Using Primitives in Learning From Observation: A Preliminary Report

Darrin C. Bentivegna and **Christopher G. Atkeson**
College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280
ATR Human Information Processing Research Laboratories, 2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan
dbent@cc.gatech.edu, cga@cc.gatech.edu
www.cc.gatech.edu/people/home/dbent, www.cc.gatech.edu/fac/Chris.Atkeson

Abstract

A virtual air-hockey game and a virtual and an actual marble-maze game have been created that allow data to be captured while the games are being played. This data is parsed into small parts of the task called primitives and databases of observed primitives are created. A learning agent then uses the database to perform the task. A virtual air-hockey player has learned to select and make shots. An agent is being created that will learn how to play the marble-maze game in a similar manner.

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 the primitives can be automatically learned by observing a teacher's performance. This paper describes how the parameters of a set of predefined primitives can be learned.

Two virtual environments and a physical implementation will be described that are being used for this research. 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. The virtual air-hockey player learns which hit to select, how to make the selected hit, and how to move the paddle from observing a human opponent. The methods used to extract this information from captured data and how the virtual player uses this information will be described.

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 2 and 3. The marble maze environment is near the beginning of its development. The construction of and the learning methods to be used in this environment will also 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, motion capture and object manipulation is simplified. A camera based motion capture system can easily be used to collect data in a hardware implementation (Bishop & Spong 1999; Ohshima *et al.* 1998). A stationary arm or some other similar robotic device can be programmed to play air-hockey on an actual table (Spong 1999; Bishop & Spong 1999). A marble-maze game (Labyrinth 1999) has been outfitted with stepper motors and a camera so that the movements may be captured and controlled by a computer.

Primitives

Robots typically must generate commands to all their actuators at regular intervals. The analog controllers for our seven degree of freedom arm are given desired torques for each joint at roughly 500Hz. Thus, a task with a one second duration is parameterized with $7 * 500 = 3500$ 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 movements are often not feasible. Special purpose techniques have been developed to deal with this problem, such as trajectory learning (An, Atkeson, & Hollerbach 1988) and learning from observation (Atkeson & Schaal 1997a; 1997b; Hayes & Demiris 1994; Kuniyoshi, Inaba, & Inoue 1994; Bakker & Kuniyoshi 1996; Dillmann *et al.* 1996; Hirzinger 1996; Ikeuchi *et al.* 1996).

It is our hope that primitives can be used to reduce the dimensionality of the learning problem (Arkin 1998; Schmidt 1988). 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 up of many primitives.

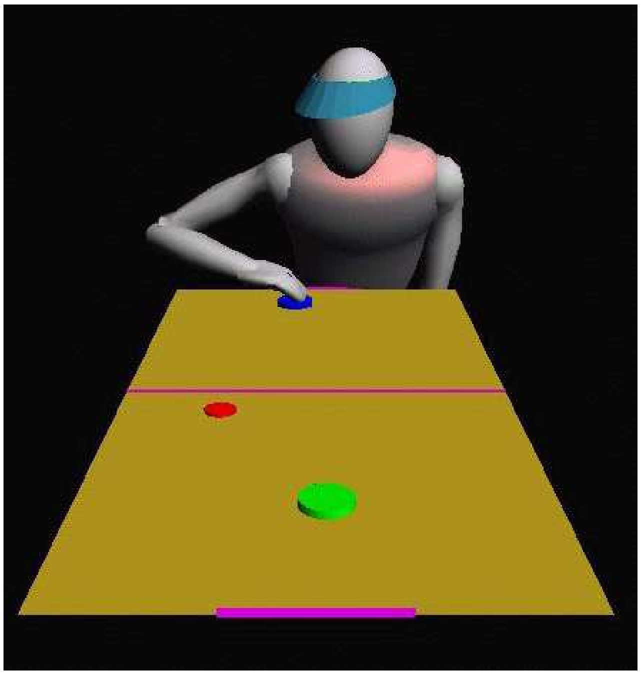

Figure 1: The virtual air hockey environment. The disc shaped object near the center line is a puck which slides on the table and bounces off the sides, the other two disc shaped objects are the paddles. The far paddle is controlled by the virtual player and the closer paddle is controlled by a human player by moving a 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).

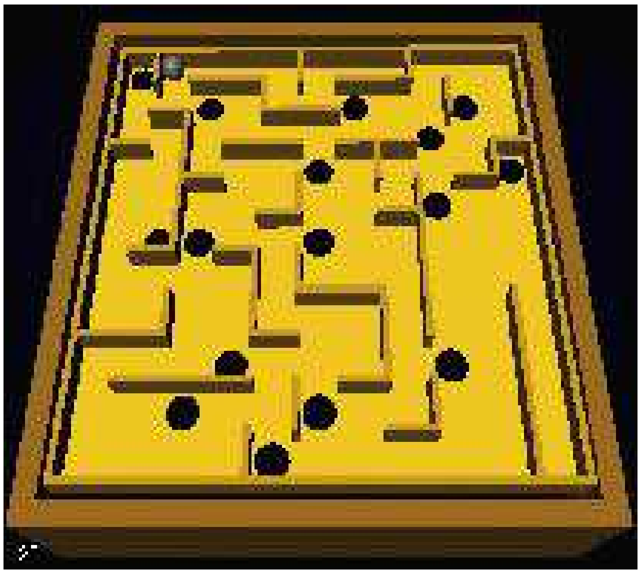

Figure 2: The virtual marble maze environment.

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 research, a human, using domain knowledge, designs the candidate primitives that are to be used. Algorithms are created to segment the observed behavior into primitives and to build a database of these primitives. The agent then uses this database to decide at certain times what primitive to perform and how to perform it.

Virtual 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 surface that create 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 of the table, and the puck bounces off of this edge with little loss of velocity. At each end of the table there is a slot that the puck can fit through. The objective of the game is to hit the puck so that it goes into the opponent's slot while also preventing it from going into your own slot.

Figure 1 shows the virtual air hockey game created that can be played on a computer running OpenInventor and Tcl/TK. The game consists of two paddles, a puck and a board to play on. A human player controls one paddle using a mouse. 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 resolving any redundancies so as to make the virtual player look "human-like". If the location is not within the limits of the board and the reach of the virtual player the location is adjusted to the closest 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.

Human domain knowledge was used to define a set of primitives to work with initially. Three hit primitives

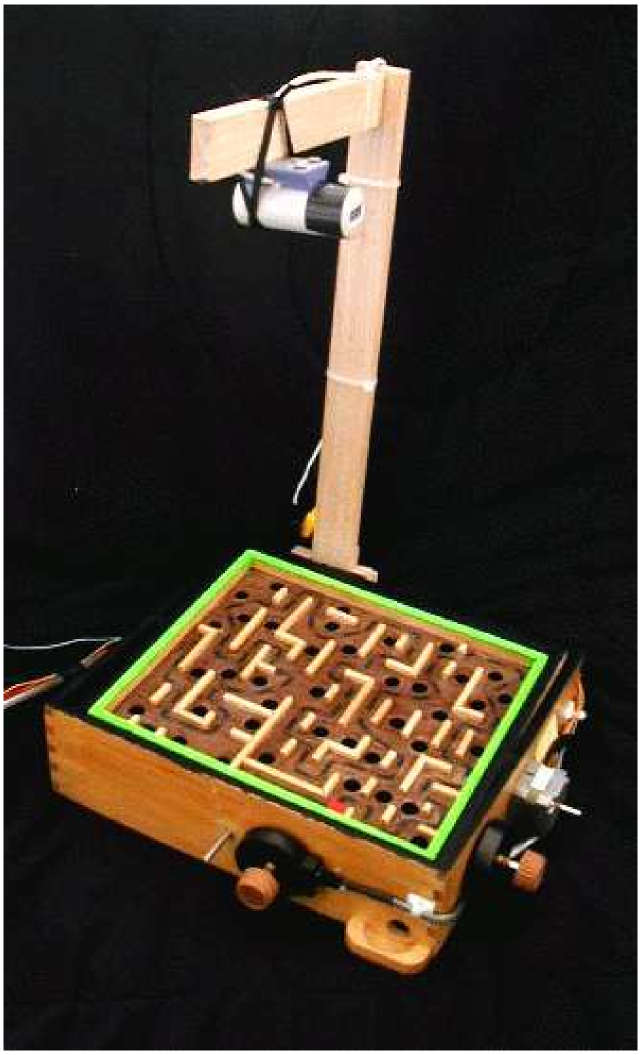

Figure 3: The physical marble maze implementation.

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 while the puck is still on the same side.

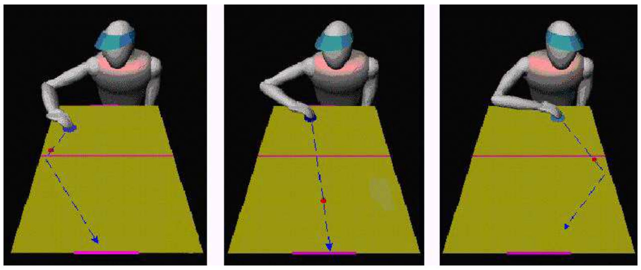

Figure 4: Three hit primitives being performed by the virtual player: right, straight, and left.

Selecting the appropriate primitive

The virtual player must decide which primitive to perform. 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 current board state.

A database was created to guide selection of primitives, incorporating prior observations of primitives being executed. The context or state in which each primitive has been performed is extracted from the 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. 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 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 (where it crossed the center line) and velocity (when it crossed the center line), the hit location, and the outgoing puck velocity and its target. 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 it was not blocked by the opponent. 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 the correct 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.

To determine which primitive to use the virtual player observes the puck's position and velocity when it crosses the centerline. It queries the database using a nearest neighbor technique, looking for the previously executed primitive with puck position and velocity closest to the current position and velocity. It then returns that primitive.

Obtaining hit parameters

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 from the nearest neighbor query to the database as part of the selected primitive as explained in the previous section. A future implementation will obtain the parameters by interpolating between parameters of previously executed primitives of the selected type.

Finding the right paddle motion

The hit parameters encode the motion of the puck before and after the hit. Now the virtual player must figure out how to move the paddle to implement this hit. This can be done in many ways. Three methods have been tried; a physical model, neural networks, and kernel regression (Atkeson, Moore, & Schaal 1997).

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 results of this method. If a physical model 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 paddle the way that the human moved the paddle to make a shot. A database is again created using critical events and 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 database is used in a learning module that tells the virtual player the paddle's velocity components and relative position that is needed to make the desired shot. This database has been approximated using neural networks and also kernel regression. 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.

The prepare 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 virtual player to learn this behavior from the human player a database is created from the observed data. This database contains the ball's and the human puck's position and velocity components during the time when the puck is on the other side of the center line. 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 center line. The learning module returns the desired position and velocity of the paddle. Only the desired position is currently being used. The virtual player moves toward this position with a given velocity that is hard-coded. If that point is not reached by the end of the time cycle, the virtual player will ignore it and move toward the most recent desired point.

Multi-Shot primitive

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. Once the virtual player starts executing this primitive, it will continue until the puck crosses the center line. The database for this primitive contains the velocity and position of the puck when it is on a player's side, 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.

Marble Maze

Primitive learning is also being explored in the marble maze environment in software, figure 2, and on hardware, figure 3. 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. In both versions the time and the board and ball positions are recorded approximately 30 times a second as a human plays the game. Research in this environment has just begun and this section describes the method that will be used for having an agent learn how to play the marble-maze game.

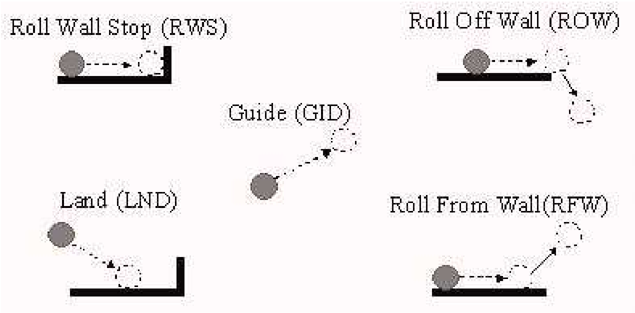

Figure 5: Primitives used in the marble-maze.

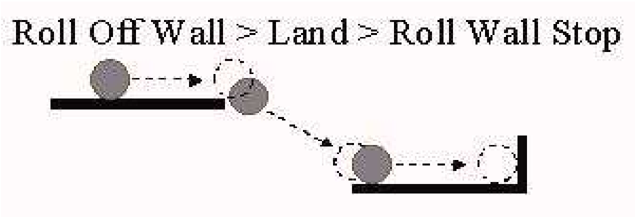

Figure 6: Connected primitives in the marble-maze.

Primitives are extracted from the captured data and a database is created for each primitive. 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 5:

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

Figure 6 shows how three primitives can be chained together.

To learn from the captured data, a database will be used in a three step process similar to the one used in the air hockey environment. A database of primitives will be created that contains all previously executed primitives. This database will contain the state of the game at the beginning and end of execution of a primitive. More specifically, the database will contain at least the following information:

Inputs:

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

Outputs:

- The primitive used under these conditions
- The primitive parameters such as the length the ball travels during primitive execution and/or the ending velocity of the ball

The agent will first query the database to decide which primitive to perform. It will then obtain the parameters needed for the performance of that primitive. Lastly the agent will use the parameters and the database to find out how to perform the selected primitive.

To select a primitive to use, the agent will observe the position and velocity of the ball and position of the board and use this as a query into the database. The primitive will be chosen from the database using a single nearest-neighbor approach. The database will then be queried again, focusing only on the primitives of the selected type performed near the given location, to obtain the appropriate parameters for the primitive to be executed.

Now that the agent knows which primitive to perform and what parameters to use, it needs to know where the board needs to be moved to in order to perform the primitive. The database of the selected primitive will be queried for the needed action at every time step during the primitive execution. This database 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. Since the database was created from captured data the board movements generated are based on the movements the human made in that situation. Kernel regression will be used to interpolate data from the database. The number of points and the kernel function that will be used in the regression to obtain desired results will need to be found by experimentation.

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.

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 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.

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.

Currently only the nearest data-point is used in deciding which primitive to use and the parameters to use with that primitive. As the research progresses methods to combine a number of nearest neighbors to decide what the correct action should be will be investigated.

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.

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 primitive database. Methods to give the air-hockey virtual player more human-like movements will also be explored.

Primitives

The choice of input and output parameters in the creation of the databases 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. On the other hand it may be that the primitive is very simple and can be hard coded.

In the above two environments 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.

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 make errors, it will appear that the incorrect primitive should be performed.

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 true 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.
- Do not have segmented data - the data is not easily segmented into primitives.

Conclusions

A virtual air-hockey game and a virtual and hardware marble-maze game have been created that allow position 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 and create primitive databases. The agent then uses these databases to perform the task. A virtual air-hockey player has learned a shot strategy, how to hit, and prepare from observing a human. An agent is being created that will learn how to play the marble-maze game in a similar manner.

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

An, C. H.; Atkeson, C. G.; and Hollerbach, J. M. 1988. *Model-Based Control of a Robot Manipulator.* Cambridge, MA: MIT Press.

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

Atkeson, C. G., and Schaal, S. 1997a. Learning tasks from a single demostration. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation (ICRA97)*, 1706–1712.

Atkeson, C. G., and Schaal, S. 1997b. Robot learning from demonstration. In D. H. Fisher, J., ed., *Proceedings of the 1997 International Conference on Machine Learning (ICML97)*, 12–20. Morgan Kaufmann.

Atkeson, C. G.; Moore, A. W.; and Schaal, S. 1997. Locally weighted learning. *Artificial Intelligence Review* 11:11–73.

Bakker, P., and Kuniyoshi, Y. 1996. Robot see, robot do: An overview of robot imitation. In *AISB96 Workshop on Learning in Robots and Animals*, 3–11.

Bishop, B. E., and Spong, M. W. 1999. Vision-based control of an air hockey robot. *IEEE Control Systems Magazine* 19(3):23–32.

Dillmann, R.; Friedrich, H.; Kaiser, M.; and Ude, A. 1996. Integration of symbolic and subsymbolic learning to support robot programming by human demonstration. In Giralt, G., and Hirzinger, G., eds., *Robotics Research: The Seventh International Symposium*, 296—307. Springer, NY.

Hayes, G., and Demiris, J. 1994. 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*, 198–204.

Hirzinger, G. 1996. Learning and skill acquisition. In Giralt, G., and Hirzinger, G., eds., *Robotics Research: The Seventh International Symposium*, 277—278. Springer, NY.

Ikeuchi, K.; Miura, J.; Suehiro, T.; and Conanto, S. 1996. Designing skills with visual feedback for APO. In Giralt, G., and Hirzinger, G., eds., *Robotics Research: The Seventh International Symposium*, 308—320. Springer, NY.

Kuniyoshi, Y.; Inaba, M.; and Inoue, H. 1994. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. In *IEEE Transactions on Robotics and Automation*, 799–822.

Labyrinth. 1999. Labyrinth game obtained from the Pavilion company.

Ohshima, T.; Satoh, K.; Yamamoto, H.; and Tamura, H. 1998. AR2 hockey: A case study of collaborative augmented reality. In *IEEE Virtual Reality Annual International Symposium*, 268–275.

Schmidt, R. A. 1988. *Motor Learning and Control.* Champaign, IL: Human Kinetics Publishers.

Spong, M. W. 1999. Robotic air hockey. http://cyclops.csl.uiuc.edu.