Using Primitives in Learning From Observation

Darrin Bentivegna[1,2] and Christopher G. Atkeson[1,2]

[1] ATR-International, Information Sciences Division
2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan
dbent@isd.atr.co.jp
[2] College of Computing, Georgia Institute of Technology
801 Atlantic Drive, GA 30332-0280, USA
cga@cc.gatech.edu

Abstract. This paper describes the use of motor primitives in robot learning from observation. Data is collected while a human performs a task. This data is parsed into small parts of the task called primitives. Modules are created for each primitive that encode the movements required during the performance of the primitive. Another module encodes when and where the low-level primitives are performed. A learning agent then uses these modules to perform the task. The feasibility of this method is currently being tested with an agent that is being taught to play a virtual and a hardware marble maze game and a virtual air hockey game. It is also being tested in teaching a humanoid robot to play air hockey against a human player.

1 Introduction

Human learning is often accelerated by observing a task being performed or attempted by someone else. If robots can be programmed to use such observations to accelerate learning their usability and functionality will be increased and programming and learning time will be decreased. This paper describes research that explores the use of primitives in learning from observation. Our ultimate goal is to show that the use of primitives accelerates learning, and that primitives can be learned automatically by observing a teacher's performance. This paper describes how a set of predefined primitives can be used in learning from observation.

Two environments, each with a virtual and a physical implementation, will be described. One of the tasks is playing air hockey. Figure 1 shows a virtual air hockey game that was created that allows a person to play against a virtual player. Research in this environment is also being performed using a humanoid robot (www.erato.atr.co.jp/DB/) and a camera based tracking system, figure 2. The air hockey playing agent learns its playing behavior from observing the human opponent.

The other task used in this research is a marble-maze game that has been implemented in a virtual environment and a physical implementation figures 5 and 6. For the physical implementation a marble-maze game [13] has been outfitted with stepper motors to move the board and a camera system to supply the position of the ball. The movements of a human player can be captured and a computer can control the board. The methods used to extract information from captured data and how the player agents use this information in these environments will be described.

These domains were chosen because of the ease with which they can be simulated in virtual environments and because they provide a starting point to obtain more information on learning from observation. The physical environments are also small enough to be operated in a laboratory. Since the basic movements in these domains are only in two dimensions, vision and object manipulation are simplified.

2 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 hopeless. In addition, since 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 [1] and learning from observation [4, 5, 9, 12, 6, 8, 10, 11].

It is our hope that primitives can be used to reduce the dimensionality of the learning problem [2, 15]. Primitives are solutions to small parts of a task that can be combined to complete the task. A solution to a task may be made

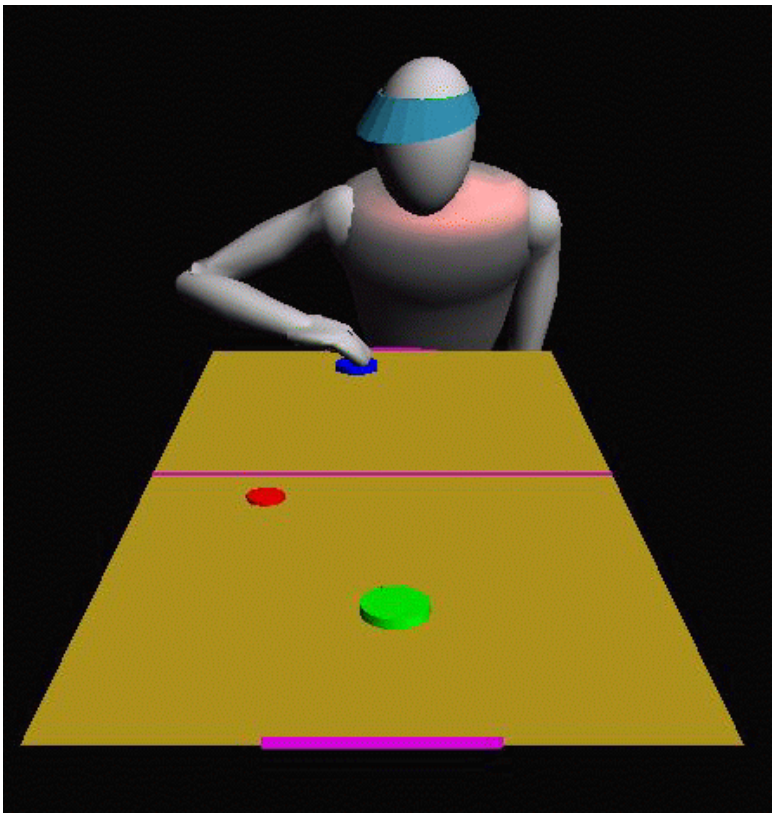

Fig. 1. The virtual air hockey environment. The disc shaped object near the centerline is a puck that slides on the table and bounces off the sides, and the other two disc shaped objects are the paddles. The virtual player controls the far paddle, and a human player controls the closer paddle by moving 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).

up of many primitives. In the air hockey environment, for example, there may be primitives for hitting the puck, capturing the puck, and defending the goal. There are many possible primitives, and it is often possible to break a primitive up into smaller primitives.

In this paper, a human, using domain knowledge, designs the candidate primitives that are to be used. Algorithms have been created to segment the observed behavior into primitives. This segmented data is then used to create modules that the agent uses to decide what primitive to perform and how to perform it.

3 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 for the puck to slide on 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. Previous researchers have used a camera based vision system to collect data in a hardware implementation of air hockey [7, 14]. and a robot arm has been programmed to play air hockey on an actual table [16, 7].

Figure 1 shows the virtual air hockey game that can be played on a computer. The game consists of two paddles, a puck and a board to play on. A human player using a mouse controls one paddle. At the other end is a simulated or virtual player. The code can be obtained at www.cc.gatech.edu/projects/Learning_Research/. The movement of the virtual player has very limited physics incorporated into it. The paddle movement is constrained to operate with a velocity limit. Paddle accelerations are not monitored and therefore can be unrealistically large. The virtual player uses only its arm and hand to position the paddle. For a given desired paddle location, the arm and hand are placed to put the paddle in the appropriate location, and any redundancies are resolved so as to make the virtual

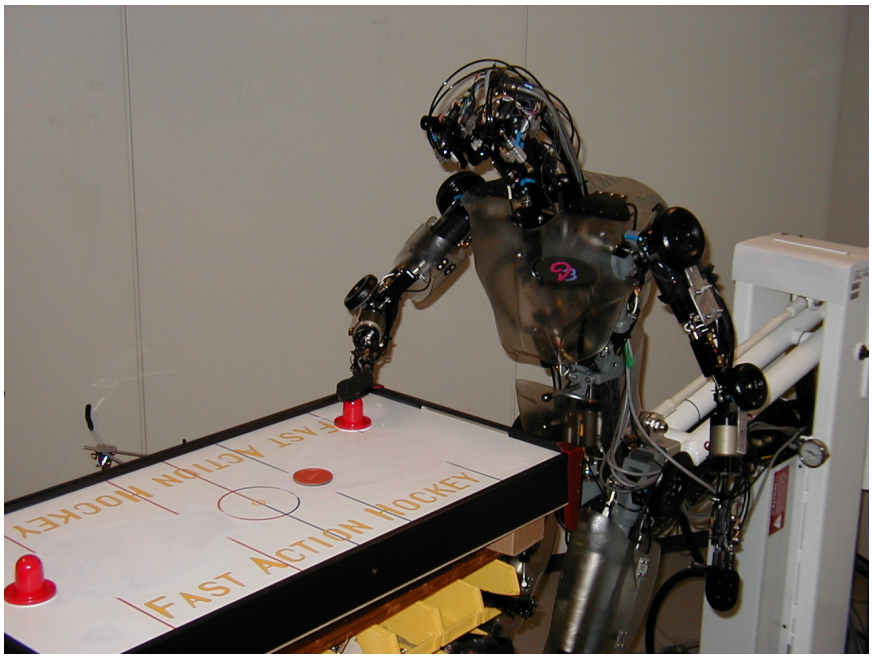

Fig. 2. The hardware air hockey environment.

player look “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 a 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.

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. 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 occurring within sampling intervals are recorded.

The hardware implementation, figure 2, consists of the humanoid robot and a small air hockey table. The robot observes the position of the ball using its onboard cameras, figure 3, and hardware designed to track the position of colored objects in the image. The humanoid's torso is moved during play to extend the reach of the robot. The head is moved so that the playing field is always within view.

Human domain knowledge was used to define a set of primitives to work with initially. Three hit primitives are shown in figure 4. The full list of primitives used is:

- Left Hit: the player hits the puck and it hits the left wall and then travels toward the opponent's goal.
- Straight Hit: the player hits the puck and it travels toward the opponent's goal without hitting the side walls.
- Right Hit: the player hits the puck and it hits the right wall and then travels toward the opponent's goal.
- No Hit: the player deliberately does not hit the puck.
- Prepare: movements made while the puck is on the opposite side from the player.
- Multi-Shot: movements made after a shot is attempted, but while the puck is still on the same side.

3.1 Selecting the Appropriate Primitive

The player agent must decide which primitive to perform for the observed state of the environment. The prepare primitive is performed whenever the puck is on the side opposite the player. In all the remaining primitives the puck is on the same side as the player, so selecting which of the other primitives to perform requires taking into account the position and velocity of the puck.

A module was created to guide selection of primitives, incorporating prior observations of primitives being executed. The context or state in which the human has performed each primitive is extracted from the observed data, and is used by a nearest neighbor lookup process to find the past primitive execution whose context is most similar to the current context. For example the puck's position and velocity when it crossed the centerline is often

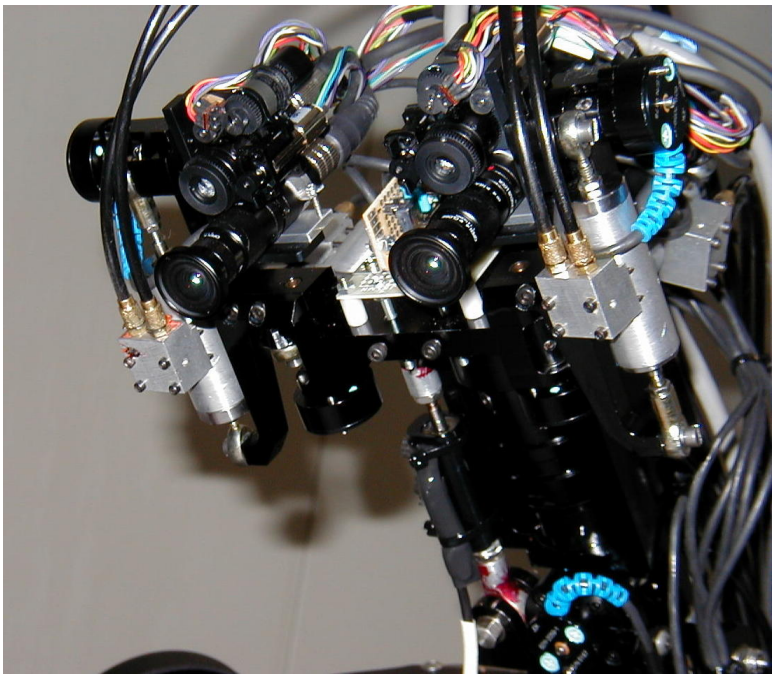

Fig. 3. The head of the cyber human contains two ”eyes,” each made up of a wide angle and narrow angle cameras on pan-tilt mechanisms.

used as the index for a lookup. In this implementation a primitive is selected and then run to completion, before the next primitive is selected and executed. In future implementations we plan to look at systems in which primitives can run concurrently, and interrupt and override other primitives.

Critical events are used to segment the observed data and to decide when a specific primitive is being performed. Critical events are usually rare occurrences. For example, the puck mostly travels in a straight line with a gradually decreasing velocity. Critical events for the puck include collisions, in which the ball speed and direction are rapidly changed. Using critical events, the raw data is segmented into the above primitives. In air hockey, the hit primitives are parameterized by the incoming puck position and velocity (when it crossed the center line), the hit location, the outgoing puck velocity and the target position. To determine the target of the hit, the puck's velocity vector after the hit is made is observed and a physical model is used to determine where the puck would hit the back wall if the opponent did not block it. This use of a physical model enables the learning agent to estimate the target being attempted without the shot having to be completed. The accuracy of the physical model can be critical in producing useful training data. Other methods can be used to reduce the reliance on the physical model, such as only considering shots that have actually hit the back wall without having hit any other walls or paddles.

The parameters for the hit primitives are the desired hit location, the puck's desired post-hit velocity, and the target location. Currently these parameters are returned along with the single nearest neighbor as part of the selected primitive. A future implementation will obtain the parameters by interpolating between parameters of previously executed primitives of the selected type.

3.2 Finding the Right Paddle Motion

Once the primitive to perform has been decided upon, and the parameters are obtained, the agent must then figure out how to move the paddle to accomplish the primitive. This can be done in many ways. Three methods have been tried in the virtual environment; a physical model, neural networks, and kernel regression [3].

The physical model contains a simulation algorithm and computes the required paddle movements to hit the puck to a desired location with the desired output velocity. The computed movement is the minimum movement needed to obtain the correct hit. 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 the physical model produces extremely accurate results but does not take into account the information obtained from the observation. The accuracy of the physical model largely determines the success of this method. If a physical model is not available some other method must be used.

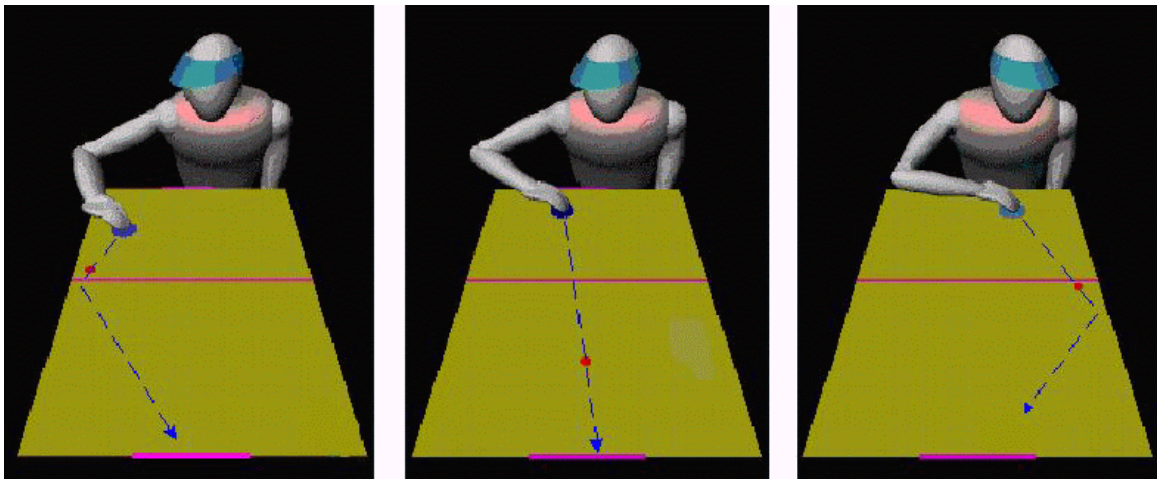

Fig. 4. Three hit primitives being performed by the virtual player: right, straight, and left.

For the neural network and kernel regression methods information is extracted from the captured data so as to have the virtual player move the paddle the way that the human moved the paddle to make a shot. The observed data is segmented into the hit primitives and a database is created that contains the following information:

Input:

- The XY location of the puck when it was hit.
- The velocity components of the puck when it was hit.
- The absolute velocity of the puck just after it is hit.
- The position on the back wall that the puck would hit if unobstructed.

Output:

- The paddle's velocity components at the time of the collision.
- The location of the paddle relative to the puck at the time of contact.

This information is used in a learning module that tells the agent the paddle's velocity components and relative position that are needed to make the desired shot. The query to the learning module is the puck's velocity and desired hit location, the desired velocity of the puck after it is hit, and the desired location to shoot for on the back wall. The learning module then outputs the information needed by the virtual player to make the shot.

3.3 The Prepare and Multi-Shot Primitive

Prepare is the action that is performed by a player when the puck is on the other side of the centerline. In order for the player agent to learn this behavior from the human player a database is created from the observed data that contains the puck's and the human paddle's position and velocity components during the time when the puck is on the other side of the centerline. Kernel regression of this data is used to determine what the virtual player will do when the puck is on the other side of the centerline.

If the teacher did not make a hit for the incoming puck parameters the primitive selection query will return the no-hit primitive. In this case something other then a hit must be performed. It may also be that the virtual player attempts a shot but does not make it correctly and the puck does not return to the other side of the board. In these situations the multi-shot primitive is performed. The prepare and multi-shot primitives give the player agent a more human like appearance.

The learning modules for these primitives are queried at every time cycle during primitive execution and returns the desired position and velocity of the paddle. The database for these primitives contains the velocity and position of the puck, the position and velocity of the player's paddle, and the position and velocity of the opponent's paddle. The position and velocity of the puck and the opponent's paddle are interpolated using kernel regression. This primitive generates the desired paddle position and velocity for the observed state. Once the virtual player starts executing these primitives, it will continue until the puck crosses the centerline.

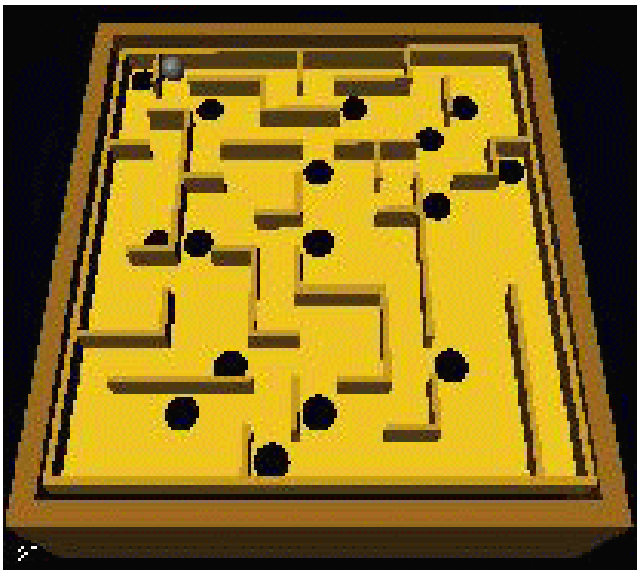

Fig. 5. The virtual marble maze environment.

4 Marble Maze

Primitive learning is also being explored in the marble maze environment in software, figure 5, and on hardware, figure 6. In the marble maze game a player controls a marble through a maze by tilting the board that the marble is rolling on. The board is tilted using two knobs. There are obstacles, in the form of holes, that the marble may fall into, and walls. In both versions the time and the board and ball positions are recorded as a human plays the game.

As in the air hockey environment, primitives are extracted from the captured data using critical events. Once again, a human designed the primitives and created an algorithm to find the primitives in the captured data. The following primitives are currently being explored and are shown in figure 7:

- Roll Wall Stop: The ball rolls along a wall and stops when it hits another wall.
- Roll Off Wall: The ball rolls along a wall and then rolls off the end of the wall.
- Guide: The ball is rolled from one location to another without touching a wall.
- Roll From Wall: The ball hits, or is on, a wall and then is maneuvered off it.

Figure 8 shows how primitives can be performed to traverse part of the maze. These primitives are used in a process similar to the one implemented in the air hockey environment is used.

4.1 Selecting the Appropriate Primitive

Following the same method as in the air hockey environment, a primitive selection module has been created that uses the segmented primitive data to decide which primitive should be performed for a given state of the environment. This module uses the following input-output information:

Inputs:

- The XY ball position
- The ball's velocity components
- The board's angular position

Outputs:

- The primitive used under these conditions
- The ball's position at the end of the primitive
- The ball's velocity components at the end of the primitive

The primitive selection module first decides which primitive to perform. Once the primitive to use has been decided upon, it then goes on to obtain the parameters needed for the performance of that primitive. This primitive will continue to be performed until the ball has reached the end location, or the ball goes outside of a bounding

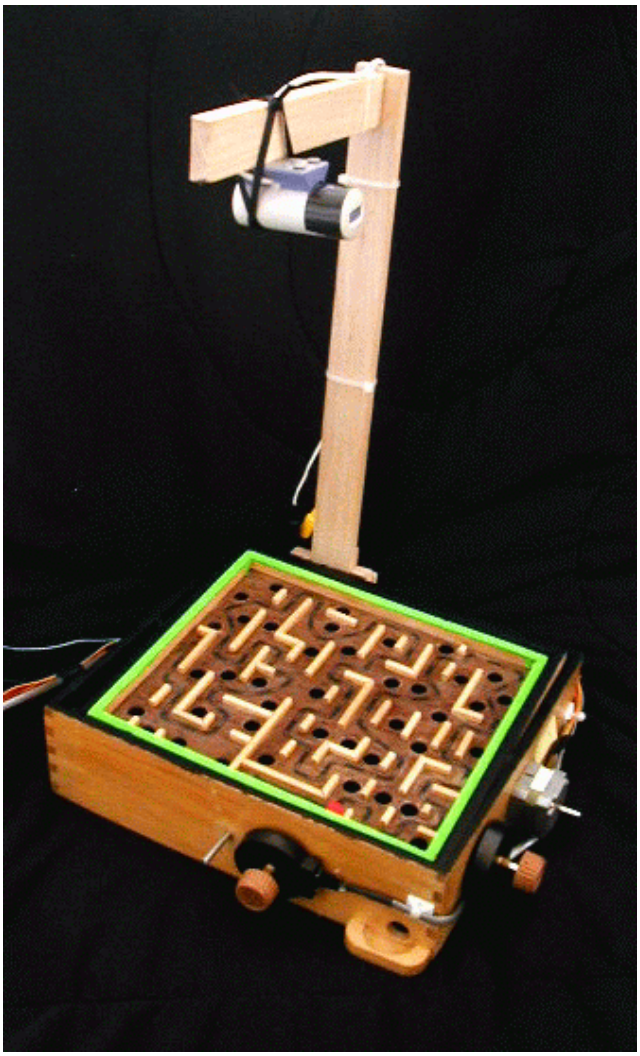

Fig. 6. The physical marble maze implementation.

box containing the start and end location, or too much time has elapsed. The primitive selection module will then be queried for the next primitive to perform.

To select a primitive to use, the agent observes the position and velocity of the ball and position of the board and uses this information as input to the primitive selection module. In our implementation the primitive is chosen from a database using a nearest-neighbor approach. First the nearest data points to the query point are retrieved from the database. These data points represent the observed primitives the human performed when in a state similar to the observed state. If the human has performed different primitives in this area, the returned data points include many types of primitives. From these returned points the module must decide on a single primitive to perform. This is currently performed by choosing the primitive that occurs the most often near the query point.

4.2 Obtaining Primitive Parameters

As in the air hockey environment, after the primitive to use has been chosen, the parameters for that primitive need to be determined. The parameters provide a subgoal, such as the end location and velocity of the ball, for the primitive to be performed. Each data point in the database also contains the parameters that were used when that primitive was performed. Only the data points that are the same as the chosen primitive and were performed near the given location are used to compute the appropriate parameters. The parameters to use for the chosen primitive are computed by locally weighting and combining the parameters of the chosen data points. Closer points receive higher weights then more distant points. The selected primitive along with the computed parameters are sent to the primitive performance module to control the low level actuators.

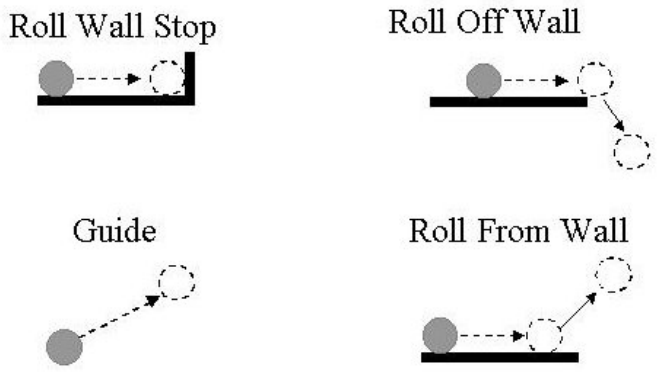

Fig. 7. Primitives used in the marble-maze.

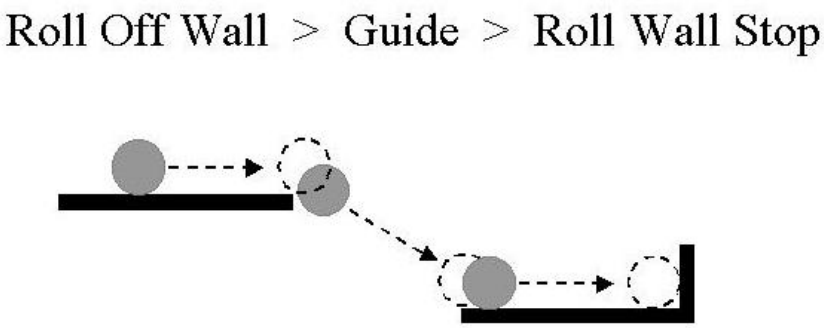

Fig. 8. Primitives being performed in the marble-maze.

4.3 Primitive Performance

Now that the agent knows which primitive to perform and what parameters to use, it needs to know how the board is to be moved in order to perform the primitive. An action generation module is created from the observed data that provides the board movement needed to complete the chosen primitive. The module is queried for the needed action at every time step during the primitive execution.

Primitive selection and parameter generation are performed using the global board state; the agent must know what primitive to perform at the observed position. Primitive performance, on the other hand can be performed using local information. In our implementation it is assumed that the robot has the same motion ability in the X and Y direction. Primitives learned in one part of the board can be performed in other parts of the board. The input to primitive performance can be local information such as the distance to the end of the primitive and the velocity in relationship to the wall be rolled on.

This primitive selection module takes as input the parameters of the primitive that the agent would like to execute and outputs the board movement necessary to successfully perform the primitive and produce the desired output. Since the database was created from captured data the board movements generated are based on the movements the human made in a similar situation. Kernel regression is used to obtain this information from databases created from the observed data.

5 Results

The learning algorithms for virtual air hockey and the virtual marble maze are fully implemented. The virtual air hockey player agent provides a fun and challenging opponent. We are currently devising ways to evaluate the performance of the air hockey player agent in a quanitative way. In the marble maze environment, all the primitives, except the guide primitive, are learned from the observed player. The guide primitive is difficult to perform on real

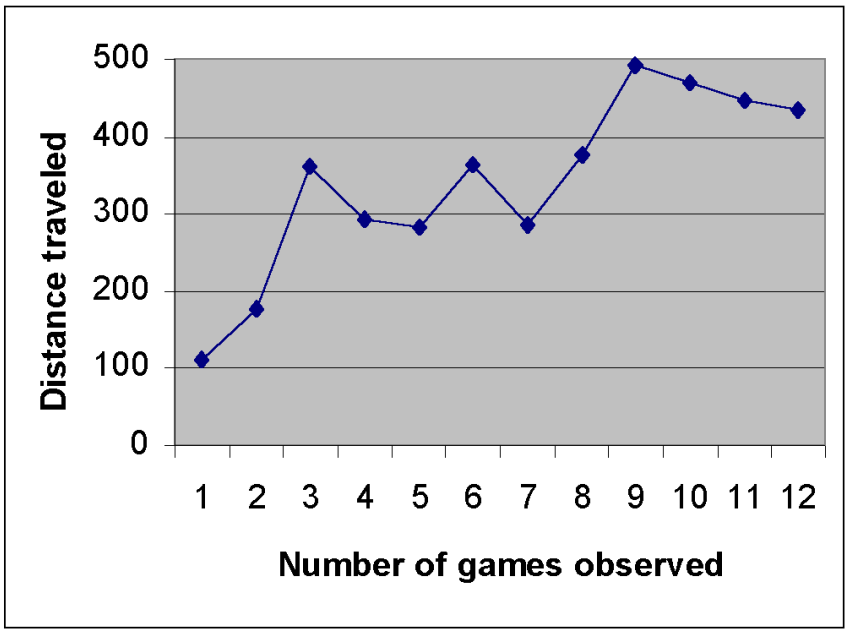

Fig. 9. The performance of the virtual marble maze playing agent as the number of games observed is increased.

hardware with stiction. We are currently looking at ways in which it can be broken up into a combination of smaller primitives, such as a land primitive in which the ball lands on a wall and a u-turn primitive in which the ball moves constantly along one axis but changes direction in the other axis.

Figure 9 shows a graph of the performance of the marble maze. This graph compares the performance of the player agent in comparison to the number of successful games it has observed. Performance is measured by placing the ball in 15 locations and measuring how far the agent traverses the maze from these locations. The distance traveled during the 15 trials are then added together. As can be seen by the graph, as the agent observes more games its performance improves. The total maximum distance the maze can be can be traversed from these locations is 859. As can be seen, the player agent must now learn further using other mechanisms such as learning from practice.

The same learning algorithms are currently being adapted for use by the humanoid robot and the marble maze playing agent in the hardware implementations. The humanoid air hockey environment has been set up and we have demonstrated the humanoid's mobility to hit the puck. In the hardware marble maze environment, the computer successfully controlled the board to traverse a marble through the entire maze. The method used was that of a direct engineering approach. The environment was modeled and the servos were given parameters needed to move the ball from one location to another. A human then input the motor and ball location waypoints. This manual implementation has given us great insight into the primitives that may be needed by the hardware version of the game.

6 Discussion

There is great deal of work still to do. This section discusses some of the issues that will be addressed soon or may be addressed in the future.

6.1 How the Maze Differs From Air Hockey

In air hockey the primitives are indexed using an absolute position, the location on the board. In the marble maze game we have indexed the primitives using a relative position, so as to support generalization. Whether air hockey should use a relative position index, or the marble maze game should use an absolute position index, remains to be explored.

In air hockey, obtaining the hit parameters and then using this information as a second query to find how to make the hit may be less effective than looking up how to make the hit in a single query. However, the hit

parameters such as target location serve as useful subgoals, and the agent can practice obtaining those subgoals independently from learning from observation.

6.2 Function Approximators and Features

One issue is finding a good function approximator for the type and distribution of data we typically observe. The number of points and the kernel function to use in kernel regression will be explored. A number of alternative function approximators will be explored.

Other features may be added to the learning method to increase the performance of the agent as it plays the games. The opponent's paddle position and/or velocity are not being considered when selecting a primitive in the air hockey domain. The opponents movements may be significant in the way the human moves and should be taken into account the when choosing a primitive.

In the two environments explored the state of the game is very simple. In a real game of air hockey the movement of the opponent's body may be significant in determining what moves are made by a player. Discovering what features are relevant or important can be very difficult. For example the head movement may not be important but the eye movement could be very important.

Once the agents are performing at an acceptable level, other ways to learn the task will be explored, such as using reinforcement learning. This will provide a method to compare with learning using primitives. Data can also be captured as the agent attempts to play the game. This data can be parsed in real time for primitives that can be added to the learning modules. Methods to give the air hockey virtual player more human-like movements will also be explored.

6.3 Primitives

The choice of input and output parameters in the creation of the primitive performance modules greatly affects its performance. The hit database, for example, was originally designed to output the needed position of the paddle for a given position of the puck to perform a hit. Using this implementation the virtual player consistently made poor shots. Changing to the use of a relative position of the puck and paddle in terms of an angle greatly improved the hit performance. The parameters for the hit primitives are really subgoals for that part of the task. We can use the notion of a target for a hit primitive to independently train the hit primitive. Other primitives do not need any parameters other than the current state of the game, since their only objective is to imitate what the human did in the next time step for the given state of the game, rather than achieve an external result. Future research will clarify the role of subgoals in learning from observation.

There is more then one way to implement a primitive. As described earlier, the hit primitive was implemented using a physical model, neural networks and kernel regression. The information needed to perform a primitive can be learned from observed data or from its own trials. In these cases a number of different numerical learning methods may be used.

It is important that the definition of each primitive support segmentation, selection, parameter generation, and execution. If the primitive is poorly defined it may be difficult to find in the observed data and may be difficult to segment from other primitives. The primitive should be defined in terms of critical events or certain environment states. Primitives also need to have all the degrees of freedom to perform the task, but not extraneous degrees of freedom. Defining primitives is an iterative process. Once a set of primitives are defined and tried out, they must then be evaluated. Some primitives may be changed or deleted. New primitives may need to be added.

The virtual player will play like the teacher, making the same mistakes as the teacher, and may never discover that there may be a better way to perform for an observed state. When a primitive is found in the training data, the state under which it is performed is recorded. This information allows us to discover what the teacher actually did. It may be that this was not what the teacher had planned to do. If the teacher consistently makes errors, it will appear that the incorrect primitive should be performed.

6.4 Automatically Finding Primitives

In our research, humans using knowledge of the domain select the primitives to be used. The selected primitives are basic and hard lines separate one primitive from another. But in reality this is not often be the case and learning primitives is very difficult for the following reasons.

- Variability - a primitive may not be performed the same way each time.
- Blending/co-articulation - primitives may blend into each other. The line separating one primitive from another may change over time. The way the primitive is performed may also change over time.
- Perceptual confusion - a primitive may be confused for a different one.

7 Conclusions

Virtual and hardware versions of an air hockey game and a marble-maze game have been created that allow data to be captured while the games are being played. Humans, using domain knowledge, select primitives to use and create software needed to parse the captured data. Modules are then created that use this data to select when and where primitives should be performed, the parameters needed for the performance of the primitives, and the low-level movements needed during the actual performance of the primitive. The agent then uses these modules to perform the task. A virtual air hockey player has learned a shot strategy, how to hit, and prepare from observing a human. A virtual marble-maze playing agent has learned to traverse a maze in a similar manner.

8 Acknowledgments

Support for both investigators was provided by the ATR Human Information Processing Research Laboratories and by National Science Foundation Award IIS-9711770. The demonstration maze program that is included with OpenInventor was used as a basis for the virtual marble maze program.

References

1. Chae H. An, Christopher G. Atkeson, and John M. Hollerbach. *Model-Based Control of a Robot Manipulator*. MIT Press, Cambridge, MA, 1988.
2. R. C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1998.
3. C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.
4. C. G. Atkeson and S. Schaal. Learning tasks from a single demostration. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation (ICRA97)*, pages 1706–1712, 1997.
5. C. G. Atkeson and S. Schaal. Robot learning from demonstration. In Jr. D. H. Fisher, editor, *Proceedings of the 1997 International Conference on Machine Learning (ICML97)*, pages 12–20. Morgan Kaufmann, 1997.
6. Paul Bakker and Yasuo Kuniyoshi. Robot see, robot do: An overview of robot imitation. In *AISB96 Workshop on Learning in Robots and Animals*, pages 3–11, 1996.
7. B. E. Bishop and M. W. Spong. Vision-based control of an air hockey robot. *IEEE Control Systems Magazine*, 19(3):23–32, June 1999.
8. R. Dillmann, H. Friedrich, M. Kaiser, and A. Ude. Integration of symbolic and subsymbolic learning to support robot programming by human demonstration. In G. Giralt and G. Hirzinger, editors, *Robotics Research: The Seventh International Symposium*, pages 296—307. Springer, NY, 1996.
9. G. Hayes and J. Demiris. 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*, pages 198–204, 1994.
10. Gerd Hirzinger. Learning and skill acquisition. In G. Giralt and G. Hirzinger, editors, *Robotics Research: The Seventh International Symposium*, pages 277—278. Springer, NY, 1996.
11. Katsuchi Ikeuchi, Jun Miura, Takashi Suehiro, and Santiago Conanto. Designing skills with visual feedback for APO. In G. Giralt and G. Hirzinger, editors, *Robotics Research: The Seventh International Symposium*, pages 308—320. Springer, NY, 1996.
12. Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. In *IEEE Transactions on Robotics and Automation*, pages 799–822, 1994.
13. Labyrinth. Labyrinth game obtained from the Pavilion company., 1999.
14. T. Ohshima, K. Satoh, H. Yamamoto, and H. Tamura. AR2 hockey: A case study of collaborative augmented reality. In *IEEE Virtual Reality Annual International Symposium*, pages 268–275, 1998.
15. R. A. Schmidt. *Motor Learning and Control*. Human Kinetics Publishers, Champaign, IL, 1988.
16. M. W. Spong. Robotic air hockey. http://cyclops.csl.uiuc.edu, 1999.