Learning How to Behave from Observing Others

Darrin C. Bentivegna[*,**] **Christopher G. Atkeson**[*,***]
[*]ATR, Human Information Science Laboratories Department 3, Kyoto, Japan
[**]College of Computing, Georgia Institute of Technology, Atlanta
[***]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA

Abstract

This paper presents a framework that allows an agent to use observed data to initially learn a predefined set of primitives and the conditions under which they are used. A method is included for the agent to increase its performance while operating in the environment. The details of implementing this framework on agents that play air hockey in simulation and on an actual table will be presented. Issues involved with using observation data and primitives to increase the learning rate of agents are discussed.

1. Introduction

Learning without any prior knowledge in environments that contain large or continuous state spaces is a daunting task. For agents that operate in the real world, learning must occur in a reasonable amount of time. Providing an agent with domain knowledge and the ability to use observed data when learning can greatly decrease the time needed to learn new tasks. A framework was created in which to conduct research that explores the use of primitives in learning from observation. The framework will be introduced and the details of software and hardware agents that use that framework to learn how to play air hockey will be described.

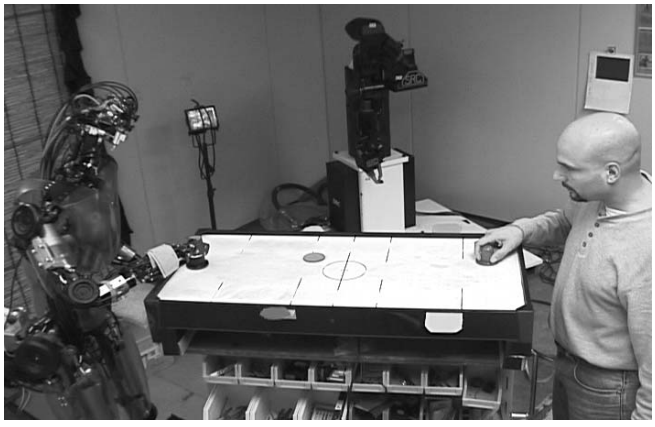

Figure 1: Hardware air hockey environment.

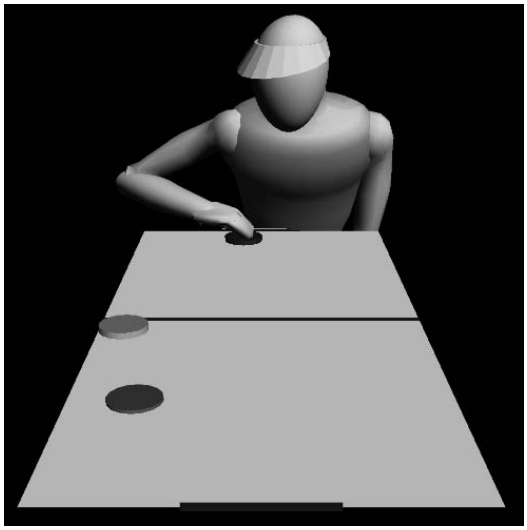

Figure 2: The virtual air hockey environment.

A hardware and software environment of air hockey, figures 1 and 2, has been created. The software version allows a human to play against a cyber player and the hardware version allows the human to play against the humanoid robot DB. In the software game an agent used data collected while observing a human to initially learn how to perform different types of air hockey primitives and then went on to increase its performance of shot primitives through practice. In both the hardware and software versions the agents have used the observed data to learn how to choose a primitive and primitive parameters when operating in the environment. This research is also being performed in a marble maze environment (Bentivegna and Atkeson, 2000), figure 3, but this paper will primarily focus on air hockey.

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.

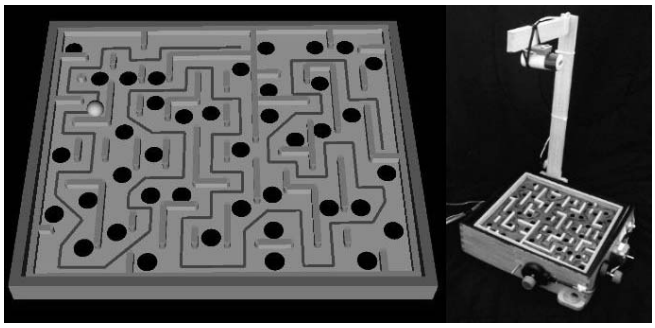

Figure 3: The software marble maze game on the left modeled after the hardware version on the right.

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 (An et al., 1988), learning from observation (Atkeson and Schaal, 1997, Hayes and Demiris, 1994, Kuniyoshi et al., 1994, Bakker and Kuniyoshi, 1996, Dillmann et al., 1996, Hirzinger, 1996, Ikeuchi et al., 1996), postural primitives (Williamson, 1996), and other techniques that decompose complex tasks or movements into smaller parts (Arkin, 1998, Bentivegna and Atkeson, 2000, Mataric et al., 1998). 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. 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 task expert predefines the set of primitives to be used for a given environment and algorithms are created to find the primitives in the captured data.

2. Issues with Using Primitives

This section outlines some of the issues involved with using primitives and the research referenced in this section provide examples of the methods by which the issue is being handled.

2.1 Defining and Learning a Set of Primitives

A task that is to be performed using primitives must first be decomposed into a set of primitives that include all the actions needed to perform the task. This issue can be dealt with by having a human task expert define the set of primitives (Ryan and Reid, 2000) or have the robot discover primitives automatically after observing a performance of the task (Fod et al., 2000) or operating in the task environment (McGovern and Barto, 2001). Once a set of primitives is defined, the robot must have a way to learn how to perform them. Some research deals with this issue by explicitly programming the primitive performance policy into the agent (Brooks, 1986) or having the robot learn the policy using learning techniques (Aboaf et al., 1989). Given a set of primitives and a task, the robot must decide on which primitive to perform at any given time. This has been accomplished by a human specifying the sequence of primitive types to be performed (Mataric et al., 1998), using a planning system (Tung and Kak, 1995), or having the robot learn the sequence from observed data (Kuniyoshi et al., 1994).

At first thought, 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 (Wooten and Hodgins, 2000). These parameters can have continuous values and therefore can be difficult to select or learn through trial and error (Likhachev and Arkin, 2001). 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 (Dietterich, 1998).

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 (Balch, 1997), the parameters to use with the chosen primitive (Pearce et al., 1992), and the primitive execution policy (Lin, 1993). The research presented shows many methods that can be used to learn each of these items and an issue is how to choose a method that works best to learn this information for a given environment.

2.2 Issues with Using Observed Data

When using observed data to learn a task other issues are introduced. From all the information that the robot is presented with, it must decide on what is relevant for learning the task (Kaiser and Dillmann, 1996). The observed data must also be segmented into primitives if it is to learn such things as primitive sequence performance, needed parameters, or primitive execution policy from the observed data. Segmenting and learning relevant features to observe can be accomplished in many ways. Explicitly providing only the presegmented information (Delson and West, 1996), specifying conditions that represent segmentation points (Mori et al., 2001, Kang and Ikeuchi, 1993), and using hidden Markov models (Hovland et al., 1996) are some of the methods that have been explored.

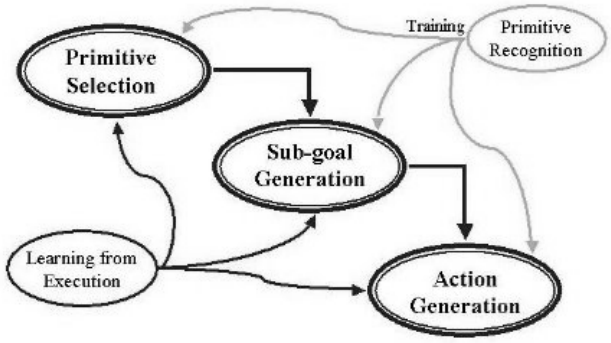

Figure 4: Framework for learning from observation using primitives.

3. Strategy for Primitive Use

Figure 4 shows our framework designed for conducting research in learning from observation using primitives. Currently, a human, using domain knowledge, designs the candidate primitives that are to be used. The primitive recognition module segments the observed behavior into the chosen primitives. This segmented data is then used to provide the training data for the primitive selection, sub-goal generation, and action generation modules.

The primitive selection module provides the agent with the primitive type to perform for the observed state of the environment. The desired outcome, or goal, of performing that primitive type is specified by the sub-goal generation module. Lastly the actuators must be moved to obtain the desired outcome. The action generation module finds the actuator commands needed to execute the chosen primitive type with the current goal. The learning from execution module provides information to the agent that can be used to improve its performance while operating in the environment. The next section provides an example of using this framework to have agents learn how to behave in an air hockey game environment.

4. Learning Air Hockey from Observation Using Primitives

The air hockey game consists of two paddles, a puck and a board to play on. In the hardware version, figure 1, the human plays against the humanoid robot DB. There are no playing restrictions placed on the human players and they can play just as if they were playing against another human. Since the observation data is obtained using DB's eyes, the human player is also not required to don special equipment. In the software version, figure 2, a human player controls one paddle using a mouse and at the other end is a simulated or virtual player. In the simulator 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. In both versions the position of the two paddles and the puck are recorded at 60Hz.

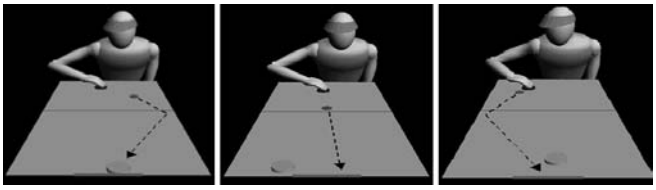

Figure 5: Three hit primitives being performed by the virtual player: left, straight, and right.

4.1 Primitives Being Explored in Air Hockey

As explained above, a human domain expert defines a set of primitives to work with. Three hit primitives are shown in figure 5. 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.
- Block: the player deliberately does not hit the puck but instead moves into a blocking position to prevent the puck from entering their goal.
- Prepare: movements made while the puck is on the opposite side from the player. The player may be preparing to setup for a shot, or preparing to defend their goal.
- Multi-Shot: movements made after a shot is attempted, but while the puck is still on the player's side. If the puck is not quickly moving toward the opponent's side, the player will have the opportunity to hit it again.

If it is determined that the puck did not travel from the other side prior to a shot being made, this shot will be classified as a multi-shot primitive. The multi-shot primitive may be performed after a failed shot attempt or a blocking primitive and the puck is moving very slowly within hitting range. In this situation the player has a lot of time to setup and make a shot. If, after a collision

with the observed player's paddle has been detected, the puck's observed trajectory does not fit the requirements for one of the hit primitives, the blocking primitive is considered. If the player's paddle is near their goal at the time of the collision, this will be classified as a blocking primitive.

The prepare primitive is performed whenever the puck is on the side opposite the player and will continue until a shot is to be attempted or a block is to be performed.

4.2 Perceiving the Primitives

The observed data must first be segmented into the above primitives. To accomplish this, critical events are used. Critical events are easily observable occurrences such as the puck mostly traveling in a straight line with a gradually decreasing velocity and puck collisions, in which the ball speed and direction rapidly change. A small portion of data that has been collected while a human played against the humanoid robot is shown in figure 6. From this data it can be seen that the puck-paddle hit locations occur when the puck and paddle are within hitting range and there is a significant change in the puck's velocity.

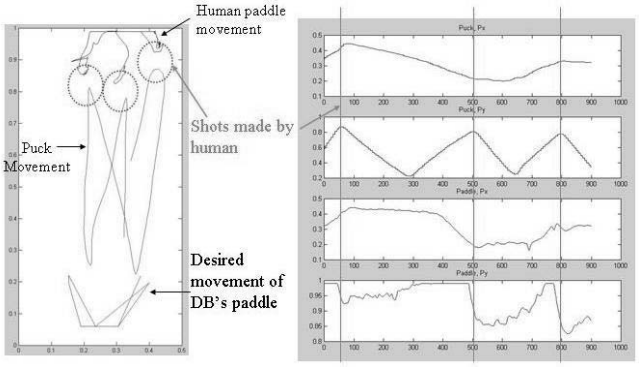

Figure 6: A small segment of data captured while a human operates in the hardware air hockey environment. The graph on the left shows 2D, x and y, traces of the paddle and puck. On the right this same data is shown plotted against time.

To find a hit primitive in the captured data, for example, a collision of the puck with the player's paddle is searched for. When this event has been found, the puck's velocity vector is then observed to determine the target of the hit, and the state of the environment under which this primitive was selected.

4.3 Selecting the Appropriate Primitive

As discussed in the strategy above, it is the responsibility of the primitive selection module to choose the type of primitive, based on the current state and prior observations of primitives being executed. In our implementation, the context or state in which the human has performed each primitive is extracted from the observed data, and is used by a weighted nearest neighbor lookup process to find the past primitive executions whose context is most similar to the current context. The puck's position and velocity when it crosses a predefined line is used as the index for a lookup. Since the player must decide on a shot well before the puck is hit, a predefined line located just past the centerline from the player was chosen as the point where the agent will make its decision.

From the segmented data a database is created that contains the primitive type that was used, the state of the environment when that type was used, and the state of the environment that results from performing that primitive. A data point in the air hockey hit database contains; the primitive type used (one of the primitive types described above), the position and velocity components of the puck when it crosses the predefined line, the location of the puck when it was hit, the absolute velocity of the puck after it is hit, and the location on the back of the wall that the puck would hit if it is not blocked by the opponent.

A simple idealized model is used to determine where the puck would hit the back wall if the opponent did not block it. This use of a model enables the learning agent 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 being blocked by the opponent.

A lookup can now be performed on this database to find the data points that are closest to the query point. To find the closest points in the state space the distance of each data point from the query point is computed as follows: $D(\boldsymbol{x}, \boldsymbol{q}) = w_j \sqrt{\sum_j (\boldsymbol{x}_j - \boldsymbol{q}_j)^2}$ where $\boldsymbol{x}$ and $\boldsymbol{q}$ are the locations of the data point and the query point in state space and w is a vector that allows each dimension to be weighted differently . j represents the jth component of the vector. The closest five data points of the same primitive type determine which primitive type is chosen.

4.4 Computing the Primitive Parameters

The sub-goals, or outcome of performing that primitive, provide the parameters needed to perform the action. The parameters needed to perform the hit primitive are the desired hit location, the angle between the puck and paddle when it is hit, the puck's desired post-hit velocity, and the target location. This information is obtained using a locally weighted regression (LWR) model(Atkeson et al., 1997). The selected data points provide the information to compute the parameters. A kernel function, $K(D) = \exp^{-\alpha D^2}$, uses

the distance to compute the weight of each data point. The output components at the query point use n data points and are computed using the following equation: $y(\boldsymbol{q}) = \frac{\sum y_i K(D(\boldsymbol{x}_i, \boldsymbol{q}))}{\sum K(D(\boldsymbol{x}_i, \boldsymbol{q}))}$ where i ranges from 1 to n. The values of the vector w and α were set globally and were chosen by trial and error. Future research will explore methods to select these values locally and also explore the use of the locally weighted learning methods of Schaal, et al. (Schaal et al., 2000).

4.5 Finding the Right Paddle Motion

Currently the agents in air hockey are programed with the ability to move the paddle from an initial location to a given location with a given velocity. To properly behave in the air hockey environment the information the agents need is where to move the paddle and when. The parameters that were obtained above provide the information for the appropriate action generation module to compute the needed paddle movements. In the blocking primitive, for example, the paddle only needs to be moved to the blocking location and will then wait there until the puck is moving away from the agent or stops. The hit primitives are more complicated and the agent needs to know where the paddle should be and what its velocity should be when the puck is hit. Three methods have been tried to provide this information; an idealized model based on physics, neural networks, and LWR.

The model based on idealized physics 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 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 paddle the way that the human moved the paddle to make a shot. Another data base is created from the observed data for the hit primitives 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 (the angle between the puck and the paddle).

The needed paddle velocity components are returned from a query to the database above. The needed paddle position is computed using the puck's desired hit location, which is passed as a sub-goal to the action generation module, and the relative paddle information, returned from a query to the database. A back propagation neural network that contains six inputs, three outputs, and a hidden layer with 20 nodes was trained using the database. After being trained the network provides the needed information very rapidly. The LWR model is the same as that described in sections 4.3 and 4.4. As can be seen, at each query of the LWR model every data point in the database is considered. For very large databases or slow computers this can be a problem.

5. Learning Using Only the Observation

Up to this point the agent's only high-level goal is to perform like the teacher. Its only knowledge of the goal of the entire task is in the implicit encoding in the primitives performed. The agent performs an action based upon the observed information. If the results of that performed action are undesirable, a method to store this information should exist. In air hockey, for example, the agent may choose a right bank shot for a given observed environment state. If the outcome of this action is that the agent misses the puck and a score is made against the agent, the agent should not attempt that same action, under the same conditions, in the future. The learning from practice module contains the information needed to evaluate the performance of each of the modules toward obtaining the high-level task objective. This information can then be used to update the modules and improve performance beyond the teacher.

6. Increasing Performance beyond the Observation

There are a number of things that these agents can learn to increase their performance while operating in the environment. They can improve the policy to become more proficient at primitive performance. The virtual air hockey player observed its own performance and collected data while playing and practicing. To practice, pucks are shot toward the agent and the agent attempts

to make a shot. This data was then used to add information to the database that is used with the neural network and LWR model to improve the hit performance of the agent. Many other methods can be used by an agent to learn a primitive through practice such as those used by Schaal and Atkeson (Atkeson and Schaal, 1997) in pole balancing and of Kamon, et al. (Kamon et al., 1998) in learning to grasp objects.

The agent can also learn to select more appropriate primitives and primitive parameters. An algorithm that provides this ability is currently being tested and will be presented in the future.

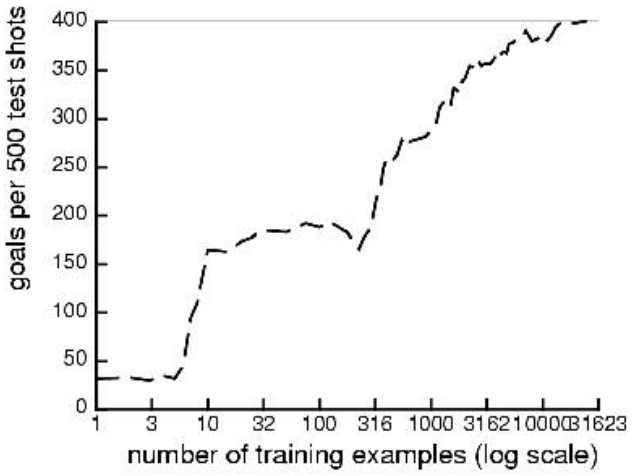

Figure 7: The number of goals obtained by the virtual agent as an increasing number of shots are observed. Note the log scale on the horizontal axis.

7. Results

Both the software and hardware air hockey agents have learned a shot taking behavior from observing a human player. The agents decide on what type of shot will be attempted, the position the puck will be hit at, and the pucks desired velocity after it is hit from the observed data. These agents provide a fun and challenging opponent. Figure 7 shows a graph of the performance of the hit primitives as the software agent observes an increasing number of hits. The agent first observes a number of shots taken by the human. The puck is then shot toward the agent 500 times just as if the opponent hit it. The number of times the agent scores for each observation set is shown in the graph. During these trials the opponent's goal is left undefended. As the number of observed hits increases, the agent's performance increases. We are currently devising other ways to evaluate the performance of the air hockey player agent in a quantitative way such as having it play against another software agent with a fixed playing strategy.

The humanoid's vision system is being improved to allow the robot to sense the objects more accurately and to also observe its own paddle. Currently the board is placed in a fixed location and the robot only assumes its paddle is at the position on the board it was commanded to. Having the ability to see its own paddle in relationship to the puck will provide the ability to incorporate a learning method without having to be as concerned about small movements in the board's position.

8. Discussion

While a framework offers structure and can help to modularized 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 figure 4. 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.

8.1 Combining Primitive Type and Parameter Selection

The presented framework first selects the primitive type to perform for the observed environment state. The selected primitive type then narrows down the search for the parameters needed. 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 policy that needs to be performed to obtain that subgoal from the agent'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 good if a primitive policy 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. 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 (Faloutsos et al., 2001). It is our belief that by first committing to a primitive 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.

8.2 *Combining Parameter Selection 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 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. The primitive policy is designed to operate under constraints of a local environment. This means that this same policy may be used in different parts of the environment 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. The policy may not encode information about the environment that is necessary to select the parameters. This allows the policy to be very specialized, but it must depend on a process with higher knowledge to specify the arguments needed to ensure continued success in the task. As can be seen, this also allows the subgoal to be generated using any method and any information needed. By separating these modules the subgoals can be generated using a course 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 (Morimoto and Doya, 1998).

9. Acknowledgments

Support for both investigators was provided by ATR, Human Information Science Laboratories Department 3 and by National Science Foundation Award IIS-9711770. This research was supported in part by the Communications Research Laboratory (CRL).

References

Aboaf, E., Drucker, S., and Atkeson, C. (1989). Task-level robot learning: Juggling a tennis ball more accurately. In *IEEE International Conference on Robotics and Automation*, pages 1290–1295, Scottsdale, AZ.

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

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

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

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

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

Balch, T. (1997). Clay: Integrating motor schemas and reinforcement learning. Technical Report GIT-CC-97-11, College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

Bentivegna, D. C. and Atkeson, C. G. (2000). Using primitives in learning from observation. In *First IEEE-RAS International Conference on Humanoid Robotics (Humanoids-2000)*.

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

Delson, N. and West, H. (1996). Robot programming by human demonstration: adaptation and inconsistency in constrained motion. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 30 –36.

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In *Proc. 15th International Conf. on Machine Learning*, pages 118–126. Morgan Kaufmann, San Francisco, CA.

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*, pages 296–307. Springer, NY.

Faloutsos, P., van de Panne, M., and Terzopoulos, D. (2001). Composable controllers for physics-based character animation. In *Proceedings of SIGGRAPH 2001*, pages 251–260, Los Angeles, CA, USA.

Fod, A., Mataric, M., and Jenkins, O. (2000). Automated derivation of primitives for movement classification. In *First IEEE-RAS International Conference on Humanoid Robotics (Humanoids-2000)*, MIT, Cambridge, MA.

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*, pages 198–204.

Hirzinger, G. (1996). Learning and skill acquisition. In Giralt, G. and Hirzinger, G., (Eds.), *Robotics Research: The Seventh International Symposium*, pages 277–278. Springer, NY.

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

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*, pages 308–320. Springer, NY.

Kaiser, M. and Dillmann, R. (1996). Building elementary robot skills form human demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2700–2705.

Kamon, I., Flash, T., and Edelman, S. (1998). Learning visually guided grasping: A test case in sensorimotor learning. In *IEEE Transactions on System, Man and Cybernetics*, volume 28(3), pages 266–276.

Kang, S. B. and Ikeuchi, K. (1993). Toward automatic robot instruction from perception: Recognizing a grasp from observation. *IEEE International Journal of Robotics and Automation*, 9(4):432–443.

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*, pages 799–822.

Likhachev, M. and Arkin, R. C. (2001). Spatio-temporal case-based reasoning for behavioral selection. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 1627–1634, Seoul, Korea.

Lin, L.-J. (1993). Hierarchical learning of robot skills by reinforcement. In *Proceedings of the 1993 International Joint Conference on Neural Networks*, pages 181–186.

Mataric, M. J., Williamson, M., Demiris, J., and Mohan, A. (1998). Behavior-based primitives for articulated control. In *Fifth International Conference on Simulation of Adaptive Behavior (SAB-98)*, pages 165–170. MIT Press.

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the 18th International Conference on Machine Learning*.

Mori, T., Tsujioka, K., and Sato, T. (2001). 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.

Morimoto, J. and Doya, K. (1998). Hierarchical reinforcement learning of low-dimensional subgoals and high-dimensional trajectories. In *Proceedings of the 5th International Conference on Neural Information Processing*, volume 2, pages 850–853.

Pearce, M., Arkin, R. C., and Ram, A. (1992). The learning of reactive control parameters through genetic algorithms. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 130–137, Raleigh, NC, USA.

Ryan, M. and Reid, M. (2000). Learning to fly: An application of hierarchical reinforcement learning. In *Proc. 17th International Conf. on Machine Learning*, pages 807–814. Morgan Kaufmann, San Francisco, CA.

Schaal, S., Atkeson, C., and Vijayakumar, S. (2000). Real-time robot learning with locally weighted statistical learning.

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

Tung, C. and Kak, A. (1995). Automatic learning of assembly tasks using a dataglove system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'95)*, volume 1.

Williamson, M. (1996). Postural primitives: Interactive behavior for a humanoid robot arm. In *Fourth International Conference on Simulation of Adaptive Behavior*, pages 124–131, Cape Cod, MA. MIT Press.

Wooten, W. L. and Hodgins, J. K. (2000). Simulating leaping, tumbling, landing and balancing humans. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 656–662.