A Framework for Learning From Observation using Primitives

Darrin C. Bentivegna*1*2, Christopher G. Atkeson*1*3 *and* Gordon Cheng*1
*1Department of Humanoid Robotics and Computational Neuroscience, ATR Computational Neuroscience Laboratories
*3Robotics Institute, Carnegie Mellon University *2College of Computing, Georgia Institute of Technology

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. It is essential for an agent to have domain knowledge if it to learn in the time scales needed [17]. Providing an agent with domain knowledge and also with the ability to use observation data for learning can greatly increase its learning rate. This paper describes a framework, Figure 1, in which to conduct research that explores the use of primitives in learning from observation and practice.

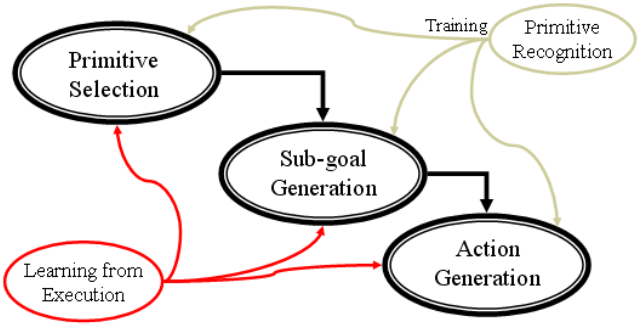

Fig. 1 Framework for learning from observation using primitives.

Virtual and hardware environments of air hockey and a marble maze game, figures 2 and 3, have been created as platforms in which to conduct this research. The virtual environments were first created and provided an invaluable tool in which to test algorithms that will be run on the hardware versions. For this reason the physics of the virtual environments are programmed to match those of the hardware versions as much as possible. In all these environments the position data can be collected as a human operates in the environment. This paper will present agents that have been created to operate in the air hockey environment to demonstrate how the framework can be implemented.

1.1 Primitives

Robots typically must generate commands to all their actuators at regular intervals. The analog controllers for our 30-degree of freedom humanoid robot are given desired torques for each joint at 420Hz. Thus, a task with a one second duration is parameterized with $30 * 420 = 12600$ parameters. Learning in this high dimensional space can be quite slow or can fail totally. Random search in such a space is 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], learning from observation [4] [5] [11] [13] [14] [16], postural primitives [25], and other techniques that decompose complex tasks or movements into smaller parts [2] [6] [18]. It is our hope that primitives can be used to reduce the dimensionality of the learning problem [2] [22].

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. 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.

869F<uIU
%-!<%o!%I!'learning from observation, locally weighted learning
*1Kyoto, Japan
*35000 Forbes Ave., Pittsburgh, PA, 15213, USA
*2Atlanta, GA

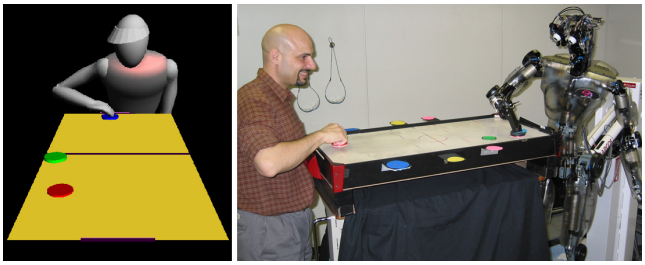

Fig. 2 Hardware air hockey environment.

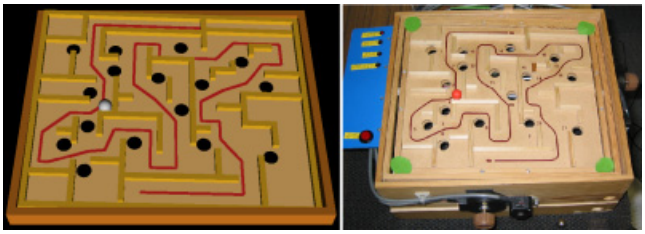

Fig. 3 Marble maze environment.

2. Strategy for Primitive Use

Figure 1 shows our framework designed for conducting research in learning from observation and practice using primitives. The primitive recognition module segments the observed behavior into the defined 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 the air hockey environment.

3. Learning Air Hockey from Observation

In the hardware version of air hockey [7], Figure 2, the human plays against the humanoid robot DB. The humanoid robot has 30 degrees of freedom and is 190cm tall and weighs 85kg. It is hydraulically actuated and attached to a stable pedestal at the hips. The robot is placed at one end of the table and plays the game using one arm. It views the position of the objects using cameras that are on pan-tilt mechanisms on the humanoid's head. The robot uses a simple interpolation scheme to locate the puck and paddles with respect to additional visual markers in the plane of the playing area.

An interpolation scheme is also used to solve the redundant inverse kinematics problem to position the paddle on the playing surface. More information on the vision system and paddle positioning method can be found in [10].

In the software version a human player controls one paddle using a mouse and at the other end is a simulated player. 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.

The manually defined library of primitives that we are exploring for this task is:

- **Straight Shot**: A player hits the puck and it goes toward the opponent's goal without hitting a wall.
- **Bank Shot**: A player hits the puck and the puck hits a wall and then goes toward the opponent's goal.
- **Defend Goal**: A player moves to a position to prevent the puck from entering their goal area.
- **Slow Puck**: A player hits a slow moving puck that is within their reach.
- **Idle** : A player rests their paddle while the puck is on the opponent's side.

3.1 Retrieving Information from the Observed Data

This section describes what, and how, information is obtained from the observed data. As the robot observes a task it looks for critical events. Critical events are large changes or discontinuities in what is happening that can easily be seen. A puck hitting a wall or a paddle is an example of a critical event. Figure 4 shows data collected from observing the hardware air hockey environment. From this figure it can be seen that collisions cause an obvious change in the puck's movement trajectory.

To learn a shot behavior from observing the task the robot follows these steps:

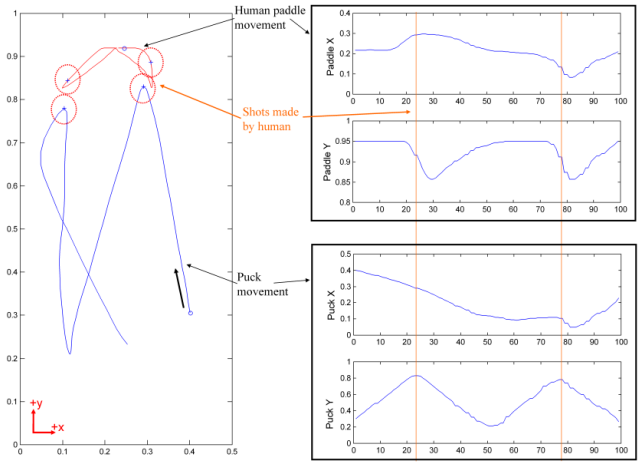

Fig. 4 Data collected while observing a human making shots in the air hockey environment. The left figure shows the data plotted in two dimensions. The right figure shows the data plotted against time. Collisions with the puck can be easily seen in this figure.

! J1 ! K Look for when the puck is close enough to a paddle that it could be in contact with it. If the puck undergoes a discontinuous change in velocity, a hit is assumed at this location. If the puck has a large velocity toward the opponent's side, the position and velocity of the puck and paddle are recorded as the hit state.

! J2 ! K Observe the puck's movements until it gets near the opponent's goal area while looking for collisions with the side walls. If a collision with a wall is observed, which side, left or right, is recorded. We are currently only considering shots with a single wall bounce.

! J3 ! K Look for a collision of the puck with the opponent's wall or goal. If the puck reaches the opponent's wall, the location is recorded as the target position.

! J4 ! K Look for a collision with the opponent's paddle. If the puck is hit by the opponent before it reaches the opponent's wall, the location that the puck would go to if it was not blocked is predicted using a simple learned model and recorded as the target position.

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

- The position of the puck when it is hit.
- The velocity of the puck just before it is hit.
- The angle between the puck and the paddle when the puck is hit.
- The velocity of the paddle when it hits the puck.
- The puck's velocity just after it is hit.
- The position on the back wall that the puck will go to if it is not blocked by the opponent.

The robot first has the opportunity to observe shots taken by the human player and can then observe its own shots while operating in the environment. Successful shots only occur at about 10 to 15 times per minute. Because of this, unless the game is watched or practiced for a long period of time, there is little information in which to learn from. We have structured our learning system to take advantage of the observed information as much as possible.

3.2 Selecting the Appropriate Primitive

The primitive selection module chooses 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 center-line from the player was chosen as the point where the agent will make its decision.

A database is created from the segmented observation data. 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 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: $d(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_j w_j \cdot (\mathbf{x}_j - \mathbf{q}_j)^2}$, where $\mathbf{x}$ and $\mathbf{q}$ are the locations of the data point and the query point in state space, w allows each dimension to be weighted differently, and j represents the jth component of the vector. The closest data point determines which prim-

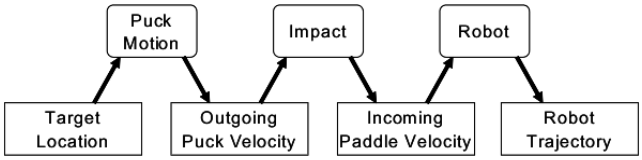

Fig. 5 The transformations involved in action generation.

itive type is chosen.

3.3 Computing the Desired Sub-goal

The desired sub-goal is computed using a locally weighted regression (LWR) model [3]. The sub-goal information needed to perform a hit primitive includes the desired hit location, the puck's desired post-hit velocity, and the target location. 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. We are currently exploring methods to select these values locally [9].

3.4 Shot Action Generation

A shot is made by moving the paddle so that it makes contact with the puck. The puck's movement will be influenced at this point of contact [20] and can therefore be controlled by the paddle. A robot trajectory leads to the paddle hitting the puck, and subsequent puck motion that causes the puck to go to the target location. An action generation module must invert this process, and find a robot trajectory that causes the puck to hit the target location (Figure 5).

We use a locally weighted learning technique, Locally Weighted Projection Regression (LWPR) [24], to represent the learned *Puck Motion, Impact* and *Robot* models [8]. The LWPR approach was chosen because new data can be added easily and the new information is available for use immediately without having to go off line to train the model on the new information. The problem with most locally weighted learning methods is that each data point added to the model increases the time needed to compute a solution [3]. LWPR maintains a reasonably stable lookup time so data may continuously be added. It is a nonparametric local learning system that uses locally linear models, spanned by a small number of univariate regressions in selected directions in the input space. LWPR is proving its usefulness in such tasks as inverse-dynamics learning [21] and inverse kinematics learning [23].

4. Learning Beyond 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. Within our framework, the learning from practice module contains the information needed to evaluate the performance of each of the modules toward obtaining the high-level task objectives. This information can be used to update the modules and improve the robot's performance beyond that of the teacher.

There are a number of things that these agents can learn to increase their performance while operating in the environment. The agents perform actions based upon the observed information. If the results of the performed actions are undesirable, a method to store this information should exist. The agents can also practice to become more proficient at performing each type of primitive. Examples of robots learning primitives through practice are shown by Schaal and Atkeson [4] in learning to balance a pole and by Kamon, et al. [15] in learning to grasp objects.

5. 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 puck's desired velocity after it is hit from the observed data.

Figure 6 shows the result of the software agent while making 500 straight shots in the simulator. For the first 200 shots the agent is using models created from observing the human's shots. Figure 6 plots the average absolute error in hitting the target location; the distance between the desired target location and the location where the puck actually hit the back wall. The dotted line at the bottom of the graph shows the results of the agent performing the action using an exact model of the simulator. The error in the exact model is due to the noise introduced into the simulator and this is effectively the best the agent can perform.

After making 200 straight shots using the models learned from observing the human, the agent then ob-

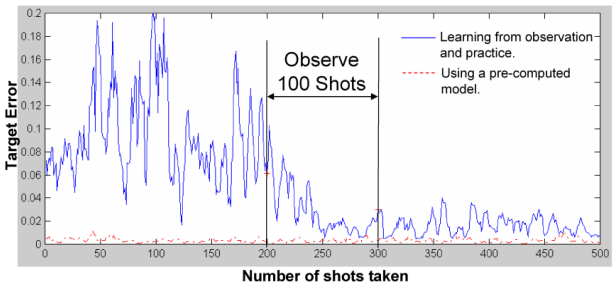

Fig. 6 The absolute error in reaching the target location during 500 straight shots made in the software air hockey environment. The solid line shows 200 shots using the LWPR model trained from observing 44 straight shots performed by the human. The agent then observes 100 of its own shots while practicing and adds that information to the LWPR model. The dotted line is the result of an agent making straight shots using an exact model of the environment. The graph shows the running average of 5 shots.

served 100 of its own shots while practicing (shots 201 to 300 in Figure 6). Whenever the agent observes its own shot it calculates the parameters in the same way as if it were observing a human. This information is then immediately given to the models. Figure 6 shows the result of using these newly trained models for the shots from 301 to 500.

The humanoid robot in the hardware air hockey environment has learned to more accurately obtain the desired paddle hit state while practicing. It has also learned to increase it performance at placing the paddle at the hit position at the correct time [8].

6. Discussion

This section discusses some of the design decisions that went into creating the framework presented in figure 1. 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.

6.1 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 sub-goal generation module provide the next sub-goal? To use that sub-goal information there must be a higher level process that can select a primitive performance policy that will be used to obtain that sub-goal 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 it is [12]. It is our belief that by first committing to a primitive we are simplifying the learning problem by only having to learn the sub-goal appropriate to that primitive type. By separating these modules, the method used to obtain the needed information can be unique for each decision providing extra flexibility.

6.2 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 sub-goal thereby eliminating the sub-goal generation module. By doing this there will be a loss in generality and flexibility. The primitive execution module contains the policy to bring the system from the current state to a new state within the constraints of the primitive type. The primitive execution policy maps the sensor readings to an action for each time step. This policy is designed to operate under constraints of a local environment.

This division of labor also allows the sub-goal to be generated using any method and any information needed. The sub-goals can be generated using a course discretization of the environment state space and the primitive execution policy can then use any method appropriate to control the system as seen in the research of Morimoto and Doya [19].

7. Conclusions

Agents must learn quickly if they are to operate in high dimension environments. Providing an agent with domain knowledge and the ability to learn from observation can greatly improve its learning rate. The presented framework provides much flexibility in conducting learning from observation research using primitives. The use of this framework in giving agents the ability to operate in a real-life and simulated version of the air

hockey and marble maze environments demonstrates its usefulness.

Acknowledgements Support for all authors was provided by ATR Computational Neuroscience Laboratories, Department of Humanoid Robotics and Computational Neuroscience, and the Communications Research Laboratory (CRL). It was also supported in part by the National Science Foundation Award IIS-9711770.

References

[1] C. H. An, C. G. Atkeson, and J. 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. Robot learning from demonstration. In D. H. Fisher, Jr., editor, *Proceedings of the 1997 International Conference on Machine Learning (ICML97)*, pages 12–20. Morgan Kaufmann, 1997.

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

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

[7] D. C. Bentivegna and C. G. Atkeson. Learning from observation using primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Seoul, Korea, 2001.

[8] D. C. Bentivegna, C. G. Atkeson, and G. Cheng. Learning from observation and practice at the action generation level. In *IEEE-RAS International Conference on Humanoid Robotics (Humanoids 2003)*, Karlsruhe, Germany, 2003.

[9] D. C. Bentivegna, C. G. Atkeson, and G. Cheng. Learning to select primitives and generate sub-goals from practice. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, Las Vegas, Nevada, USA, 2003.

[10] D. C. Bentivegna, A. Ude, C. G. Atkeson, and G. Cheng. Humanoid robot learning and game playing using pc-based vision. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, Switzerland, 2002.

[11] 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.

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

[13] 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.

[14] K. Ikeuchi, J. Miura, T. Suehiro, and S. 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.

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

[16] 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.

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

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

[19] J. Morimoto and K. Doya. 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, 1998.

[20] C. B. Partridge and M. W. Spong. Control of planar rigid body sliding with impacts and friction. In *International Journal of Robotics Research*, pages 336–348, 2000.

[21] S. Schaal, C. Atkeson, and S. Vijayakumar. Scalable locally weighted statistical techniques for real time robot learning. In *Applied Intelligence - Special issue on Scalable Robotic Applications of Neural Networks*, volume 17, pages 49–60, 2002.

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

[23] S. Vijayakumar, A. DŚouza, T. Shibata, J. Conradt, and S. Schaal. Statistical learning for humanoid robots. In *Autonomous Robot*, volume 12, pages 55–69, 2002.

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

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

Darrin C. Bentivegna

Darrin Bentivegna is a graduate student in College of Computing at Georgia Institute of Technology and an Intern Researcher in the Department of Humanoid Robotics and Computational Neuroscience at the ATR Computational Neuroscience Laboratories. His primary research interest is in understanding methods that can give robots human-like intelligence and abilities.

Christopher G. Atkeson

Add Profile

Gordon Cheng

Add Profile