Humanoid Robot Learning and Game Playing Using PC-Based Vision

Darrin C. Bentivegna[1,4], Aleš Ude[1,3], Christopher G. Atkeson[1,2], and Gordon Cheng[1]

[1] *ATR Human Information Science Laboratories, Dept. 3, CyberHuman Project, Kyoto, Japan*
[2] *Carnegie Mellon University, Robotics Institute, Pittsburgh, PA, USA*
[3] *Jožef Stefan Institute, Department of Automatics, Biocybernetics and Robotics, Ljubljana, Slovenia*
[4] *Georgia Institute of Technology, College of Computing, Atlanta, GA, USA*

Abstract

This paper describes humanoid robot learning from observation and game playing using information provided by a real-time PC-based vision system. To cope with extremely fast motions that arise in the environment, a visual system capable of perceiving the motion of several objects at 60 fields per second was developed. We have designed a suitable error recovery scheme for our vision system to ensure successful game playing over longer periods of time. To increase the learning rate of the robot it is given domain knowledge in the form of primitives. The robot learns how to perform primitives from data collected while observing a human. The robot control system and primitive use strategy are also explained.

1 Introduction

We are interested in learning humanoid robot behaviors by observing how people perform a task. The ability to observe humans and their actions using a set of cameras mounted on a humanoid robot's head is a necessary prerequisite towards this end. We would greatly benefit from a system that can detect and track multiple objects from images acquired, from moving cameras, at high frame (field) rates, e. g. at 60 Hz, which is the highest rate we can get from a standard, interlaced NTSC camera.

Air hockey was chosen as an environment in which to conduct this research for many reasons. It can be set up within a normal size laboratory and a software version can be created that allows a person to play using only a mouse. Currently we are only concerned with the movements of the objects in the plane of the board. Thus the sensing needs are simplified because information from only one camera is sufficient to play the game.

At each time step, a robot must at least know the position of the puck on the table. Its performance could be improved if it could also take into consideration the position of the opponent's paddle and the position of its own paddle. It is very important that this data is acquired at a high enough frequency, e. g. at 60 Hz, otherwise it is very difficult to predict accurate puck trajectories in time to command the robot hand motion necessary to hit the puck or to prevent the opponent from scoring a goal.

The next section briefly describes the framework that uses predefined primitives in which this research was performed. More details on the framework can be found in [2] and [3]. This paper focuses more on a recently developed PC-based vision system, control of a high degree of freedom humanoid robot, and the coupling of the vision data to the control system.

2 Learning from Observation using Primitives

It is our hope that primitives can be used to reduce the dimensionality of the learning problem [1, 7]. 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. It is often possible to break a primitive up further into smaller primitives.

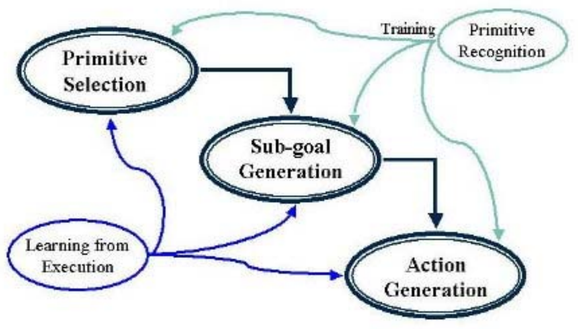

Figure 1: *Framework for learning from observation using primitives.*

2.1 Strategy for Primitive Use

Figure 1 shows our framework designed for conducting research in learning from observation using primitives. Currently, a human, using domain knowledge, designs

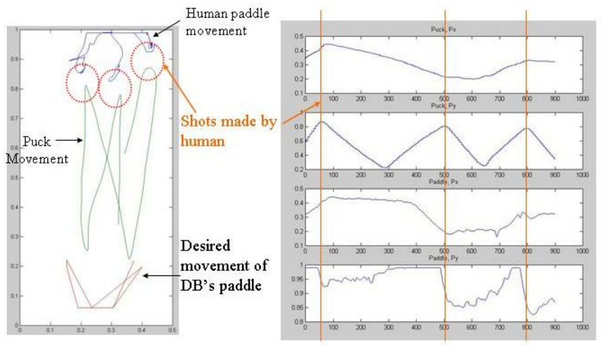

Figure 2: *A small segment of data captured while a human operates in the air hockey environment. The graph on the left shows 2D, x and y, traces of the paddle and puck. On the right this same data is shown plotted against time.*

the candidate primitives that are to be used. The primitive recognition module segments the observed behavior into the chosen primitives. This segmented data is then used to provide the training data for primitive selection, subgoal generation, and action generation. The primitive selection module provides the agent with the primitive type to perform for the observed environment state. The desired outcome, or goal, of performing that primitive type is specified by the subgoal 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.

After the agent has obtained initial training from observing human performance, it should then increase its skill at that task through practice. Using only the four modules mentioned above, the only high-level goal of the robot is to perform like the teacher. The only encoding of the goal of the entire task is in the implicit encoding in the observed primitives. The learning from execution module contains additional domain information needed to evaluate the performance of the system toward obtaining a high level task objective. This information can then be used to update the modules and improve performance beyond that of a teacher.

Using this framework, DB has learned a shot taking behavior from observing a human player. DB decides on the type of shot that will be attempted, the position the puck will be hit, and the pucks desired velocity after it is hit, from observed data. We are currently devising other ways to evaluate the performance of DB in a quantitative way such as comparing the desired hit outcome with the actual outcome.

2.2 Air Hockey Primitives

As explained above, human domain knowledge was used to define a set of primitives to work with. The full list of primitives currently being explored in the air hockey environment is:

- Left Hit: the player hits the puck and it hits the left wall and then travels toward the opponent's goal.
- Straight Hit: the player hits the puck and it travels toward the opponent's goal without hitting the side walls.
- Right Hit: the player hits the puck and it hits the right wall and then travels toward the opponent's goal.
- Block: the player deliberately does not hit the puck but instead moves into a blocking position to prevent the puck from entering their goal.
- Prepare: movements made while the puck is on the opposite side from the player. The player is either preparing for a shot, or preparing to defend their goal.
- Setup: the puck is moving very slow within reach and the player moves to a setup position in preparation to make an accurate shot.

The observed data must first be segmented into the above primitives. To accomplish this, critical events, which are easily observable occurrences, are used. An example of a critical event in air hockey is a puck collision. During a collision the ball speed and direction change rapidly. A small portion of data that has been collected while a human played against the humanoid robot is shown in Figure 2. From this data it can be seen that the puck-paddle hit locations occur when the puck and paddle are within hitting range and there is a significant change in the puck's velocity.

3 Perceiving the Primitives

In order to perceive the primitives and to play air hockey, the robot must be able to sense object locations in the environment in real-time. The task is made more difficult because the robot uses its own eyes to play the game. This causes extremely fast image motions due to the combined real world and camera motions.

3.1 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 know that such a mapping can be modeled by a 3x3 homography, which is defined up to a scale factor and thus has 8 degrees of freedom [11]. 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; 4 fixed points on the board for calibration, puck, and both paddles. To make the recalibration more accurate and the tracking process more robust, it is desirable to make use of more than 4 points to recalibrate the system.

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

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

where $\boldsymbol{x}_i(t) = [x_i(t), y_i(t), 1]^T$ are the known positions of the markers on the hockey board (we measure them by hand before starting the game) 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

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

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 a 9×9 matrix $\boldsymbol{A}^T(t)\boldsymbol{A}(t)$ [11]. Alternatively, one could solve Eq. (2) directly by setting one of the parameters, typically $h_{3,3}(t)$, to 1.

3.2 Visual Processing

The core of DB's visual system is a probabilistic tracker that uses color and shape information to find relevant objects in the scene. In this system, the observed environment is represented by a number of mutually independent random processes (blobs). Each entity to be tracked is represented by one process. We also introduced an additional outlier process that models everything not captured by other processes.

A Bayesian approach is used to evaluate the probability that a pixel was generated by one of the modeled processes. To model color probabilities, we observed that there were quite significant variations in lighting conditions on the hockey board. For example, the puck appears much brighter when it approaches the human player than on the other side of the board (see Fig. 3). This made it impossible to model colors in the RGB space. However, 2-D HSV space color models, in which we ignored the brightness component, turned out to be sufficiently stable to model color variations by a covariance matrix of the Gaussian probability distribution.

The puck, paddles, and markers on the edge of the board all have a roughly ellipsoidal shape and their 2-D projected shapes can be approximated by the center of the projection image and by the covariance matrix of pixels contained in the projection. Thus the shape part of the probability that a pixel belongs to one of the blobs can also be characterized by a Gaussian distribution. However, since the image sizes of the objects vary as they, and the robot, move, we cannot assume that the underlying covariance matrices are constant.

The calculation of the objects' position and shape is based on an EM (expectation-maximization) algorithm. The color models are kept constant in the current version of the hockey game tracker and are learned off-line. See [9] for more details about the implemented algorithm.

To achieve real-time operation of the system, we employed techniques such as multiscale representations, windowing, masking and parallel processing on a dual processor PC. We also developed a special technique based on affine warping to subsample images selectively only in those regions that contain too much data for real-time operation [10]. On a dual processor 2 GHz Pentium 4 PC, the system needs 9.5 milliseconds to track 9 objects (six markers on the board, both paddles, and the puck) at 60 Hz with the window size for affine warping set to 61×61 pixels for all objects and with the number of EM iterations limited to 2.

3.3 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 a vision system. Since it may not be possible to completely avoid tracking failures, we designed a specialized error

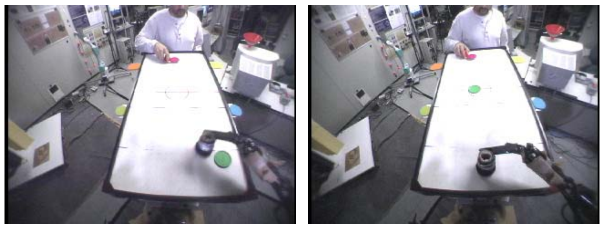

Figure 3: *Brightness variations in the appearance of the puck*

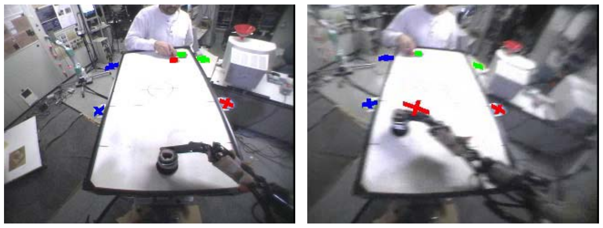

Figure 4: *The view from the robot's eyes with the tracked objects marked. The puck is totally occluded in the right image. The right image is also blurred due to the movement of the robot.*

recovery scheme that ensures the successful operation of the vision system over longer periods of time.

Most of the blobs never come close to each other in the air hockey game. There are three groups: the blobs fixed at the edge of the board, the two blobs associated with the paddles used for hitting the puck and the puck itself. Since the shape-related probability distributions assign larger values to the neighboring pixels, there is no need for blobs from within these groups to have different colors. We can thus in principle use only three different colors to identify the objects of interest, although in practice it might still be advantageous to use more colors.

Recovering from Occlusions. There are a few situations when our tracker might fail. The most common problem is that a robot arm sometimes completely occludes the puck and it is therefore not possible to locate it, see Fig. 4. Such cases are detected by calculating the sum of probabilities normalized by a number of pixels considered in the calculations. If this sum becomes very small (less than 10% of the expected area covered by the puck within the region of interest) or even zero, then the calculation of the corresponding blob location becomes unstable and the tracker is assumed to have failed. When this happens, the tracker keeps looking for a puck for half of a second by starting the EM iterations using the latest detected location as an initial value.

After half of a second it is considered unlikely that the puck would still be situated 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 a contact with the puck in the future and the other one in the banded region along the robot side of the board where the puck might still be situated in the case of occlusion. The second process randomly generates initial puck positions within the search region and thus ensures that the 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, that couldn't be carried out at 60 Hz. The success of one of the search processes terminates them both.

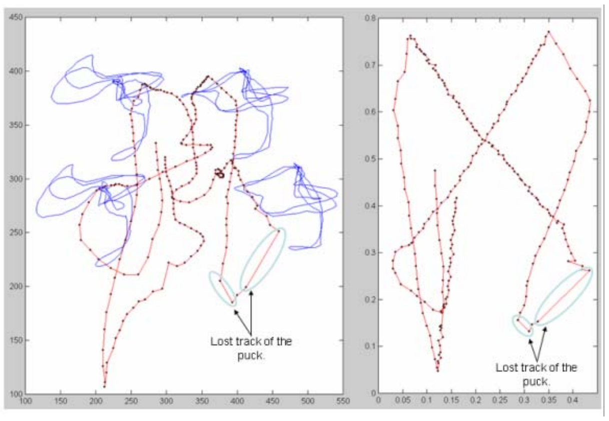

Figure 5: *The left graph shows the raw vision cordinates of four objects placed at known locations and the moving puck. On the right is the computed position of the puck using the information shown to the left. The circled segments are where the vision system lost track of the puck.*

Figure 5 shows the results of the visual tracking system during approximately 4.2 seconds of game play. During this interval the set markers are always being tracked, but on two occasions the puck was lost 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 100msec before the puck is hit.

Recovering from Tracking Errors. It does happen occasionally that one of the side blobs used for recalibration is lost by the tracker. This is most often caused by a 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 can't see at least 4 side blobs, the recalibration cannot be carried out and the robot can't play the game. But 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 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 sides of the hockey board. Thus if one of the paddles is lost, we can start a search process in a suitable banded region along the side of the board. The paddle positions are randomly generated in this region and used as initial values for the EM iteration until the paddle is found. This is similar to the second search process when looking for a puck.

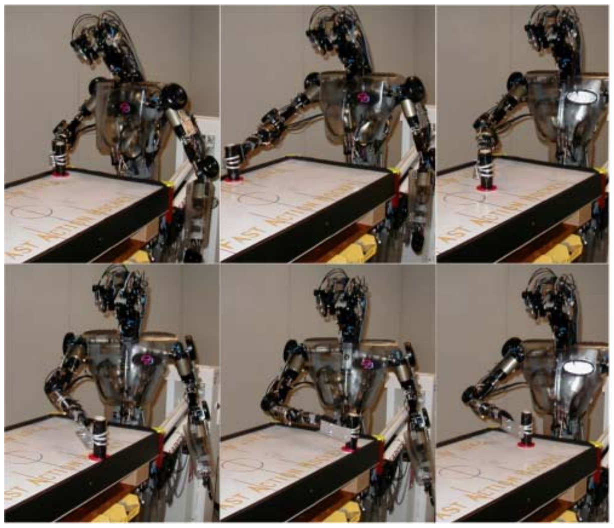

Figure 6: *The six given configurations of the robot used to compute all enclosed configurations.*

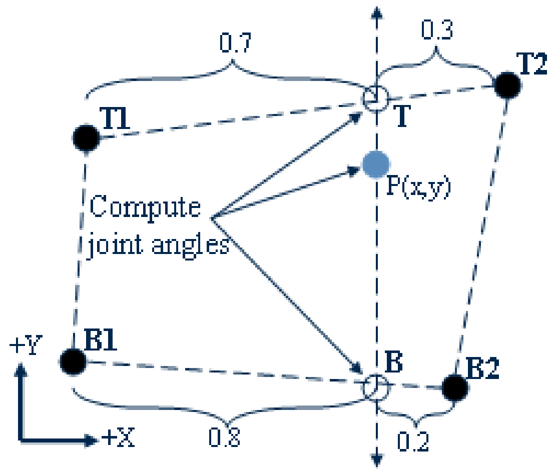

Figure 7: *An example of using four given corner configurations to compute a configuration within the polygon.*

4 Positioning the Humanoid Robot

At first it may appear that the robot only needs to use one arm to play air hockey. Using only the arm simplifies the system in that there is less movement of the head and therefore the cameras. It also eliminates many degrees of freedom. Our first implementation used this scheme and we found it to be extremely impractical. It severely limits the moves the robot can make and the area that can be reached on the board. Moving the torso increases the robot's abilities and extends its reach. But besides adding three more degrees of freedom to control, it causes the head to move in ways that are unnatural looking and the air hockey board to not always be fully within view. Therefore the head and eyes must also be controlled to ensure the board is always within the field of view of the robot. The total degrees of freedom needed for this task is 17. The following joints are used in this approach: shoulder (2 joints), arm rotation, elbow, wrist rotation, hand (2 joints), waist (3 joints), head (3 joints), and eyes (2 joints each eye, pan and tilt). Using all the joints above, except for the head and eyes, the robot must be positioned so that the paddle is flat on the board and moves smoothly from one location to another. The other joints are used to position the head and eyes so that the entire board is in view at all times.

We have manually positioned the robot in several positions on the board while maintaining these constraints, figure 6. To get joint angles for any desired puck position, we interpolate using the four surrounding training positions and use an algorithm similar to that used in graphics for texture mapping [4]. This approach allows us to solve the inverse kinematics of the robot with extreme redundancy in a simple way.

Figure 7 shows an example of using the four training positions *B1*, *B2*, *T1*, and *T2* to compute the configuration needed to place the paddle at the desired position of $P(x, y)$. The four set positions, *B1*, *B2*, *T1*, and *T2*, surrounding the desired position $P(x, y)$ define a polygon with Y up and X to the right. A vertical line is drawn at $P(x)$ and points are defined at the locations where this line intersects the top and bottom of the polygon, points *T* and *B*. The joint angles are computed for the bottom-intersect location, *B*, by using the two joint configurations associated with the two bottom set locations, *B1* and*B2*. *B1* and *B2* configurations are weighted averaged together with each weight computed from the percentage of the point *B* from *B1* and *B2*. As shown in the example in figure 7, the joint angles computed at point *B* will be 0.8 times the angles specified by point *B2* plus 0.2 times the angles specified at point *B1*. The configuration for the top point, *T*, is computed in the same way using the top two corners of the polygon, *T1* and *T2*. The configuration to place the paddle at the locations *B* and *T* now exist and are used to compute the configuration needed to place the paddle the location $P(x, y)$. The percentage of the distance that $P(y)$ is from the bottom intersect point to the top intersect point is computed and is used as a weight to average the two computed joint configurations of *B* and *T*.

The robot configuration needed to place the paddle at any location within the training examples can now be computed. The robot is moved by specifying a desired board location along with a desired movement speed. A straight line trajectory is then computed from the present location to the desired location. The robot is commanded to a position 420 times per second and therefore there must be a point on the trajectory at each 1/420 second interval. The position of the trajectory points that the paddle will move through is computed using a fifth order polynomial equation. The robot is moved through the trajectory points and at each point a configuration for the robot is computed as described earlier.

Currently, a hit is made only when the robot is not moving and the desired end velocity of a move is set to zero. With this strategy the maximum velocity of the robot's move, and therefore the paddle movement, occurs at the halfway point of the straight line trajectory.

5 Future Work

In term of vision processing we are looking toward providing an active stereo tracking system utilizing the eyes and head of the robot. Our goal is to provide a better field of view for the robot, allowing better estimation of the puck and the paddle of the opponent via active tracking. Currently, all of the vision processing is performed with only one PC. For a future extension to our system, and to provide robustness and flexibility, we are currently building a Beowulf PC Clusters system [8] to support additional cue processing, and stereo vision processing, as well as control.

Currently much of the causal information of the primitives is pre-given by a domain expert. We believe a better way of determining these primitives would be through a process of segmentation, which can detect the causality of an observed action. In doing so, we will be able to generalize the abilities to extract primitives, thus allowing us to move across domains. The research of Dillmann et al. [5] and Fod et al. [6] give some insights into automatically segmenting observed data into primitives.

6 Conclusions

We have developed efficient methods that allow a humanoid robot to learn and play the game of air hockey from the observation of human performance. Our approach is based on the description of the game in terms of motion primitives that can be combined to describe the complete task. We have shown that it is possible to play an air hockey game using the robot's eyes and PC-based vision that can extract the necessary information at 60 Hz from a standard NTSC video. Using our own vision system enabled us to design an effective error recovery strategy, which would be difficult to implement with a closed commercial tracking system.

We have shown how to identify motion primitives in the data collected by the robot's eyes and how to extract the parameters needed for the performance of the primitives. Finally, efficient methods have been described that allow us to select proper movement primitives during the actual air hockey game and to plan the high degree of freedom joint space trajectories needed to replicate the selected movement primitives with the robot.

The overall framework described in section 2 provides much flexibility in conducting this research. The ability to use the 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 within this framework. Techniques described in this paper provide the basis for our research on learning from observation using primitives.

Acknowledgments

Support for all authors was provided by ATR Human Information Science Laboratories, Department 3, and the Communications Research Laboratory (CRL). It was also supported in part by the National Science Foundation Award IIS-9711770.

References

[1] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1998.

[2] 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.

[3] D. C. Bentivegna and C. G. Atkeson. Learning how to behave from observing others. In *SAB'02-Workshop on Motor Control in Humans and Robots: on the interplay of real brains and artificial devices.*, Edinburgh, UK, 2002.

[4] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, Oct. 1976.

[5] R. Dillmann, M. Kaiser, and A. Ude. Acquisition of elementary robot skills from human demonstration. In *International Symposium on Intelligent Robotics Systems*, Pisa, Italy, 1995.

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

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

[8] T. L. Sterling, J. Salmon, D. J. Becker, and D. F. Savarse. *How to Build a Beowulf: A guide to the implementation and Application of PC Clusters*. The MIT Press, 1999.

[9] 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*, pages 746–751, Maui, Hawaii, October/November 2001.

[10] A. Ude and C. G. Atkeson. Probabilistic detection and tracking at high frame rates using affine warping. In *Proc. Int. Conf. Pattern Recognition*, Québec City, Canada, August 2002.

[11] Z. Zhang. A flexible new technique for camera calibration. Technical Report MSR-TR-98-71, Microsoft Research, Microsoft Corporation, Redmond, Washington, December 1998.