Available online at www.sciencedirect.com

SCIENCE DIRECT®

Robotics and Autonomous Systems 47 (2004) 163–169

Robotics and Autonomous Systems

www.elsevier.com/locate/robot

Learning tasks from observation and practice

Darrin C. Bentivegna[a,b,*], Christopher G. Atkeson[a,c], Gordon Cheng[a]

[a] *ATR Computational Neuroscience Laboratories, Department of Humanoid Robotics and Computational Neuroscience, Kyoto, Japan*
[b] *Georgia Institute of Technology, College of Computing, Atlanta, GA, USA*
[c] *Carnegie Mellon University, Robotics Institute, Pittsburgh, PA, USA*

Abstract

This paper presents a framework that gives robots the ability to initially learn a task behavior from observing others. The framework includes a method for the robots to increase performance while operating in the task environment. We demonstrate this approach applied to air hockey and the marble maze task. Our robots initially learn to perform the tasks using learning from observation, and then increase their performance through practice.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Learning from observation; Movement primitives; Imitation; Locally weighted learning; Action recognition

1. Introduction

We are exploring how primitives, small units of behavior, can speed up robot learning and enable robots to learn difficult dynamic tasks in reasonable amounts of time. In this paper, we describe work on learning from observation and learning from practice in air hockey and on a tilt maze task. This paper discusses our research strategy, results, and open issues.

Primitives are units of behavior above the level of motor or muscle commands. There have been many proposals for such units of behavior in neuroscience, psychology, robotics, artificial intelligence, and machine learning [1–5].

We have used two tasks to develop our thinking, air hockey and marble maze, Figs. 1 and 2. We have developed versions of these games to be played by simulated agents and by robots. Although hardware implementations necessarily include real-world effects, we can collect useful training data from the simulations without the cost of running the full robot setups, and can perform more repeatable and controllable experiments in simulation.

In the air hockey task a player tries to hit the puck into the opponent's goal and also tries to prevent the puck from entering their own goal. In the marble maze task, a player tilts a maze to roll a marble to a goal, avoiding hazards such as holes. The manually defined library of primitives for the air hockey task is:

- *Straight Shot*. A player hits the puck and the puck hits a wall and then 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.

* Corresponding author.
E-mail address: darrin@atr.co.jp (D.C. Bentivegna).

0921-8890/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2004.03.010

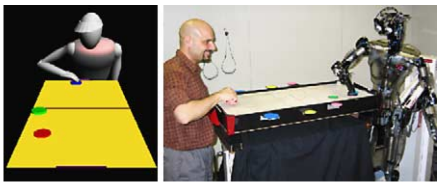

Fig. 1. The software air hockey game on the left and air hockey playing with a humanoid robot on the right.

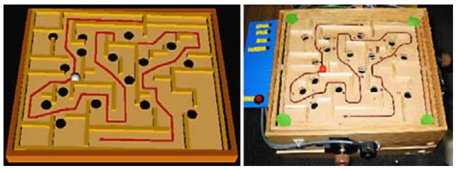

Fig. 2. Software and hardware marble maze environments.

- *Idle*. A player rests their paddle while the puck is on the opponent's side.

The manually defined library of primitives for the marble maze task is (Fig. 3):

- *Roll To Corner*. The marble rolls along a wall and stops in a corner.
- *Roll Off Wall*. The ball marble rolls along a wall and then rolls off the end.
- *Guide*. The marble rolls without touching a wall.
- *Roll From Wall*. The marble rolls on a wall and then rolls away from it.
- *Leave Corner*. The marble is captured in a corner and then the board is positioned in preparation to move the marble from the corner location.

In both hardware and simulation versions of these tasks object positions can be recorded as a human plays the game. The hardware air hockey task has been created using a humanoid robot, Fig. 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 hardware maze has two motors that control the tilt of the board. A human can control the motors using knobs that are connected to encoders. The computer receives the encoder signals and creates the proper motor commands. A level sensor is connected to the board and is used to initially level the board. After it is level, its orientation is measured using two encoders. The real-world agents are equipped with a vision system which estimates the position of the objects in the environment at 60 Hz [6].

2. Our framework

Our behavioral framework, Fig. 4, has three main parts when operating in the environment using observed information processed by the primitive recognition module [7]. The first part, *primitive selection*, is a classifier that uses the current location in state space to choose a behavioral primitive to execute. Execution of the current primitive may be terminated when a new primitive type is selected, after a specified time interval, or when the current primitive indicates completion. For tasks that have a set sequence of actions, such as dancing, primitive selection can be done by specifying the sequence of primitives to be performed [8,9]. Tung and Kak [10] show how a planning system can be used to specify the primitive execution sequence in an environment where objects are not moving during training and there is a high probability

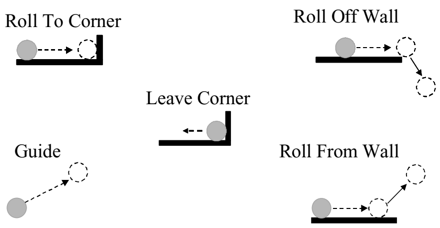

Fig. 3. Primitives being explored in the tilt maze environment.

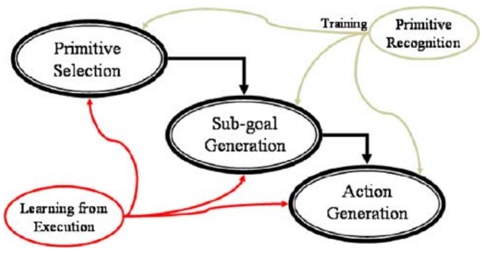

Fig. 4. Our framework.

of successful performance of the primitives. Robots operating in a similar static environment have learned a primitive execution sequence from observed data [11]. We are interested in dynamic tasks where primitive execution sometimes fails, other agents interfere, and a fixed sequence of primitives is not adequate.

The second part of our framework, *sub-goal generation*, is a module that specifies a goal of performing the behavioral primitive chosen. For the *Roll Off Wall* primitive, for example, this module specifies the velocity at which the marble will roll off the wall and the tilt of the board at the end of the primitive execution. The third part of our framework, *action generation*, specifies the actuator commands to achieve a behavioral sub-goal.

In learning from only observation, the robot's goal is to behave like the teacher. No knowledge of higher level goals or how to improve its performance autonomously is needed. In order to learn from practice, there must be additional domain knowledge that provides information needed to evaluate progress towards task objectives. The learning from execution module contains that knowledge and provides feedback to the other modules so their behavior can be changed.

3. Choosing primitives, generating sub-goals, and performing the action

It is the responsibility of the primitive selection module, Fig. 4, to choose the primitive type, based on the current state and prior observations of primitives that have been executed. In our implementation, during training the context or state in which the human has performed each primitive is extracted from the observed data, and during execution is used by a nearest neighbor lookup process to find the most appropriate primitive type as follows.

A database is created from the observed data that contains states of the environment and corresponding primitive types. A lookup is performed on this database to find the states that are closest to the query state. The distance of each data point from the query point is computed as $d(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_j w_j \cdot (\mathbf{x}_j - \mathbf{q}_j)^2}$, where $\mathbf{x}$ and $\mathbf{q}$ are the locations of the data point and the query point in state space, and the vector w allows each dimension to be weighted differently. A query to the database is the current state of the environment. In the marble maze task for example, the query consist of the marble's position and velocity, and board tilt angles (Mx, My, $\dot{M}x$, $\dot{M}y$, Bx, By).

A pure nearest neighbor lookup scheme would use the closest point to select the primitive type. The data point also contains the observed outcome of the human's performance of that primitive, which can be used as the desired sub-goal. An alternate approach is to use several nearby points, and implement some sort of voting scheme. The primitive type (a discrete choice) can be chosen by selecting the primitive type that occurs most often within the closest N data points, for example. We are currently selecting the primitive type indicated by the nearest data point.

3.1. Computing the desired sub-goal

Once the primitive type has been chosen the sub-goal can be computed. It is important to first choose the primitive type because the sub-goals of different primitive types cannot be combined. For example, it would not make sense to use the sub-goal of the *Roll Off Wall* primitive with the *Roll Into Corner* primitive. The *Roll Into Corner* primitive will be expecting a corner for the marble to land in as a sub-goal location and the *Roll Off Wall* primitive will be specifying a sub-goal location at the end of a wall.

The algorithm used in the primitive selection module identifies multiple data points that are in the vicinity of the current state and their distance to the query point. The closest data point also contains the observed outcome of the human's performance of that primitive. This information can be used as the desired sub-goal. A more robust approach is to use the n closest points that indicate the selected primitive type to compute the sub-goal.

The outcomes of the returned points are used to compute the sub-goal using a locally weighted learning (LWL) model [12]. A kernel function, $K(d) = \exp^{-\alpha d^2}$, uses the distance d, as shown in the previous section, to compute the weight of each data point. α defines the range over which generalization is performed [12] discusses the effect of α and the use of other kernel functions on the weighting of the data points. The output components needed at the query point are computed using the equation $y(\mathbf{q}) = \sum y_i K(d(\mathbf{x}_i, \mathbf{q})) / \sum K(d(\mathbf{x}_i, \mathbf{q}))$ where i ranges from

1 to n, the number of closest data points of the same type. 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. The values of the vector w and α are set globally and were chosen by trial and error.

The methods being used for primitive type selection and sub-goal generation require all the data points to be visited. Because the agents operate in dynamic environments they must make decisions quickly. By having the sub-goal generation module reuse the information computed by the primitive selection module helps to minimize the time needed to make decisions.

3.2. *Performing the primitive*

The action generation modules contain a policy to control the robot. There is a separate action generation module for each type of primitive. These modules can use any algorithm that provides the needed actuator commands to obtain the desired sub-goal. Various methods have been explored including idealized models based on physics, neural networks, and kernel regression techniques [12].

The policy used within the action generation module can be local and general. This will allow the module to be used at multiple locations within the task domain and, if properly formatted, can also be used in similar domains. The environment state and the policy's specified actions must be transformed as needed. Information needed by the action generation modules is broken into small units that encode environment or action outcome information.

4. Practising to increase performance

There are many things these robots can learn while they are performing the task. The learning from practice module contains the information needed to evaluate the performance of the agents towards completing the task. This information is used to update primitive selection, and sub-goal and action generation. A tough question is where the task criterion comes from. Ideally, it should be learned from observation. The learner should infer the intent of the teacher. This is very difficult, and we defer addressing this question by manually specifying a task criterion.

If the learning problem is structured appropriately, the agents can increase their knowledge of the environment. This knowledge can be used by the agent to update its action generation modules to increase its performance of the individual primitives. The agents can also learn about the effectiveness of choosing particular actions.

5. Discussion and results

Agents were created to operate in the testing environments using the 'Learning from Observation using Primitives' framework. After the agents observe a human perform the task they performed the task using only the observed information. Therefore they did not learn beyond what was observed and the policy used by each module remained fixed. These agents followed four basic steps: (1) observe the state of the environment; (2) decide what primitive to perform; (3) compute the parameters to use with the selected primitive type; (4) perform the primitive until it has terminated. Steps 2 through 4 use the information obtained from observing the human. The agent's intention is to act as the human did, or as the human would, for the observed state. But if the agent incorrectly predicts the human's action, or cannot correctly perform the chosen action, it has no way of knowing if the outcome is desirable for completing the task.

The agents are then given the ability to increase there performance while practising. The air hockey agent is given the ability to learn about the environment from observing its own performance. This knowledge is used in the action generation modules to increase it performance at making shots. The tilt maze agents have the ability to change their primitive selection and sub-goal generation behavior using the information provided by the learning from execution module. This section presents the results of these agents operating in the testing environments with and without learning from practice.

5.1. *Model learning in air hockey*

While watching a human play simulated air hockey for approximately ten minutes, an agent observed 44 straight shots and 108 bank shots. This information

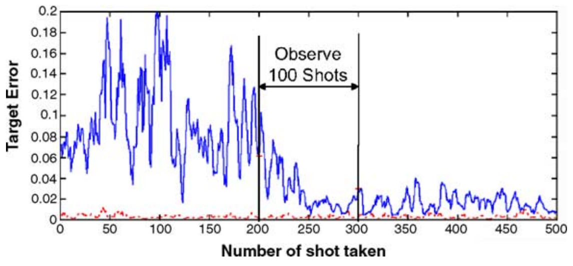

Fig. 5. Absolute error in reaching the target location during 500 straight shots made by the agent in the air hockey simulator.

was used to create environment models that are used by the action generation modules of the *Bank Shot* and *Straight Shot* primitives.

The solid line in Fig. 5 shows the result of the agent making 500 straight shots in the simulator. This figure 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.

For the first 200 shots the agent is using models created from observing the human's shots. The width of the goal is 20 cm and from Fig. 5 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 model.

After making 200 straight shots using the models learned from observing the human, the agent then observed 100 of its own shots while practising (shots 201–300 in Fig. 5). 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. Fig. 5 shows the result of using these newly trained models for the shots from 301 to 500.

5.2. Learning primitive selection and sub-goal generation in the marble maze task

After observing three games played by a human, the hardware marble maze agent played ten consecutive games. During those games the marble falls into holes six times and completed the maze four times. The agent in the software environment also observed the human perform the task three times. If this agent falls into a hole or gets stuck, it is penalized 10 s and play continues just past the failure point. The human performed the task in the simulator and completed the maze in 23.3, 30.3, and 32.7 s, never fell into a hole and was never penalized for not making progress. The top solid line on the graph in Fig. 6 shows the performance of the agent during 30 trials in the simulation environment using the observed information. Each trial consist of playing three consecutive games and the graph shows the running average time to complete the three games. The skilled human's average of 28.77 s is shown by the bottom solid line in the graph.

The same agents are now given the ability to change their primitive selection and sub-goal generation policy while operating in the environment. The hardware agent completed ten consecutive games without falling into a hole after it practised for 30 games. The dashed line on the graph in Fig. 6 shows the performance of the software agent playing the game with the ability to update its primitive and parameter selection policy. Again the graph shows 30 trials with each trial being the average of three games. This agent immediately decreased the time it takes to complete the maze and

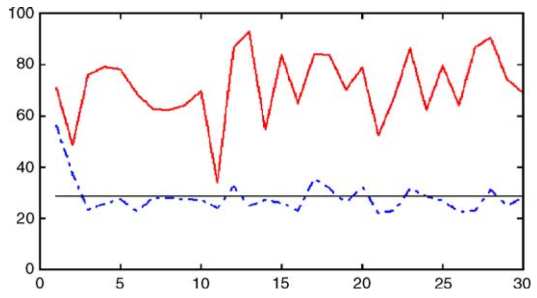

Fig. 6. Performance in the software marble maze. Top solid line: agent using only observed information; dashed line: agent also learning while practising; bottom solid line: average time of the three observed games performed by the human.

its performance is as good or better than that of the observed human's average of 28.77 s.

6. Conclusions

Choosing actions to perform when operating in dynamic environments, such as the environments described in this paper, is a difficult task. Because the state space is large and continuous, expecting to learn entirely from random actions is not realistic. An initial policy can be created using knowledge of primitive actions performed in the environment and information obtained from observing others. Within our research we find that the performance of the initial policy is quite high but there is still room for improvement. This initial policy provides a very good starting point from which to practice to further increase competence at the task.

Our learning from observation using primitives framework described in Section 2 provides flexibility in conducting research in learning from observing others. The framework uses the observed data in a systematic way, and provides the ability to learn while practising. The organization of the data allows lookups to be performed using LWL techniques. Agents using our framework have learned an initial policy to use in the marble maze and air hockey tasks. The agents go on to increase performance while practising the task.

Acknowledgements

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

References

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

[2] R.C. Arkin, Behavior-based Robotics, MIT Press, Cambridge, MA, 1998.

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

[4] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, 1995.

[5] A. Barto, S. Mahadevan, Recent advances in hierarchical reinforcement learning, Discrete Event Systems 13 (2003) 41–77.

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

[7] D.C. Bentivegna, C.G. Atkeson, A framework for learning from observation using primitives, in: Proceedings of the RoboCup 2002 International Symposium, Fukuoka, Japan, 2002.

[8] J.K. Hodgins, W.L. Wooten, D.C. Brogan, J.F. O'Brien, Animating human athletics, Computer Graphics 29, Annual Conference Series, 1995, pp. 71–78.

[9] M.J. Mataric, M. Williamson, J. Demiris, A. Mohan, Behavior-based primitives for articulated control, in: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior (SAB-98), MIT Press, 1998, pp. 165–170.

[10] C. Tung, A. Kak, Automatic learning of assembly tasks using a dataglove system, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 1, 1995.

[11] Y. Kuniyoshi, M. Inaba, H. Inoue, Learning by watching: extracting reusable task knowledge from visual observation of human performance, IEEE Transactions on Robotics and Automation 10 (1994) 799–822.

[12] C.G. Atkeson, A.W. Moore, S. Schaal, Locally weighted learning, Artificial Intelligence Review 11 (1997) 11–73.

Darrin C. Bentivegna is a graduate student in the College of Computing at Georgia Institute of Technology and an Intern Researcher in the Department of Humanoid Robotics and Computational Neuroscience at the ATR Computational Neuroscience Laboratories. He received his MS degree in Space Systems from Florida Institute of Technology. Darrin served in the US Navy for 15 years in the Fleet Ballistic Missile Submarine community. His primary research interest is in understanding methods that can give robots human-like intelligence and abilities.

Chris G. Atkeson is an Associate Professor in the Robotics Institute and Human–Computer Interaction Institute at CMU. He received the MS degree in Applied Mathematics (Computer Science) from Harvard University and the PhD 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.

Gordon Cheng is currently a senior research scientist and head of the Department of Humanoid Robotics and Computational Neuroscience, Computational Neuroscience Laboratories of ATR International. He was a Center of Excellence (COE) and a Science and Technology Agency (STA) Research Fellow; both fellowships were taken while working in the Humanoid Interaction Laboratory, Intelligent Systems Division at the Electrotechnical Laboratory (ETL), Japan. Before coming to Japan he conducted his PhD study at the Department of Systems Engineering, The Australian National University, Australia.