compute actions based on the global state of the environment. This design decision restricts their learned primitive selection and subgoal generation knowledge to a single task configuration. A robot learning to select primitives in a maze, for example, cannot use that information when presented with a maze with a different configuration. But if the robot could represent the task space in a way that allows mapping situations from one location to another, it would have the ability to reuse learned primitive selection knowledge in various locations. In the marble maze for example, the state of the environment could be that there are walls on both sides of the marble and there is a corner in front of the marble. If the robot knows that it should roll into the corner when presented with this configuration, the robot now knows what action to perform whenever it observes this local situation, regardless of the global maze configuration.

8.2 Contributions

This thesis addresses many of the challenges of using primitives that were presented in Chapter 2. Much of the research that uses primitives on actual robots is being performed in the context of

assembly and mobile robots. In these environments the primitive sequence is usually predefined and the robots have sufficient time to switch between them. The environments presented in this thesis are very dynamic and have continuous state spaces. Due to adversaries and random environment conditions, the primitive performance sequence cannot be fully predefined and the robot must decide very quickly on a primitive type to use. The algorithms presented in this thesis provide a method in which a primitive type and subgoal can be selected based on the state of the environment and data collected from observing a human perform in the environment.

This thesis explored learning methods that can make effective use of observed information and have the ability to learn through practice. The types of information that can be learned from observing others and that which cannot has been highlighted. This research presents a method of breaking the learning problem into small learning models. The individual models have more opportunities to learn and can therefore more quickly become experts. The use of the small models is also a step toward creating models that can be learned in one task and used in other similar tasks.

This thesis provides many insights into the use of primitives and observation data to increase the learning rate of robots. The framework presented in this thesis has proven to be a useful tool in which to conduct learning from observation research. The learning methods represent a significant step towards building robots that learn and interact with humans in a human-like way.

REFERENCES

[1] ABOAF, E., DRUCKER, S., and ATKESON, C., "Task-level robot learning: Juggling a tennis ball more accurately," in *IEEE International Conference on Robotics and Automation*, (Scottsdale, AZ), pp. 1290–1295, 1989.

[2] ALDRIDGE, H., BLUETHMANN, W., AMBROSE, R., and DIFTLER, M., "Control architecture for the robonaut space humanoid," in *First IEEE-RAS International Conference on Humanoid Robotics (Humanoids-2000)*, 2000.

[3] ALOIMONOS, Y., "Active vision revisited," in *Active Perception*, pp. 1–18, Lawrence Erlbaum Associates, 1993.

[4] AMBROSE, R., ALDRIDGE, H., ASKEW, R., BURRIDGE, R., BLUETHMANN, W., DIFTLER, M., LOVCHIK, C., MAGRUDER, D., and REHNMARK, F., "Robonaut: Nasa's space humanoid," *IEEE Intelligent Systems*, vol. 15, pp. 57–63, July / Aug 2000.

[5] "Amplified Bible." http://www.Lockman.org, 2004.

[6] AN, C. H., ATKESON, C. G., and HOLLERBACH, J. M., *Model-Based Control of a Robot Manipulator*. Cambridge, MA: MIT Press, 1988.

[7] ARKIN, R. C., *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1998.

[8] ASADA, H. and ASARI, Y., "The direct teaching of tool manipulation skills via the impedance identification of human motions," in *IEEE Int'l Conf. on Robotics and Automation*, pp. 1269–1274, 1988.

[9] ATKESON, C. G., HALE, J. G., POLLICK, F., RILEY, M., KOTOSAKA, S., SCHAAL, S., SHIBATA, T., TEVATIA, G., UDE, A., VIJAYKUMAR, S., and KAWATO, M., "Using humanoid robots to study human behavior," *IEEE Intelligent Systems*, vol. 15, no. 4, pp. 46–55, 2000.

[10] ATKESON, C. G., MOORE, A. W., and SCHAAL, S., "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, pp. 11–73, 1997.

[11] ATKESON, C. G. and SCHAAL, S., "Robot learning from demonstration," in *Proc. 14th International Conference on Machine Learning*, pp. 12–20, Morgan Kaufmann, 1997.

[12] ATKESON, C. G. and SCHAAL, S., "Robot learning from demonstration," in *Proceedings of the 1997 International Conference on Machine Learning (ICML97)* (FISHER, JR., D. H., ed.), pp. 12–20, Morgan Kaufmann, 1997.

[13] BAKKER, P. and KUNIYOSHI, Y., "Robot see, robot do: An overview of robot imitation," in *AISB96 Workshop on Learning in Robots and Animals*, pp. 3–11, 1996.

[14] BALCH, T., "Clay: Integrating motor schemas and reinforcement learning," Tech. Rep. GIT-CC-97-11, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, March 1997.

[15] BALCH, T., BOONE, G., COLLINS, T., FORBES, H., MACKENZIE, D., and SANTAMARIA, J., "Io Ganymede and Callisto: A multiagent robot trash-collecting team," *AI Magazine*, vol. 16, pp. 39–51, Summer 1995.

[16] BALLARD, D., "Animate vision," *Artificial Intelligence*, vol. 48, pp. 57–86, 1991.

[17] "Banryu homepage." http://www.banryu.jp/.

[18] BARTO, A. and MAHADEVAN, S., "Recent advances in hierarchical reinforcement learning," *Discrete Event Systems*, vol. 13, pp. 41–77, 2003.

[19] BENSON, S. and NILSSON, N., "Reacting, planning, and learning in an autonomous agent," in *Machine Intelligence, 14*, 1995.

[20] BENTIVEGNA, D. C. and ATKESON, C. G., "Using primitives in learning from observation," in *First IEEE-RAS International Conference on Humanoid Robotics (Humanoids-2000)*, 2000.

[21] BENTIVEGNA, D. C., UDE, A., ATKESON, C. G., and CHENG, G., "Humanoid robot learning and game playing using PC-based vision," in *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, (Switzerland), 2002.

[22] BILLARD, A. and SCHAAL, S., "Robust learning of arm trajectories through human demonstration," in *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Maui, HI, USA), pp. 734–739, 2001.

[23] BISHOP, B., SHIRKEY, P., and SPONG, M., "An experimental testbed for intelligent control," in *American Control Conference, Seattle*, 1995.

[24] BLACKBURN, M., EVERETT, H., and LAIRD, R., "After action report to the joint program office: Center for the robotic assisted search and rescue (crasar) related efforts at the world trade center," Tech. Rep. 3141, Space and Naval Warfare Systems Center, San Diego, CA, August 2002.

[25] BLINN, J. F. and NEWELL, M. E., "Texture and reflection in computer generated images," *Communications of the ACM*, vol. 19, pp. 542–547, Oct. 1976.

[26] BROOKS, R. and FLYNN, A., "Fast, cheap, and out of control; a robot invasion of the solar system," *Journal of the British Interplanetary Society*, vol. 42, no. 10, pp. 478–485, 1989.

[27] BROOKS, R. A., "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14–23, 1986.

[28] BROOKS, R. A., "A robot that walks: Emergent behaviors from a carefully evolved network," in *Artificial Intelligence at MIT, Expanding Frontiers* (WINSTON, P. H. and SHELLARD, S. A., eds.), pp. 28–39, MIT Press, Cambridge, MA, USA, 1990.

[29] BURKHARD, H. D., DUHAUT, D., FUJITA, M., LIMA, P., MURPHY, R., and ROJAS, R., "The road to robocup 2050," *IEEE Robotics & Automation Magazine*, vol. 9, pp. 31–38, June 2002.

[30] CLARK, R. J., ARKIN, R. C., and RAM, A., "Learning momentum: On-line performance enhancement for reactive systems," in *IEEE Int'l Conf. on Robotics and Automation*, (Nice, France), pp. 111–116, 1991.

[31] DAVIDS, A., "Urban search and rescue robots: from tragedy to technology," *IEEE Robotics & Automation Magazine*, vol. 17, pp. 81–83, Mar/Apr 2002.

[32] DELSON, N. and WEST, H., "The use of human inconsistency in improving 3d robot trajectories," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Munchen), pp. 1248–1255, 1994.

[33] DELSON, N. and WEST, H., "Robot programming by human demonstration: adaptation and inconsistency in constrained motion," in *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 30 –36, 1996.

[34] DEMIRIS, J. and HAYES, G., "Imitation as a dual-route process featuring predictive and learning components: a biologically-plausible computational model," in *Imitation in Animals and Artifacts* (DAUTENHAHN, K. and NEHANIV, C., eds.), ch. 13, MIT Press, 2001.

[35] DI PELLEGRINO, G., FADIGA, L., FOGASSI, L., GALLESE, V., and RIZZOLATI, G., "Understanding motor events: a neurophysiological study," in *Experimental Brain Research, 91:176-180*, 1992.

[36] DIETTERICH, T. G., "The MAXQ method for hierarchical reinforcement learning," in *Proc. 15th International Conf. on Machine Learning*, pp. 118–126, Morgan Kaufmann, San Francisco, CA, 1998.

[37] DILLMANN, R., FRIEDRICH, H., KAISER, M., and UDE, A., "Integration of symbolic and subsymbolic learning to support robot programming by human demonstration," in *Robotics Research: The Seventh International Symposium* (GIRALT, G. and HIRZINGER, G., eds.), pp. 296–307, Springer, NY, 1996.

[38] DILLMANN, R., KAISER, M., and UDE, A., "Acquisition of elementary robot skills from human demonstration," in *International Symposium on Intelligent Robotics Systems*, (Pisa, Italy), 1995.

[39] ERDMANN, M. A. and MASON, M. T., "An exploration of sensorless manipulation," *IEEE Journal of Robotics and Automation*, vol. 4, pp. 369–379, 1988.

[40] FALOUTSOS, P., VAN DE PANNE, M., and TERZOPOULOS, D., "Composable controllers for physics-based character animation," in *Proceedings of SIGGRAPH 2001*, (Los Angeles, CA, USA), pp. 251–260, 2001.

[41] FIKES, R., HART, P., and NILSSON, N., "Learning and executing generalized robot plans," in *Artificial Intelligence Vol. 3*, 1972.

[42] FIKES, R. E. and NILSSON, N. J., "STRIPS: A new approach to the application for theorem proving to problem solving," in *Advance Papers of the Second International Joint Conference on Artificial Intelligence*, (Edinburgh, Scotland), pp. 608–620, 1971.

[43] FOD, A., MATARIC, M., and JENKINS, O., "Automated derivation of primitives for movement classification," in *First IEEE-RAS International Conference on Humanoid Robotics (Humanoids-2000)*, (MIT, Cambridge, MA), 2000.

[44] FORBES, J. and ANDRE, D., "Representations for learning control policies," in *ICML-2002 Workshop on Development of Representations*, 2002.

[45] FUJITA, M., KUROKI, Y., ISHIDA, T., and DOI, T. T., "A small humanoid robot SDR-4X for entertainment applications," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, vol. 2, pp. 938–943, July 2003.

[46] GRUDIC, G. and LAWRENCE, P., "Human-to-robot skill transfer using the spore approximation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2962–2967, 1996.

[47] GRUVER, W., SOROKA, B., CRAIG, J., and TURNER, T., "Evaluation of commercially available robot programming languages," in *13th Int'l Symp. on Industrial Robots*, pp. 12–58, 1983.

[48] HAYES, G. and DEMIRIS, J., "A robot controller using learning by imitation," in *A. Borkowski and J. L. Crowley (Eds.), Proceedings of the 2nd International Symposium on Intelligent Robotic Systems*, pp. 198–204, 1994.

[49] HIRAI, K., HIROSE, M., HAIKAWA, Y., and TAKENAKA, T., "The development of Honda humanoid robot," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1321–1326, May 1998.

[50] HIRZINGER, G., "Learning and skill acquisition," in *Robotics Research: The Seventh International Symposium* (GIRALT, G. and HIRZINGER, G., eds.), pp. 277–278, Springer, NY, 1996.

[51] HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., and O'BRIEN, J. F., "Animating human athletics," in *Proceedings of Siggraph '95*, pp. 71–78, 1995.

[52] HOVLAND, G., SIKKA, P., and MCCARRAGHER, B., "Skill acquisition from human demonstration using a hidden markov model," in *Proceedings of IEEE International Conference on Robotics and Automation*, (Minneapolis, MN), pp. 2706–2711, 1996.

[53] HUBER, M., "A hybrid architecture for hierarchical reinforcement learning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3290–3295, 2000.

[54] HUGUES, L. and DROGOUL, A., "Shaping of robot behaviors by demonstration," in *First International Workshop on Epigenetic Robotics*, 2001.

[55] IBA, W., "Learning to classify observed motor behavior," in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 732–738, 1991.

[56] IKEUCHI, K., MIURA, J., SUEHIRO, T., and CONANTO, S., "Designing skills with visual feedback for APO," in *Robotics Research: The Seventh International Symposium* (GIRALT, G. and HIRZINGER, G., eds.), pp. 308–320, Springer, NY, 1996.

[57] IMAMIZU, H., MIYAUCHI, S., TAMADA, T., SASAKI, Y., TAKINO, R., PUETZ, B., YOSHIOKA, T., and KAWATO, M., "Human cerebellar activity reflecting an acquired internal model of a novel tool," *Nature*, vol. 403, pp. 192–195, 2000.

[58] JENKINS, O., MATARIC, M., and WEBER, S., "Primitive-based movement classification for humanoid imitation," in *First IEEE-RAS International Conference on Humanoid Robotics (Humanoids-2000)*, (MIT, Cambridge, MA), 2000.

[59] KAELBLING, L. P., "Hierarchical learning in stochastic domains: Preliminary results," in *International Conference on Machine Learning*, pp. 167–173, 1993.

[60] KAELBLING, L. P., "Learning to achieve goals," in *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1094–1099, 1993.

[61] KAISER, M. and DILLMANN, R., "Building elementary skills from human demonstration," in *Proceedings of the IEE International Conference on Robotics and Automation*, pp. 2700–2705, 1996.

[62] KAISER, M., RETEY, A., and DILLMANN, R., "Robot skill acquisition via human demonstration," in *Proceedings of the International Conference on Advanced Robotics (ICAR '95)*, 1995.

[63] KAMINKA, G., FIDANBOYLU, M., CHANG, A., and M.VELOSO, "Learning the sequential coordinated behavior of teams from observation," in *Proceedings of the 2002 International RoboCup Symposium*, 2002.

[64] KAMON, I., FLASH, T., and EDELMAN, S., "Learning visually guided grasping: A test case in sensorimotor learning," in *IEEE Transactions on System, Man and Cybernetics*, vol. 28(3), pp. 266–276, May 1998.

[65] KANEKO, K., KANEHIRO, F., KAJITA, S., YOKOYAMA, K., AKACHI, K., KAWASAKI, T., OTA, S., and ISOZUMI, T., "Design of prototype humanoid robotics platform for HRP," in *IEEE/RSJ International Conference on Intelligent Robots and System*, vol. 3, pp. 2431–2436, September 2002.

[66] KANG, S., *Robot instruction by human demonstration*. PhD thesis, Robotics Institute, Carnegie Mellon University, 1994.

[67] KANG, S. and IKEUCHI, K., "A grasp abstraction hierarchy for recognition of grasping tasks from observation," in *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, (Yokohama, Japan), pp. 621–628, July 1993.

[68] KANG, S. B. and IKEUCHI, K., "Toward automatic robot instruction from perception: Recognizing a grasp from observation," *IEEE International Journal of Robotics and Automation*, vol. 9, no. 4, pp. 432–443, 1993.

[69] KUNIYOSHI, Y., "The science of imitation – towards physically and socially grounded intelligence." RWC Technical Report, TR-94001, Special Issue, 1994.

[70] KUNIYOSHI, Y., INABA, M., and INOUE, H., "Learning by watching: Extracting reusable task knowledge from visual observation of human performance," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 6, pp. 799–822, 1994.

[71] KUNIYOSHI, Y. and INOUE, H., "Qualitative recognition of ongoing human action sequences," in *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1600–1609, 1993.

[72] LARSON, A. and VOYLES, R., "Automatic training data selection for sensorimotor primitives," in *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Maui, HI, USA), pp. 871–876, 2001.

[73] LEE, J. B. and ARKIN, R. C., "Learning momentum: Integration and experimentation," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, (Seoul, Korea), pp. 1975–1980, 2001.

[74] LIKHACHEV, M. and ARKIN, R. C., "Spatio-temporal case-based reasoning for behavioral selection," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, (Seoul, Korea), pp. 1627–1634, 2001.

[75] LIMA, P., BALCH, T., FUJITA, M., ROJAS, R., VELOSO, M., and YANCO, H. A., "Robocup 2001," *IEEE Robotics and Automation Magazine*, vol. 9, pp. 20–30, June 2002.

[76] LIN, L. J., "Hierarchical learning of robot skills by reinforcement," in *Proceedings of the 1993 International Joint Conference on Neural Networks*, pp. 181–186, 1993.

[77] MAHADEVAN, S. and CONNELL, J., "Automatic programming of behavior-based robots using reinforcement learning," in *AAAI, Vol. 2*, pp. 768–773, 1991.

[78] "Mars exploration rover mission homepage." http://marsrovers.jpl.nasa.gov/.

[79] MATARIC, M., "Sensory-motor primitives as a basis for imitation: Linking perception to action and biology to robotics," in *Imitation in Animals and Artifacts*, MIT Press, 2000.

[80] MATARIC, M., ZORDAN, V., and WILLIAMSON, M., "Making complex articulated agents dance," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 1, pp. 23–43, 1999.

[81] MATARIC, M. J., WILLIAMSON, M., DEMIRIS, J., and MOHAN, A., "Behavior-based primitives for articulated control," in *Fifth International Conference on Simulation of Adaptive Behavior (SAB-98)*, pp. 165–170, MIT Press, 1998.

[82] "Matlab homepage." http://www.mathworks.com/.

[83] MCGOVERN, A., "acQuire-macros: An algorithm for automatically learning macro-actions," in *NIPS'98 Workshop on Abstraction and Hierarchy in Reinforcement Learning*, 1998.

[84] MCGOVERN, A. and BARTO, A. G., "Automatic discovery of subgoals in reinforcement learning using diverse density," in *Proceedings of the 18th International Conference on Machine Learning*, pp. 361–368, Morgan Kaufmann, San Francisco, CA, 2001.

[85] MICHELMAN, P. and ALLEN, P., "Forming complex dextrous manipulations from task primitives," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3383–3388, 1994.

[86] MISHKIN, A., MORRISON, J., NGUYEN, T., STONE, H., COOPER, B., and WILCOX, B., "Experiences with operations and autonomy of the Mars pathfinder microrover," in *IEEE Aerospace Conference*, 1998.

[87] MORI, T., TSUJIOKA, K., and SATO, T., "Human-like action recognition system on whole body motion-captured file," in *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Maui, Hawaii, USA), 2001.

[88] MORIMOTO, J. and DOYA, K., "Hierarchical reinforcement learning of low-dimensional subgoals and high-dimensional trajectories," in *Proceedings of the 5th International Conference on Neural Information Processing*, vol. 2, pp. 850–853, 1998.

[89] MORROW, J., *Programming Robotic Assembly Skills*. PhD thesis, Carnagie Mellon Univesity, May 1997.

[90] MORROW, J. and KHOSLA, P., "Manipulation task primitives for composing robot skills," in *IEEE International Conference on Robotics and Automation*, (Albuquerque, NM, USA), pp. 3354–9, 1997.

[91] MORROW, J., NELSON, B., and KHOSLA, P., "Vision and force driven sensorimotor primitives for robotic assembly skills," in *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems (IROS-95)*, vol. 3, (Pittsburg, PA), pp. 234–240, 1995.

[92] MURPHY, R. R., CASPER, J., and MICIRE, M., "Potential tasks and research issues of mobile robots in RoboCup," *RoboCup-2000: Robot Soccer World Cup IV*, vol. 9, no. 4, pp. 339–344, 2001.

[93] NAKAWAKI, D., JOO, S., and MIYAZAKI, F., "Dynamic modeling approach to gymnastic coaching," in *Proceedings of the 1998 IEEE International Conference on Robotics and Automation (ICRA98)*, 1998.

[94] NILSSON, N. J., "A mobile automaton: An application of artificial intelligence techniques," in *Proceedings of the International Joint Conference on Artificial Intelligence*, (Washington, D. C.), pp. 509–520, 1969.

[95] NILSSON, N. J., "Teleo-reactive programs for agent control," *Journal of Artificial Intelligence Research*, vol. 1, pp. 139–158, 1994.

[96] "New International Version Bible." http://www.ibs.org, 2004.

[97] PARK, J., PARTRIDGE, C. B., and SPONG, M. W., "Neural network based state prediction for strategy planning of an air hockey robot," *Journal of Robotic Systems*, vol. 18, pp. 187–196, April 2001.

[98] PARR, R. and RUSSELL, S., "Reinforcement learning with hierarchies of machines," in *Advances in Neural Information Processing Systems* (JORDAN, M. I., KEARNS, M. J., and SOLLA, S. A., eds.), vol. 10, The MIT Press, 1998.

[99] PARTRIDGE, C. B. and SPONG, M. W., "Control of planar rigid body sliding with impacts and friction," in *International Journal of Robotics Research*, pp. 336–348, April 2000.

[100] PEARCE, M., ARKIN, R. C., and RAM, A., "The learning of reactive control parameters through genetic algorithms," in *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Raleigh, NC, USA), pp. 130–137, 1992.

[101] PRICE, B. and BOUTILIER, C., "Imitation and reinforcement learning in agents with heterogeneous actions," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, 2000.

[102] PRICE, B. and BOUTILIER, C., "Implicit imitation in multiagent reinforcement learning," in *Proc. 16th International Conf. on Machine Learning*, pp. 325–334, Morgan Kaufmann, San Francisco, CA, 1999.

[103] RAM, A. and SANTAMARIA, J., "A multistrategy case-based and reinforcement learning approach to self-improving reactive control systems for autonomous robotic navigation," in *Proceedings of the Second International Workshop on Multistrategy Learning*, (Harpers Ferry, WV), 1993.

[104] RIEZENMAN, M. J., "Robots stand on own two feet," *IEEE Spectrum*, vol. 39, pp. 24–25, Aug 2002.

[105] RILEY, P., VELOSO, M., and KAMINKA, G., "An empirical study of coaching," *Distributed Autonomous Robotic Systems*, vol. 5, pp. 215–224, 2004.

[106] "RoboCup homepage." http://www.robocup.org/.

[107] RUSSELL, S. J. and NORVIG, P., *Artificial Intelligence: A Modern Approach*. PrenticeHall, 1995.

[108] RYAN, M. and REID, M., "Learning to fly: An application of hierarchical reinforcement learning," in *Proc. 17th International Conf. on Machine Learning*, pp. 807–814, Morgan Kaufmann, San Francisco, CA, 2000.

[109] RYAN, M. R. K. and PENDRITH, M. D., "RL-TOPs: an architecture for modularity and re-use in reinforcement learning," in *Proc. 15th International Conf. on Machine Learning*, pp. 481–487, Morgan Kaufmann, San Francisco, CA, 1998.

[110] SACERDOTI, E., "Planning in a hierarchy of abstraction," *Artificial Intelligence*, vol. 5, no. 2, pp. 115–135, 1974.

[111] SANTAMARIA, J., SUTTON, R., and RAM, A., "Experiments with reinforcement learning in problems with continuous state and action spaces," *Adaptive Behavior*, vol. 6, no. 2, 1998.

[112] SCHAAL, S., "Is imitation learning the route to humanoid robots?," *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[113] SCHAAL, S., ATKESON, C., and VIJAYAKUMAR, S., "Scalable locally weighted statistical techniques for real time robot learning," *Applied Intelligence - Special issue on Scalable Robotic Applications of Neural Networks*, vol. 17, no. 1, pp. 49–60, 2002.

[114] SCHAAL, S., KOTOSAKA, S., and STERNAD, D., "Nonlinear dynamical systems as movement primitives," in *International Conference on Humanoid Robotics*, (Cambridge, MA), 2000.

[115] SCHAAL, S., "Learning from demonstration," in *Advances in Neural Information Processing Systems* (MOZER, M. C., JORDAN, M. I., and PETSCHE, T., eds.), vol. 9, p. 1040, The MIT Press, 1997.

[116] SCHMIDT, R. A., *Motor Learning and Control*. Champaign, IL: Human Kinetics Publishers, 1988.

[117] SINGH, S. P., "Reinforcement learning with a hierarchy of abstract models," in *National Conference on Artificial Intelligence*, pp. 202–207, 1992.

[118] SMART, W. D. and KAELBLING, L. P., "Practical reinforcement learning in continuous spaces," in *Proc. 17th International Conf. on Machine Learning*, pp. 903–910, Morgan Kaufmann, San Francisco, CA, 2000.

[119] SMITHERS, T. and MALCOLM, C., "Programming robotics assembly in terms of task achieving behavioural modules," in *International Advanced Robotics Programme, Second Workshop on Manipulators, Sensors, and Steps towards Mobility*, pp. 15.1–15.16, 1988.

[120] STONE, P. and VELOSO, M., "Layered learning," in *European Conference on Machine Learning*, pp. 369–381, 2000.

[121] SUTTON, R., "Integrating architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings of the Seventh International Conference on Machine Learning (ML-90)*, (Austin, TX, USA), pp. 216–224, 1990.

[122] SUTTON, R. and BARTO, A., *Reinforcment Learning: An Introduction*. MIT Press, 1998.

[123] SUTTON, R. S., PRECUP, D., and SINGH, S. P., "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[124] TANIE, T., "Humanoid robot and its application possibility," in *Proceedings of IEEE International Conference Multisensor Fusion and Integration for Intelligent Systems*, July 2003.

[125] THOMAS, U. and WAHL, F. M., "A system for automatic planning, evaluation and execution of assembly sequences for industrial robots," in *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Maui, Hawaii, USA), 2001.

[126] TROXELL, W., "A robotic assembly description language derived from task-achieving behaviors," in *Proceedings of Manufacturing International '90*, (Atlanta, GA), March 1990.

[127] TUNG, C. and KAK, A., "Automatic learning of assembly tasks using a dataglove system," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 1995.

[128] UDE, A. and ATKESON, C. G., "Real-time visual system for interaction with a humanoid robot," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, (Maui, Hawaii), pp. 746–751, October/November 2001.

[129] VIJAYAKUMAR, S. and SCHAAL, S., "Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional spaces," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, (Stanford, CA), 2000.

[130] VIJAYAKUMAR, S., D'SOUZA, A., SHIBATA, T., CONRADT, J., and SCHAAL, S., "Statistical learning for humanoid robots," *Autonomous Robot*, vol. 12, no. 1, pp. 55–69, 2002.

[131] WASSON, G., KORTENKAMP, D., and HUBER, E., "Integrating active perception with an autonomous robot architecture," in *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)* (SYCARA, K. P. and WOOLDRIDGE, M., eds.), (New York), pp. 325–331, ACM Press, 1998.

[132] WILLIAMSON, M., "Postural primitives: Interactive behavior for a humanoid robot arm," in *Fourth International Conference on Simulation of Adaptive Behavior*, (Cape Cod, MA), pp. 124–131, MIT Press, 1996.

[133] WOLPERT, D. M. and KAWATO, M., "Multiple paired forward and inverse models for motor control," *Neural Networks*, vol. 11, no. 7-8, pp. 1317–1329, 1998.

[134] WOOTEN, W. L. and HODGINS, J. K., "Simulating leaping, tumbling, landing and balancing humans," in *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 656–662, 2000.

[135] ZHANG, Z., "A flexible new technique for camera calibration," Tech. Rep. MSR-TR-98-71, Microsoft Research, Microsoft Corporation, Redmond, Washington, December 1998.

[136] ZONFRILLI, F., ORIOLO, G., and NARDI, D., "A biped locomotion strategy for the quadruped robot Sony ERS-210," in *2774 International Conference on Robotics and Automation*, vol. 3, pp. 2678–662, 2002.