International Journal of Humanoid Robotics
Vol. 1, No. 4 (2004) 585–611
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

LEARNING TO ACT FROM OBSERVATION AND PRACTICE

DARRIN C. BENTIVEGNA
ATR Computational Neuroscience Laboratories
Department of Humanoid Robotics and Computational Neuroscience
Kyoto, 619-0288, Japan
Computational Brain Project, ICORP
Japan Science and Technology Agency, Kyoto, Japan

CHRISTOPHER G. ATKESON
ATR Computational Neuroscience Laboratories
Department of Humanoid Robotics and Computational Neuroscience
Kyoto, 619-0288, Japan
Carnegie Mellon University, Robotics Institute, Pittsburgh, PA, USA

ALEŠ UDE
ATR Computational Neuroscience Laboratories
Department of Humanoid Robotics and Computational Neuroscience
Kyoto, 619-0288, Japan
Jožef Stefan Institute, Department of Automatics
Biocybernetics and Robotics, Ljubljana, Slovenia

GORDON CHENG
ATR Computational Neuroscience Laboratories
Department of Humanoid Robotics and Computational Neuroscience
Kyoto, 619-0288, Japan
Computational Brain Project, ICORP
Japan Science and Technology Agency, Kyoto, Japan

Received 25 May 2004
Revised 11 June 2004
Accepted 15 June 2004

We present a method for humanoid robots to quickly learn new dynamic tasks from observing others and from practice. Ways in which the robot can adapt to initial and also changing conditions are described. Agents are given domain knowledge in the form of task primitives. A key element of our approach is to break learning problems up into as many simple learning problems as possible. We present a case study of a humanoid robot learning to play air hockey.

Keywords: Learning from observation; movement primitives; imitation; locally weighted learning; humanoid robot; air hockey.

1. Introduction

We are exploring learning from observation and learning from practice using primitives. This paper presents a case study of humanoid learning: we describe a system in which a robot learns to play air hockey from observing a human player and from practice. It also shows how learning at the action generation level is used to improve performance and quickly adapt to changing conditions. This research provides insight into how learning problems can be structured for quick learning from observation and practice. It is our hope that our approach will also lead to ideas and methods that can be used to automatically define primitives from observing a task.

In this research we found that quickly adapting to small changes in the task was very important for acceptable performance by the humanoid robot. More generally, for humanoid robots to be accepted as partners with humans they will be expected to learn quickly and adapt to changes in ways similar to humans. Within a single task there may be many things that can change as humans interact with the robot. The placement of items within the workspace, the physics of the task, and the reaction of the human to the robot's movements are some examples of the things which can vary during the interaction.

1.1. *Why primitives?*

We are exploring learning using primitives in both air hockey and a marble maze task.[4] The agents can perform surprisingly well using only observed information, but are capable of even better performance if they are allowed to learn from practicing the task and adapt to any changes in task conditions. Using primitives in learning is helpful in several ways: (i) it speeds up learning; (ii) primitives can improve generalization to other tasks; (iii) we think humans use primitives, and we are attempting to match our primitives to task-level primitives that we think the humans are using. This helps our humanoid robot behave in a human-like fashion.

We have developed a three level approach (Fig. 1).[4] The first level, *Primitive Selection*, allows the robot to learn which type of primitive to use in any given situation. The second level, *Subgoal Generation*, allows the robot to learn which subgoals a human tries to achieve. The third level, *Action Generation*, allows the robot to learn how to achieve the desired subgoals. Having these three levels separates and simplifies different learning problems: what kind of shot to take (behavior selection)? Where to aim the shot (parameter generation)? How to make that particular shot (execution)? This paper discusses learning from observation at all three levels, and learning from practice at the action generation level.

1.2. *Why air hockey?*

Air hockey is a game played by two players. They use round paddles to hit a flat round puck across a table. Air is forced up through many tiny holes in the table's surface, which creates a cushion of air for the puck to slide on with relatively little

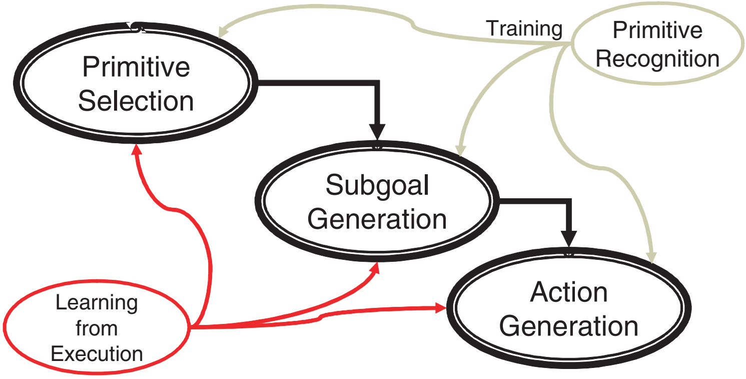

Fig. 1. Our framework.

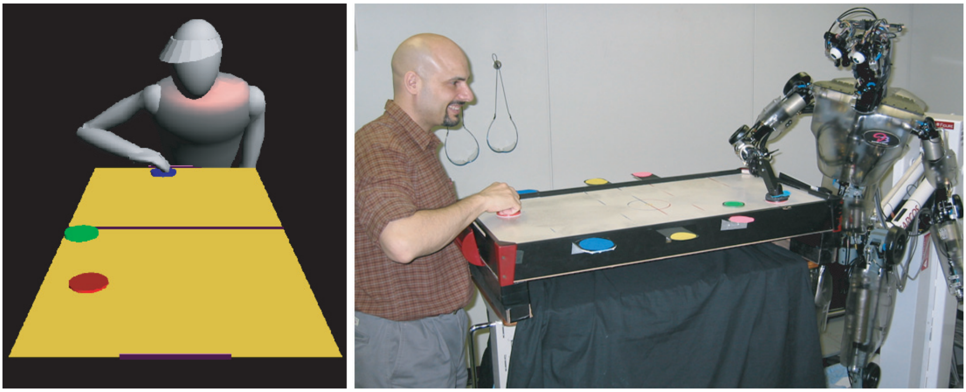

Fig. 2. Simulated air hockey (left) and playing air hockey with a humanoid robot (right). In the simulated game the disc shaped object near the centerline is a puck that slides on the table and bounces off the sides, and the other two disc shaped objects are the paddles. The virtual player controls the far paddle, and a human player controls the near paddle by moving the mouse. The object of the game is to score points by making the puck hit the opposite goal (the marked area at the ends of the board). In the physical game the paddles are held by the human and the humanoid robot, and the puck is about to be hit by the robot.

friction. The table has an edge around it that prevents the puck from going off the table, and the puck bounces off this edge with little loss of velocity. At each end of the table there is a goal area. The objective of the game is to hit the puck so that it goes into the opponent's goal area while also preventing it from going into your own goal area.

A simulated and a hardware version of air hockey have been created as testbeds for this research (Fig. 2).[3] There are many reasons why air hockey is an interesting task for this case study. It is a dynamic task where the puck is almost always moving. It is fast, requiring rapid perception, thinking, and movements. It is demanding, requiring considerable movement accuracy. It is complex, when perceptual and movement time delays, board placement and leveling, varying board surface conditions and friction, air flow, temperature effects, and actuator dynamics are taken into account. There are disturbances, modeling errors, and an adversary, so there is much to learn. The task is small and simple enough to be implemented on a computer in simulation and in a normal size laboratory. Since the playing area is two-dimensional, sensing and moving is simplified. Air hockey is closely related to racquet sports such as ping-pong and tennis, as well as other interactive games such as playing catch. We believe that our approach to air hockey can generalize to a wide range of intermittent dynamic tasks.

The air hockey task is also being explored by Spong and colleagues.[6,11,12] Bishop *et al.* discuss difficulties, such as table position errors and specifying a puck movement model, that our learning and adaptation models try to overcome.[6] Whereas the research of Spong *et al.* provides insight into the physical interactions of air hockey, our research seeks methods for humanoid robots to quickly learn task strategies and models through observation and practice.

1.3. *Why adaptive action generation?*

In our approach much of the ability to rapidly adapt to changes in task conditions is at the action generation level. We believe adaptive action generation is necessary for successfully behaving in human environments, and it is an ability expected of humanoid robots. We cannot rely on structuring the environment, as is done in much factory automation.

2. The Air Hockey Task

The hardware air hockey task has been created using a humanoid robot (Fig. 2).[1] The humanoid robot has 30 degrees of freedom and is 190 cm tall and weighs 85 kg. 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 manually defined primitives that we are exploring for this task are:

- **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.

This paper will focus on the selection and execution of the shot primitives (**Straight Shot** and **Bank Shot**). The details of action generation will be presented to show how learning and adaptation are used. During the shot primitives the puck and the robot are moving very quickly. Because movement errors and errors in the prediction of the future puck state can have a large effect on the outcome, the shot primitives provide a good opportunity for learning.

2.1. *Perceiving the position of the objects*

In order to play air hockey, the robot must be able to sense object locations in real-time. The task is made more difficult because the robot uses its own eyes (cameras) to see the game. Combined real world and camera motions lead to fast image motions.

2.1.1. *Calibration*

Since air hockey is played on a flat surface, we can model the image plane to hockey board mapping as a perspective mapping between two planes. It is well known that such a mapping can be modeled by a 3 × 3 homography, which is defined up to a scale factor and thus has eight degrees of freedom.[18] Since this mapping is invertible, the information from one eye (camera) suffices to uniquely determine the position of the puck on the board. However, we must be able to update this mapping at every measurement time because the robot's head moves during the game. In theory, we could do this by calibrating the camera at a preferred configuration and use forward kinematics to calculate the current image-to-board mapping, but this is impractical because the humanoid robot motion involves many degrees of freedom and is highly nonlinear. It is therefore better to recalibrate the system at every time step. Since every homography has eight degrees of freedom, we must know the position of at least four points on the table and in the image to recalibrate the camera at every measurement. This increases the number of objects that we need to track to at least seven; four fixed points on the board for calibration, puck, and both paddles. To make the recalibration more accurate and the tracking process more robust we have created an implementation that uses six points to recalibrate the system.

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

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

where $\boldsymbol{x}_i(t) = [x_i(t), y_i(t), 1]^T$ are the known positions of the markers on the hockey board (measured by hand) and $\boldsymbol{u}_i(t) = [u_i(t), v_i(t), 1]^T$ are the positions of the detected markers in the image. Note that we calculate the homography from the image plane to the hockey board. Let $\boldsymbol{h}_1(t)$, $\boldsymbol{h}_2(t)$ and $\boldsymbol{h}_3(t)$ be the columns of

$\boldsymbol{H}(t)$. Writing $\boldsymbol{z}(t) = [\boldsymbol{h}_1(t)^T, \boldsymbol{h}_2(t)^T, \boldsymbol{h}_3(t)^T]^T$, Eq. (1) can be rewritten as

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

Writing Eq. (2) in a matrix form results in a matrix equation $\boldsymbol{A}(t)\boldsymbol{z}(t) = 0$, where $\boldsymbol{A}(t)$ is a $2N \times 9$ matrix. As $\boldsymbol{H}(t)$ is defined only up to a scaling factor, the solution is well known to be the eigenvector associated with the smallest eigenvalue of the 9×9 matrix $\boldsymbol{A}^T(t)\boldsymbol{A}(t)$.[18] Alternatively, one could solve Eq. (2) directly by setting one of the parameters, typically $h_{3,3}(t)$, to 1. See Ref. 14 for details about the color tracking system used to observe objects.

2.1.2. *Strategy for error recovery*

Typically, a game of air hockey goes on for several minutes and the vision system is expected to provide locations of the objects of interest during this period. It is extremely annoying if the data collection or the actual hockey game must be stopped due to the failure of the vision system. Since it may not be possible to completely avoid tracking failures, we designed a specialized error recovery scheme that ensures the successful operation of the vision system over longer periods of time.

There are a few situations when the robot has difficulty observing the position of the objects. The most common problem is that the robot arm sometimes completely occludes the puck and it is therefore not possible to locate it, see Fig. 3. When this happens, the tracker keeps looking for the puck for half a second at the latest detected location. After half a second it is considered unlikely that the puck would still be near this position and the tracker starts two new search processes: one in front of the opponent's paddle where we can assume that there will be contact with the puck in the future and the other one in the region near the robot where the puck might still be situated in the case of an occlusion. The second process randomly generates initial puck positions within the search region and thus ensuring that the

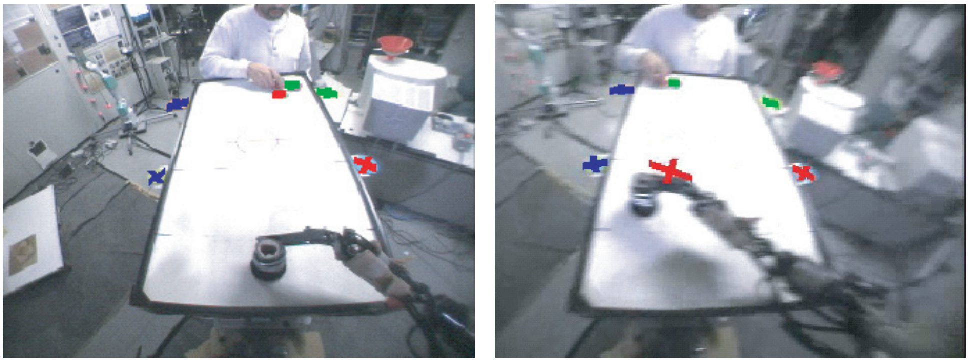

Fig. 3. The view from the robot's eyes with the tracked objects marked. The puck is totally occluded in the right image. The right image is also blurred due to the movement of the robot head.

puck will eventually be found regardless of its current position within this region. It is important to note that we do not embark on an exhaustive search covering the whole board as that could not be carried out at 60 Hz. The success of one of the search processes terminates them both.

Occasionally one of the side blobs that is used for recalibration is lost by the tracker. This is most often caused by very fast motion of the robot's head when the robot performs a shot, which results in low-quality images and huge image motions. If the robot cannot see at least four side blobs, the recalibration cannot be carried out and the robot cannot play the game. Fortunately, this problem is not too serious because the robot calculates the motion trajectory before it starts executing the shot. After executing the shot, the robot returns to a preferred configuration and since the position of the board remains nearly constant, we can store the positions of all of the side blobs at the preferred configuration and restart the tracking using the stored positions as initial values. The side blobs are detected again when the robot returns to its preferred configuration after the shot execution.

Although problems with the paddle tracking are rare, we have nevertheless implemented an error recovery scheme. It is based on the fact that the motion of both paddles is confined to a region along the opposite ends of the hockey board. Thus if one of the paddles is lost, we can start a search process in a region along the appropriate end of the board.

Figure 4 shows the results of the visual tracking system using four fixed markers during approximately 4.2 seconds of game play. During this interval the fixed markers are always being tracked, but on several occasions the puck was lost and

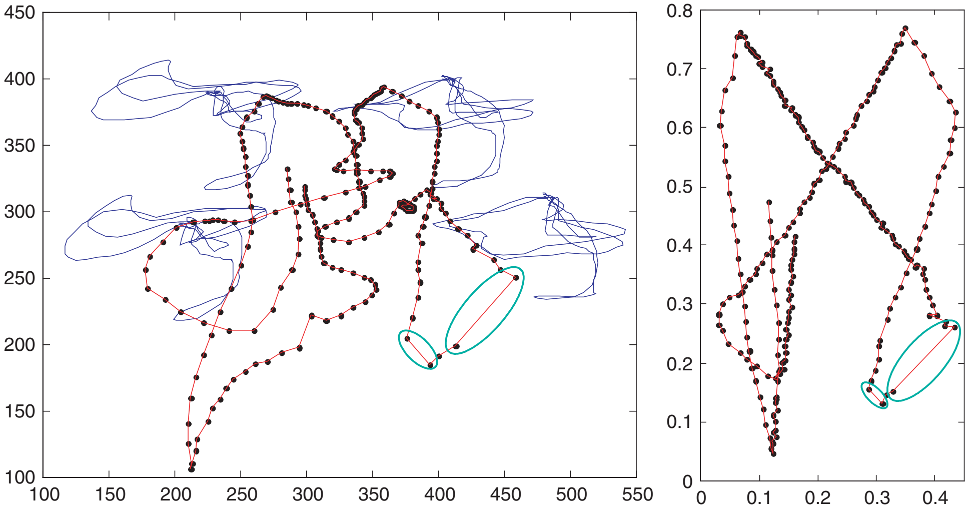

Fig. 4. The left graph shows the raw vision cordinates (pixels) of four objects placed at known locations and the moving puck. On the right is the computed position (meters) of the puck based on the information shown on the left. The circled segments are where the vision system lost track of the puck for several samples.

then reacquired. The puck was lost just after the robot hit it. This is not a critical time since the robot's hit motions are fully planned and begin more than 100 msec before the puck is hit.

2.2. *Robot positioning*

An interpolation scheme is used to solve the redundant inverse kinematics problem to position the paddle on the playing surface. Figure 5 shows six manually generated body configurations that place the paddle in different locations. To position the paddle at any location within these configurations, the closest four configurations are combined using a bilinear interpolation method. While this approach is simple and allows us to solve the redundant inverse kinematics problem, we have found that the accuracy of positioning the paddle is affected by many things such as the initial position of the board, the paddle's movement speed, and the friction between the paddle and the board. More information on the vision system and paddle positioning method can be found in Bentivegna *et al.*[5]

3. Retrieving Information from Observed Data

To learn from observation, the robot must first have the ability to retrieve appropriate 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

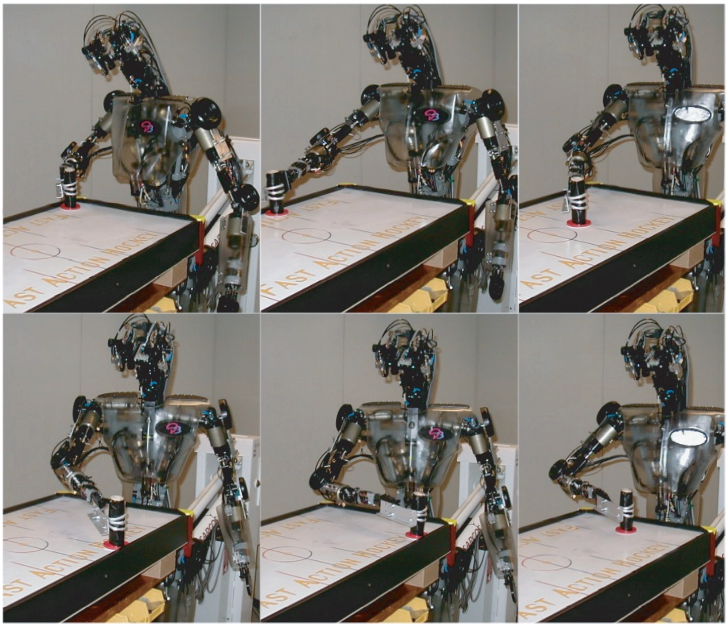

Fig. 5. Six manually defined configurations are used to compute all enclosed configurations.

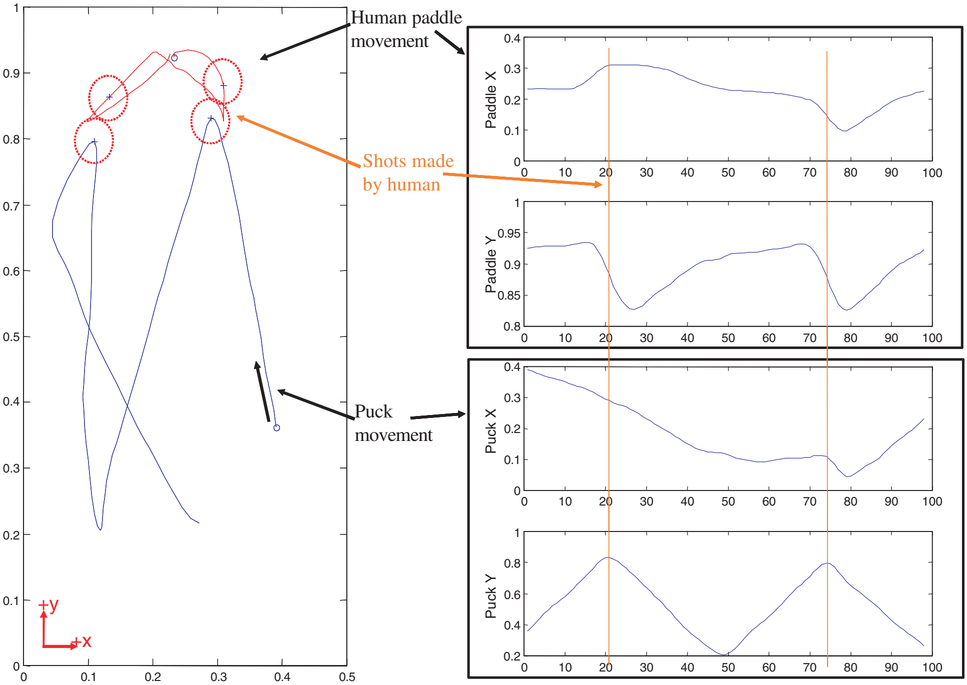

Fig. 6. Raw data collected while observing a human making shots in air hockey. The left figure shows the data plotted in two dimensions (meters). The right figure shows the object positions plotted against time and collisions with the puck can be easily seen in this figure.

is happening that can easily be seen. A puck hitting a wall or a paddle is an example of a critical event. Figure 6 shows data collected from observing hardware air hockey. 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:

(i) 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.

(ii) 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.

(iii) 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.

(iv) 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 observations the following information can be collected each time a shot is observed:

- the position and velocity of the puck shortly before it crosses the center-line heading toward the player;
- the position and velocity of the puck when it is hit;
- the relative location of 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.

Databases are created from the observed data that are used by the primitive selection, subgoal generation, and action generation modules. This section describes how the observed information is formatted to be used to select a shot type and subgoals when the puck is traveling toward the player. Section 5 describes how the observed information is formatted and used by the action generation module.

A database is created to encode the actions taken by an observed player when the puck crosses a pre-specified line heading towards the player. The actions contained in the database are the **Straight Shot**, **Bank Shot**, and **Defend Goal** primitives. The pre-specified line, called the decision line, is on the opponent's side and is just out of reach of the opponent, Fig. 7. This means that the puck can no longer be influenced by the opponent's actions and future puck positions can be predicted. If, after the puck crosses the decision line, a collision with the player's paddle is observed and the puck goes toward the opposite side, it is classified as a shot primitive. In all other situations where the puck does not enter the player's goal, it is classified as a **Defend Goal** primitive.

If the **Defend Goal** primitive is observed, the position and velocity of the puck when it crosses the decision line and the location the paddle is moved to defend the goal are recorded. If one of the shot primitives is observed, the following information is recorded (Fig. 7):

- $(x_{dl}, y_{dl}, \dot{x}_{dl}, \dot{y}_{dl})$ — the position and velocity of the puck when it crosses the decision line;
- y_{hit} — the line where the puck was hit;
- *PrimType* — the type of shot that was taken;
- *PuckSpeed* — the speed of the puck after it is hit;
- x_{target} — the target location on the back wall that the puck moved to after being hit.

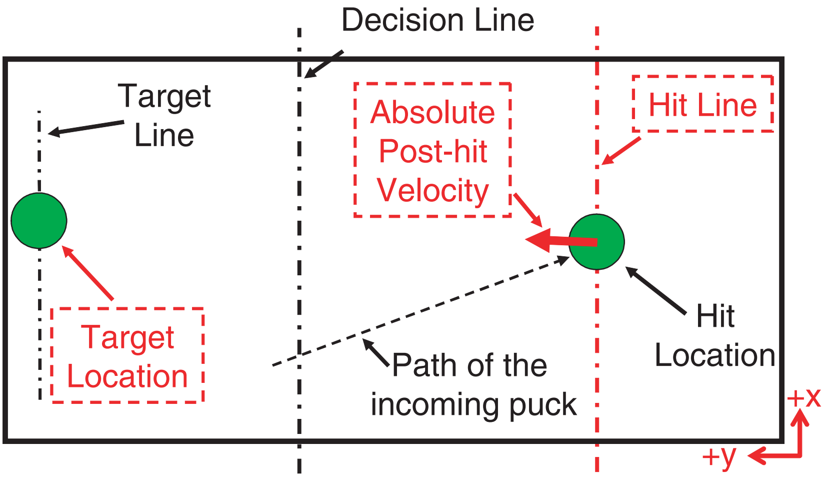

Fig. 7. Information recorded in the database when a shot is observed.

4. Choosing a Primitive to Use and Generating Subgoals

When the robot is playing air hockey and observes the puck travel toward it from the opposite side it will use the database to first select a primitive type to use. The primitive selection module accomplishes this by finding the data points in the primitive database that are closest to the observed state. This is done by comparing the distance of each data point from the observed state, or query point. The distance is given by $d(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_j w_j \cdot (\mathbf{x}_j - \mathbf{q}_j)^2}$, where $\mathbf{x}$ and $\mathbf{q}$ are the locations of the data point and the query point, and w allows each dimension to be weighted differently. For air hockey the query is the state of the puck when it crosses the decision line $(x_{dl}, y_{dl}, \dot{x}_{dl}, \dot{y}_{dl})$.

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

Once the primitive type has been chosen the subgoal can be computed. It is important to first choose the primitive type because the subgoals of different primitive types may not be compatible. For example, it would not make sense to use the subgoal of the **Defend Goal** primitive with the **Straight Shot** primitive. The **Straight Shot** primitive will be expecting a target position on the back wall as a subgoal location and the **Defend Goal** primitive will be specifying a subgoal location to which to move the paddle to defend the player's goal.

The closest data point's information can be used as the source of the subgoal as well. A more robust approach is to use the N closest points with the selected primitive type to compute the subgoal. The outcomes of the returned points are used to compute the subgoal using a locally weighted learning (LWL) model[2]:

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

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

After a shot primitive is chosen by the primitive selection module, the subgoal generation module, using the previously observed information, will specify a line at which the hit should take place (y_{hit}), the target location to shoot the puck at (x_{target}), and the desired post-hit puck speed (*PuckSpeed*) as shown in Fig. 7. The line at which the hit should take place tells the action generation module *where* the action should occur. Because the puck is moving it also provides an indication of *when* the action should occur. The target location tells *what* the desired outcome of performing this action is. The target location is not fixed at the center of the opponent's goal but can vary along a line across the back wall as shown in Fig. 7. Shooting the puck to a target location can be done at a variety of speeds. How fast (post-hit puck speed) a player makes shots will have a large effect on their performance. Therefore, the desired post-hit puck speed is supplied by the subgoal generation module and gives an indication of *how* the action should be performed.

5. Shot Action Generation

Once the shot type has been selected and the desired outcome has been specified the action generation modules must generate appropriate actions. This section and the next describe the details of these modules and how they learn to control the robot and learn about interactions from observation and practice. Section 3 showed how training information for the action generation modules is obtained from observing the task. The robot first has the opportunity to observe shots taken by the human player and can then observe its own shots. Successful shots only occur at the rate of 10 to 15 per minute. Because of this, unless the game is watched or practiced for a long period of time, there is little information to learn from. Therefore, we have structured our learning system to take advantage of the observed information as efficiently as possible.

5.1. *Making the most of observed information*

To increase the usefulness of the observed information, the hockey shot data used for the **Bank Shot** and **Straight Shot** primitives are represented in a local coordinate frame. Using a local representation allows a single **Bank Shot** model to be created from observing both left and right bank shots. The local coordinate frame for the **Bank Shot** action generation module uses the wall that the puck will hit as the x reference. The reference point $(0, 0)$, shown in Fig. 8, is where the puck hit location lines up with the side wall. The x-coordinate is positive in the direction from the wall toward the puck and the y-coordinate is positive in the direction of the opponent's goal. When a left or right bank shot is observed, puck and paddle parameters are transformed into the local coordinate frame before being provided to the action generation model.

In the action generation module for the **Straight Shot** primitive the observed information is transformed to a local coordinate system that is centered on the position at which the puck is hit, also shown in Fig. 8. Positive y is toward the opponent's goal and positive x is to the right. This will generalize shots that have approximately the same incoming velocity vector and a similar target displacement and post-hit puck velocities.

5.2. *Creating models of the task*

A robot trajectory leads to the paddle hitting the puck, and subsequent puck motion. An action generation module must invert this process, and find a robot trajectory that causes the puck to hit the target location (Fig. 9). In this paper we present methods to learn the *Puck Motion, Impact* and *Robot* models. This section describes how the **Straight Shot** and **Bank Shot** primitive action generation modules specify the paddle parameters needed to perform the primitive as specified by the subgoal generation module. Section 6 describes how the robot learns to move the paddle.

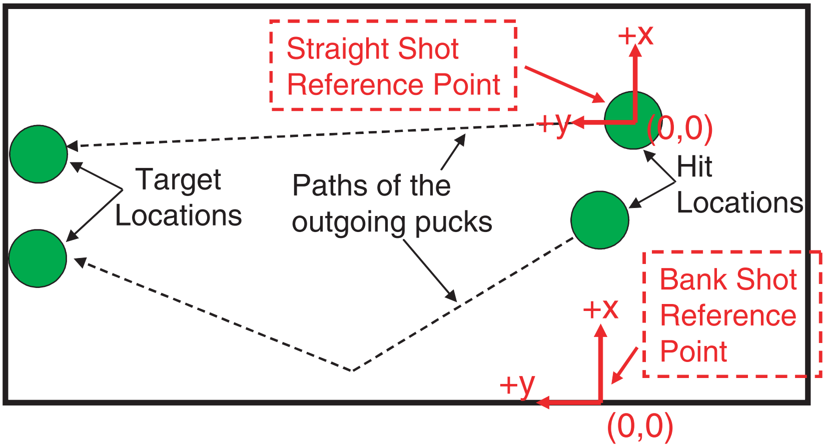

Fig. 8. Bank shot and straight shot coordinate frames.

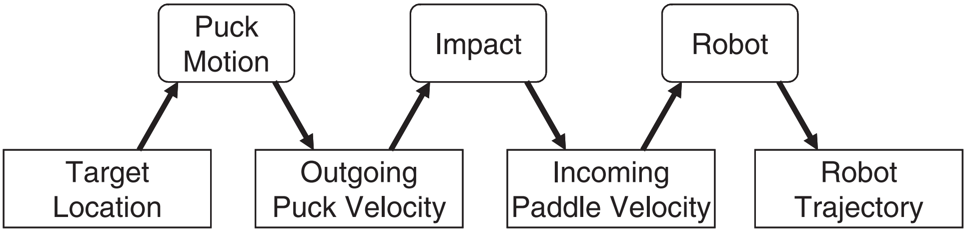

Fig. 9. The models involved in action generation.

We use a locally weighted learning technique, Locally Weighted Projection Regression (LWPR),[16] to represent these learned models. 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.[2] 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, using 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[13] and inverse kinematics learning.[15] The *Puck Motion, Impact* and *Robot* models are implemented as LWPR models that are trained from the observed information These models are used during action generation in the **Straight Shot** and **Bank Shot** primitives.

5.3. *Learning the puck motion model*

The puck motion model predicts the direction to hit the puck, given a target, where to hit the puck, and the speed of the puck after the hit. To obtain this information we used the data obtained from observation to train one LWPR model for the **Straight Shot** primitive and another for the **Bank Shot** primitive. The inputs to these models are as follows:

- **Straight Shot** *Puck Motion* model:
  — the target position (x, y);
  — the desired post hit speed of the puck.
- **Bank Shot** *Puck Motion* model:
  — the target position (x, y);
  — the puck position x at the time it is hit;
  — the desired post hit speed of the puck.

The specified locations are in the local coordinate frames of the primitives as discussed in Sec. 5.1. Therefore, the puck for the straight shot always starts at the origin and the target is the desired x and y displacement. For the bank shot the puck always starts at the line $y = 0$ and the y value in the target is the desired y displacement of the puck. The output of both models is the direction in which the puck should travel to reach the given target point from the location the puck will be hit. This information, along with the desired post-hit speed provided by the subgoal generation module, is then used by the *impact* model to generate the paddle parameters needed to hit the puck to cause it to have the correct post hit velocity (magnitude and direction).

5.4. *Learning the impact model*

The *impact* model must specify where the paddle should be relative to the puck, and the paddle's velocity at the time of the hit. One way to compute the needed paddle state at hit time is to use a pre-specified model of the interaction such as the one presented by Partridge and Spong.[12] If the model parameters cannot be precisely defined and determined, the model will not be accurate. Partridge and Spong assume that the mass of the paddle is much higher than that of the puck and therefore the paddle's velocity is unchanged by the collision. This does not seem to be the case in hardware air hockey where small changes in the paddle's trajectory can sometimes be observed at the point where it hits the puck. Therefore, in our implementation, the effective mass, damping, and stiffness of the robot are additional items that have an effect on the outgoing puck's velocity. It would be difficult to define a parametric model for our humanoid robot that would include all the items that have an effect on the puck's outgoing velocity. It is also difficult to evaluate the significance of each item to determine which items can be ignored. For these reasons we would like to have the robot learn the *impact* model in the same way it learns the *puck motion* model.

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

- Input:
  - the velocity $(\dot{x}, \dot{y})$ of the puck when it is hit;
  - the desired puck speed after it is hit;
  - the desired movement direction of the puck.
- Output:
  - the angle between the puck and the paddle at the hit time;
  - the movement direction of the paddle when it hits the puck;
  - the speed of the paddle when it hits the puck.

5.5. *Model learning in the simulator*

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

The solid line in Fig. 10 shows the result of the agent 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 10 plots the average absolute error in hitting the target location, the distance between the target location and the location where the puck actually hit the back wall. The values plotted are the running average of five shots. 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.

The width of the goal is 20 cm and from Fig. 10, it can be seen that if the agent was targeting the center of the goal it would be in range to enter the goal most of the time. But by comparing this agent to the perfect agent, it appears it can perform better than this. One way to increase its performance is to have it observe the human making more shots. But this can be time consuming as it took over ten minutes to see only 152 shots. A better way is to have the agent observe its own behavior and add that information to the models.

After making 200 straight shots using the models learned from observing the human, the agent then observed 100 of its own shots while practicing (shots 201 to 300 in Fig. 10). 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 10 shows the result of using these newly trained models for the shots from 301 to 500.

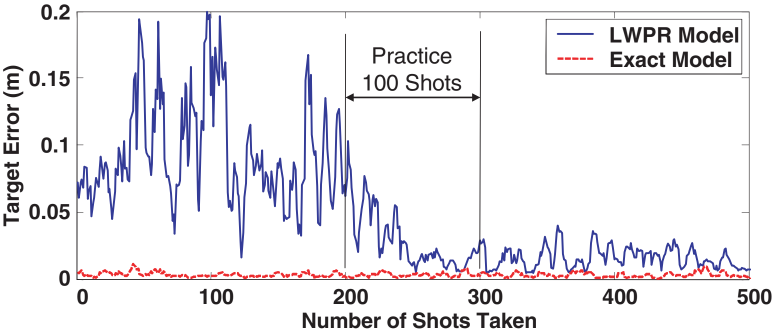

Fig. 10. This graph shows the magnitude of the error in reaching the target location during 500 straight shots made by the agent in simulated air hockey. The solid line shows the result of the agent making 200 shots using the LWPR model trained from observing 44 straight shots performed by the human. It then observes 100 of its own shots while practicing and adds that information to the LWPR model. The dashed line is the result of an agent making straight shots using an exact model of the task. The graph shows the running average of five shots.

6. Action Learning in Physical Air Hockey

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

The humanoid robot positions the paddle on the table using the interpolation method described in Sec. 2. This is a simple and useful method for positioning the paddle, but if the table is not accurately placed, or moved during the task, there will be an error in positioning the paddle on the table. We have found that paddle placement accuracy is also affected by the desired movement velocity. The accuracy is much higher during slow shot maneuvers than during fast shot maneuvers. A reason for this is due to the design of the robot and the fact that some of the degrees of freedom (DOFs) are reaching their maximum velocity or torque limits. One way to reduce the effect of this problem is to ensure that we keep the desired movement velocity lower than the slowest DOF. But this would severely limit the robot's ability to perform this task. The robot has shown that it is capable of moving the paddle at velocities close to those of a human player. The problem is that the paddle does not always correctly follow the specified trajectory at these high velocities.

For the task of making shots in air hockey, there is only one important instant, and that is when the puck and paddle collide. It is at this instant that the paddle affects the movement of the puck. Therefore, it is not the entire trajectory that is important, but the state of the paddle at the instant it hits the puck. If the robot can repeatedly control the paddle to be in the correct state at the correct time, it can make accurate shots in air hockey irrespective of the path the paddle takes to arrive at that hit point.

6.1. *Learning the robot model*

The robot model must generate a trajectory which has the paddle arrive at the hit location at the correct time with the correct velocity. The robot command consists of a desired location and a time in which to reach that location. The movement follows a fifth order polynomial with zero start and end velocities and accelerations. It therefore should have its highest velocity in the middle of the movement and this is where the puck-paddle collision should occur. But if the board is not in the correct position, or the robot is trying to move with too high a velocity, the paddle will not be at the correct hit point, with the correct velocity, at the correct time. The left graph in Fig. 11 shows the path, the lines with the boxes on them, of three hit maneuvers. The lines with the circles on them are the desired trajectories. The robot's initial position for the three maneuvers is approximately (0.235, 0.05). The robot is commanded to move to the position (0.31, 0.16) in 14.3 ms. The graph

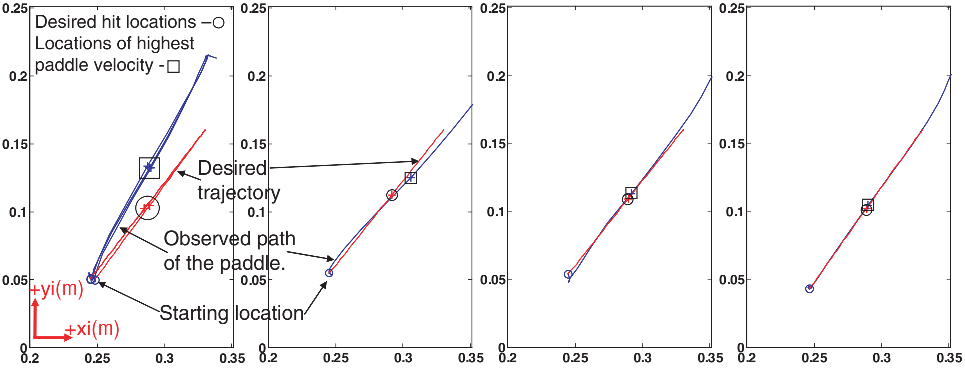

Fig. 11. The lines with the boxes on them in these graphs show the path of the paddle during shot maneuvers. The lines with the circles on them are the desired trajectories. The graph on the left shows three similar shot attempts overlaid. The errors are due to board placement errors and the robot's variance in making shots at different velocities. The three right graphs show the same shot maneuver being made using a trained *robot* model.

shows where the paddle-puck collision is expected to occur and where in the actual trajectory of the paddle the highest velocity is observed. From this graph it can be seen that there is a repeatable error for this given command and there should exist some set of commands that would place the puck at the desired hit location with the desired velocity vector. It is the function of the *robot* model to learn the robot commands that will correctly place the paddle at the time of the hit.

The robot learns this model when the board is placed and play is about to begin. At this time the robot makes a set sequence of 30 paddle movements that would be typical for making hockey shots. When a movement is commanded, the starting location of the paddle as seen by the robot, the desired movement command, and the time the command is sent to the robot are recorded. The movement of the paddle is then observed and the paddle velocity is computed whenever new vision data arrives (60 Hz). The paddle's position and velocities, and the time of the observation, are recorded. When the paddle velocity returns to zero, a data point is created with the following information:

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

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

6.2. *Using the robot model*

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

The three graphs on the right in Fig. 11 show the results of using this trained *robot* model to make the same maneuver as the one shown in the graph on the left. Each graph shows one hit maneuver and the robot starts in approximately the same initial state. The initial paddle position and desired paddle hit state are input to the *robot* model and the values returned from the model are used as the movement command. The graphs show that the model has learned to more accurately place the paddle at the desired hit location with the velocity vector pointing in the correct direction and the location at which the highest paddle velocity is seen is now much closer to the desired hit position.

6.3. *Using the timing information*

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

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

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

Figure 12 shows the path of the puck and the paddles during three 2-second intervals of game play. The humanoid robot is on the left and the human player is on

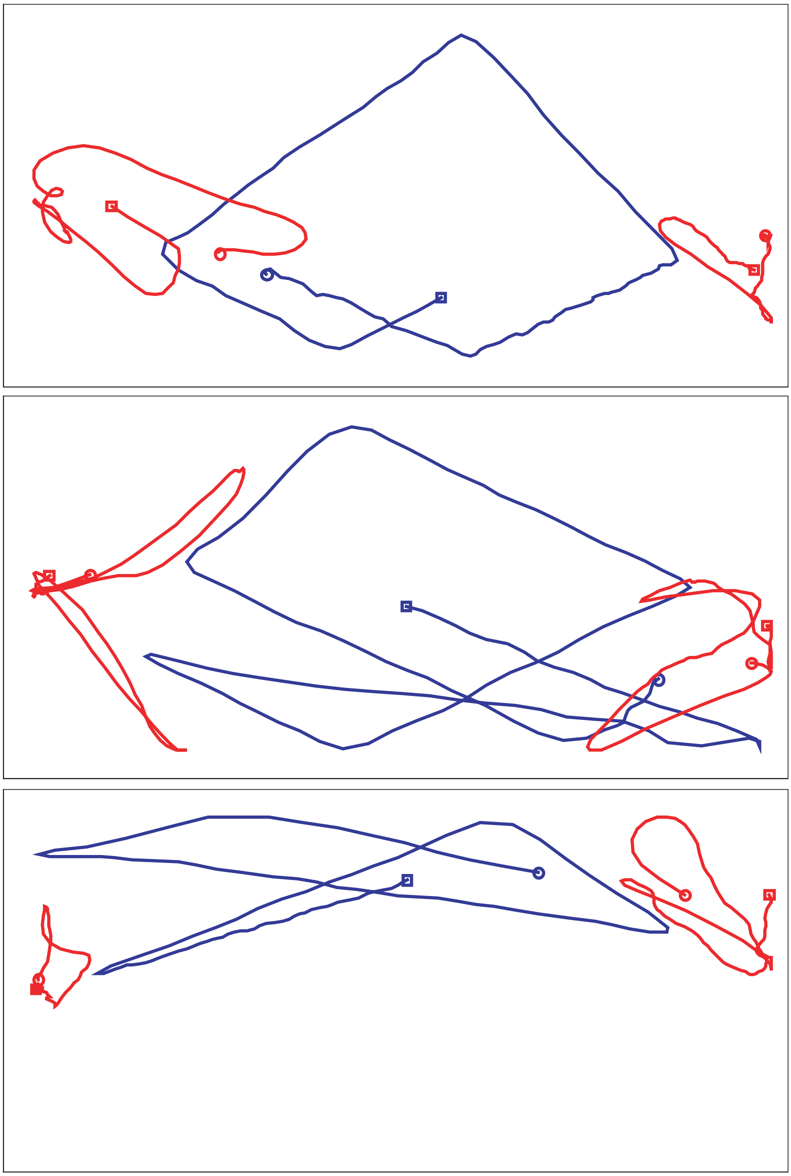

Fig. 12. The paths of the puck and paddles during three 2-second intervals of game play with a humanoid robot (left) and a human (right). The "◯" symbol denotes the start of the path and the "□" denotes the end of the path.

the right. Figure 13 shows the position of the puck and paddles plotted against time for 10 seconds of game play. The top graph plots the objects y positions on the same graph and most puck-paddle collisions can easily be seen on this graph. The bottom three graphs in Fig. 13 show the x position of the paddles and puck during the same

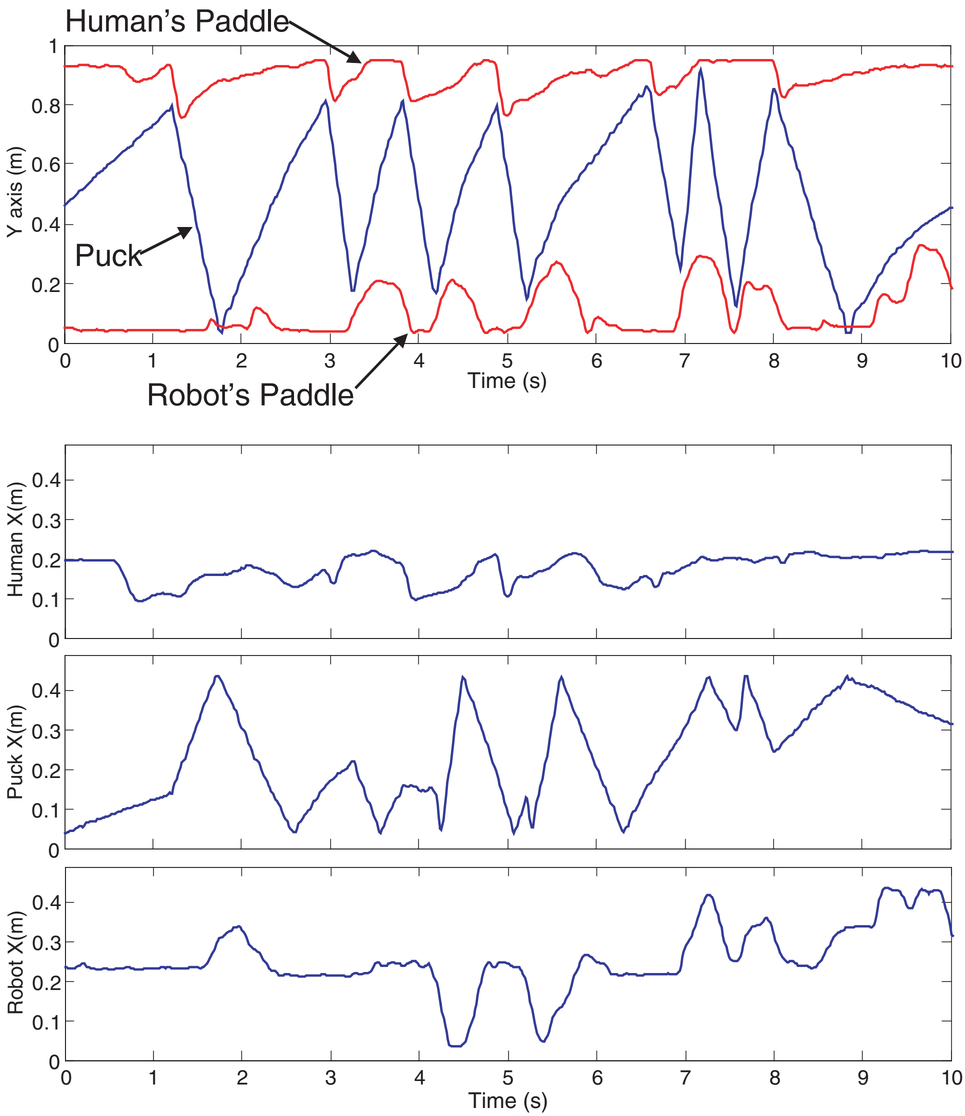

Fig. 13. The position of objects in air hockey plotted against time. The top graph shows the y position of all objects on the same graph. The bottom three graphs show the x position of the objects during the same time.

time period. The level of performance in Figs. 12 and 13 is based on 20 minutes of observed human play and 10 minutes of practice. The regression parameters for selecting primitives and computing subgoals are as follows: the number of data points used in the regression $(N) = 5$, each of the dimensions are scaled to ± 1.0 and the weight of each dimension is 1.0, and the kernel function is $e^{-d^2/\alpha}$, where $\alpha = 1.0$.

7. Discussion and Future Research

7.1. *How to structure learning*

One purpose of this paper is to explore the utility of having a lot of structure in learning. Section 1 explains our motivation in using primitives in robot learning. This motivation leads us to seek out a structure for robot learning that supports fast learning from observing and practice and has the ability to quickly adapt to changes. This paper uses the air hockey task as a case study in which we show how action generation has been structured to make efficient use of the observed information. The action generation module contains models that allow the robot to learn specific skills that may generalize to other tasks. This paper presents the results of training these models from information obtained whenever a shot is observed. By structuring the problem in the manner that we have, the robot has the opportunity to learn and improve the performance of the models within the action generation module at times other than only observing a full shot sequence. The *impact* model, for example, can learn about the impact interaction whenever a paddle-puck collision is observed. The ultimate outcome of the shot is irrelevant to this model. The same is true for the *puck motion* and *robot* models. These models can be trained whenever the puck is seen moving from the hit area to the goal area or the robot attempts to make a shot. We are currently adding this ability, learning from smaller temporal interactions, to our humanoid robot.

We are also exploring ways in which we can further break large models into combinations of smaller ones. There are two *puck motion* models that provide the desired velocity vector; one for straight shots and one bank shots. The bank shot has at least three clear segments: (i) the puck moves from the hit point to the wall, (ii) undergoes a change in velocity during the wall-puck collision, and (iii) travels to the target position. Segments 1 and 3 are the same as the straight shot model. Therefore, it may be possible to have the system learn faster and be more accurate by structuring the problem to have a no-collision *puck motion* model and a *wall-puck collision* model that coordinate to provide the needed information for a bank shot.

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

7.2. *When and what to generalize?*

As discussed in the previous section, combining the left and right bank shots into one learning module provides the robot with more training data to improve the performance of this module. But should this be done? Can a left bank shot and a right bank shot be transformed to a standard shot? If the board is symmetric, it appears that this can be done as shown by the results presented in Sec. 5.5. But what if the board is not symmetric or is tilted to the side? It would be helpful if the agent had methods to detect when information can be combined and when it cannot.

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

7.3. *When to forget*

In our work so far, humans decide when information is added to models and when data is forgotten and replaced with new data. A future step would be for the robot to have control over its own learning and decide for itself when models need further training or replacing. If the robot has the ability to continually evaluate its movements and the outcome of its actions, it can use this information to decide if previously learned models are no longer accurate and should be updated or replaced. In air hockey, for example, if the board is suddenly moved during the game, the robot should immediately notice an error in its paddle movements. It can then attempt to add new data to the *robot* model in an effort to have the model adapt to the new board position. When data is added to the current *robot* model, the data can also be use to train an entirely new, off-line, *robot* model. If the current model is not adapting to the changed situation, it can be replaced by the new model.

7.4. *Perceptual learning*

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

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

We currently use a parametric model with learned parameters to predict the future state of the puck. The accuracy of the future predicted puck state is determined by the accuracy at which the vision delays and task parameters are known. Section 6 shows how we are taking into account the vision delays when initiating movements in air hockey. But if our model parameters are not correct, the puck will not be at the predicted hit position when the paddle arrives there. The model parameters that are currently being used are global and remain constant. It is likely that the friction is not constant across the playing surface and changes over time due to factors such as the fan motor wearing out. For these reasons the robot should have the ability to learn and update a model of the puck's movements from observing the task. The next paragraph describes an implementation that we are currently exploring.

As mentioned in Sec. 6, the most important instant is when the paddle and puck collide. This collision occurs within a small range near the robot. Our continuing research is exploring a method in which the robot observes the position of the puck just beyond the center line for two or three vision cycles. The state of the puck when it crosses the desired hit line is then observed. It is our hope that this information can be used to train an LWPR model that will provide the robot with the ability to accurately predict the puck's state at the hit time. The research of Park *et al.*[11] shows that a neural net can be trained to provide this prediction on a larger air hockey table given the puck's position and velocity as input. But to accurately compute the puck's velocity, more than two observations will be needed. The increased size of the table used by Park *et al.* provides them with more opportunities to observe the puck in locations that are out of reach of the players. Their model is also trained with 3,000 observations. We would like the humanoid robot player to learn from much less data and have the ability to update the model while it is operating.

7.5. *Automatically discovering primitives and structure*

In many research fields there is a large interest in methods agents can use to automatically define a library of primitives or actions from observing the performance of a task.[7,8,10] Even though in our research we have manually defined the library of primitives, we also have a strong interest in automating this process. Our research in creating a method in which robots learn to operate in dynamic tasks using a library of primitives and observing others gives insight into the type of information that an automated agent would need to know about and search for. Our method of breaking the learning problem into small independent models can assist in automatically discovering primitives in many ways. Wolpert and Kawato's[17] research on approaches that have the ability to learn forward and inverse models of task components provides insight into how the models can be configured to predict events. Figure 9 shows how a primitive can be composed of a sequence of model activations. The state can be observed and recorded at the beginning and end of this sequence to provide all the information needed for an agent using our framework.

One way the models can be used is to describe which events are occurring as they are fed the observed data. The sequence of model activations can be recorded as the task is observed. The research of Kaminka *et al.*[9] on learning the sequential behavior of teams from observation gives some ideas into how the sequential list can be represented and used to discover primitives. For example, a sequence of model activations that are seen reoccurring many times is a good indication of a primitive. By structuring our models to be general they can also become activated while observing similar events in other tasks. This has the effect that as the agent learns more models, it has an increased ability to discover primitives.

8. Conclusions

This paper described research that allows humanoid robots to quickly learn new tasks from observing others. We have structured the problem to support fast learning from observation and practice and to quickly adapt to changes. The humanoid has compensated for movements of the playing area, errors in timing, and errors in controlling the paddle to be in the proper state at the time of the hit. This capability has increased the ability of the robot to play a challenging dynamic game. Learning from practice gives the robot an opportunity to discover details that may have been missed during the observation. This research gives us insight into how we can also make other robots more closely approximate human adaptability. One design principle is to break learning problems into as ‘simple’ parts as possible. This subdivision of the control or learning task is limited by the measurements available to the robot, in that the robot needs to perceive the input and output of any learned relationship.

Acknowledgments

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

References

1. C. G. Atkeson, J. G. Hale, F. Pollick, M. Riley, S. Kotosaka, S. Schaal, T. Shibata, G. Tevatia, A. Ude, S. Vijaykumar and M. Kawato, Using humanoid robots to study human behavior, *IEEE Intell. Syst.* **15**(4), 46–55 (2000).
2. C. G. Atkeson, A. W. Moore and S. Schaal, Locally weighted learning, *Artif. Intell. Rev.* **11**, 11–73 (1997).

3. D. C. Bentivegna and C. G. Atkeson, Learning from observation using primitives, in *Proc. IEEE Int. Conf. Robotics and Automation*, Seoul, Korea, 2001.
4. D. C. Bentivegna and C. G. Atkeson, A framework for learning from observation using primitives, in *Proc. RoboCup 2002 Int. Symp.* Fukuoka, Japan, 2002.
5. D. C. Bentivegna, A. Ude, C. G. Atkeson, and G. Cheng, Humanoid robot learning and game playing using PC-based vision, in *Proc. 2002 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Switzerland, 2002.
6. B. Bishop, P. Shirkey and M. Spong, An experimental testbed for intelligent control, *American Control Conf.*, Seattle, 1995.
7. A. Fod, M. Mataric and O. Jenkins, Automated derivation of primitives for movement classification, in *1st IEEE-RAS Int. Conf. Humanoid Robotics (Humanoids-2000)* (MIT, Cambridge, MA, 2000).
8. W. Iba, Learning to classify observed motor behavior, in *Proc. Int. Joint Conf. Artificial Intelligence*, 1991, pp. 732–738.
9. G. Kaminka, M. Fidanboylu, A. Chang and M. Veloso, Learning the sequential coordinated behavior of teams from observation, in *Proc. 2002 Int. RoboCup Symp.*, 2002.
10. A. McGovern and A. G. Barto, Automatic discovery of subgoals in reinforcement learning using diverse density, in *Proc. 18th Int. Conf. Machine Learning*, 2001.
11. J. Park, C. B. Partridge and M. W. Spong, Neural network based state prediction for strategy planning of an air hockey robot, *J. Robot. Syst.* **18**(4), 187–196 (2001).
12. C. B. Partridge and M. W. Spong, Control of planar rigid body sliding with impacts and friction, in *Int. J. Robot. Res.* **19**(4), 336–348 (2000).
13. S. Schaal, C. Atkeson and S. Vijayakumar, Scalable locally weighted statistical techniques for real time robot learning, *Appl. Intell. (Special issue on Scalable Robotic Applications of Neural Networks)* **17**(1), 49–60 (2002).
14. A. Ude and C. G. Atkeson, Real-time visual system for interaction with a humanoid robot, in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Maui, Hawaii, October/November 2001, pp. 746–751.
15. S. Vijayakumar, A. D'Souza, T. Shibata, J. Conradt and S. Schaal, Statistical learning for humanoid robots, *Auton. Robot.* **12**(1), 55–69 (2002).
16. S. Vijayakumar and S. Schaal, Locally weighted projection regression: An $O(n)$ algorithm for incremental real time learning in high dimensional spaces, in *Proc. 17th Int. Conf. Machine Learning (ICML 2000)*, Stanford, CA, 2000.
17. D. M. Wolpert and M. Kawato, Multiple paired forward and inverse models for motor control, *Neural Networks* **11**(7–8), 1317–1329 (1998).
18. Z. Zhang, A flexible new technique for camera calibration, Technical Report MSR-TR-98–71, Microsoft Research, Microsoft Corporation, Redmond, Washington, December 1998.

Darrin C. Bentivegna received his M.S. degree in Space Systems from the Florida Institute of Technology in 1995 and his Ph.D. degree in Computer Science from the Georgia Institute of Technology in 2004. From 1980 to 1995, he served in the US Navy in the Fleet Ballistic Missile Submarine community. He is now a Visiting Researcher in the Department of Humanoid Robotics and Computational Neuroscience at the ATR Computational Neuroscience Laboratories, Kyoto, Japan. His primary research interest is in understanding methods that can give robots human-like intelligence and abilities.

Christopher G. Atkeson is an Associate Professor in the Robotics Institute and Human-Computer Interaction Institute at CMU. He received his M.S. degree in Applied Mathematics (Computer Science) from Harvard University and his Ph.D. degree in Brain and Cognitive Science from MIT. Chris joined the MIT faculty in 1986, moved to the Georgia Institute of Technology College of Computing in 1994, and moved to CMU in 2000.

Aleš Ude studied applied mathematics at the University of Ljubljana, Slovenia, and computer science at the University of Karlsruhe, Germany, where he received a doctoral degree in 1996. From 1998 to 2000, he was an STA fellow in the Kawato Dynamic Brain Project, ERATO, JST. Currently he holds a research position at the Jožef Stefan Institute, Ljubljana, Slovenia, and is also associated with the ATR Computational Neuroscience Laboratories, Kyoto, Japan. His research focuses on humanoid robot vision and visual perception of human activity.

Gordon Cheng is the head of the Department of Humanoid Robotics and Computational Neuroscience, ATR Computational Neuroscience Laboratories, Kyoto, Japan. He is also the Group Leader for the newly initiated JST International Cooperative Research Project (ICORP), Computational Brain. Before taking up these positions, he held fellowships from the Center of Excellence (COE), Science, and Technology Agency (STA) of Japan. Both of these fellowships were taken at the Humanoid Interaction Laboratory, Intelligent Systems Division at the ElectroTechnical Laboratory (ETL), Japan. At ETL he played a major role in developing a completely integrated humanoid robotics system. He received a Ph.D. in Systems Engineering from the Department of Systems Engineering, The Australian National University, and Bachelor and Master degrees in Computer Science from the University of Wollongong, Australia. His research interests include humanoid robotics, biomimetic of human vision, computational neuroscience of vision, action understanding, human–robot interaction, active vision, mobile robot navigation and object-oriented software construction.