Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

Learning Similar Tasks From Observation and Practice

Darrin C. Bentivegna*‡, Christopher G. Atkeson*‡, and Gordon Cheng*†
*ATR Computational Neuroscience Laboratories,
Department of Humanoid Robotics and Computational Neuroscience, Kyoto, Japan
†Computational Brain Project, ICORP, Japan Science and Technology Agency, Kyoto, Japan
‡Carnegie Mellon University, Robotics Institute, Pittsburgh, PA, USA
(darrin@atr.jp, cga@cmu.edu, gordon@atr.jp)

***Abstract*— This paper presents a case study of learning to select behavioral primitives and generate subgoals from observation and practice. Our approach uses local features to generalize across tasks and global features to learn from practice. We demonstrate this approach applied to the marble maze task. Our robot uses local features to initially learn primitive selection and subgoal generation policies from observing a teacher maneuver a marble through a maze. The robot then uses this information as it tries to traverse another maze, and refines the information during learning from practice.**

Fig. 1. Software and hardware Marble Maze environments.

I. Introduction

Learning can be very expensive for a robot. It is therefore necessary that they make as much use of learned information as possible. One way to do this is by having the ability to use learned information and skills in a variety of similar tasks.

Behavioral primitives have been proposed as a way of speeding up learning and improving generalization and are defined as solutions to small parts of a task that are combined to complete a task [1], [2]. There is an extensive amount of research in programing and teaching robots and simulated agents primitives in manipulation [3], navigation [1], athletics [4], and dance [5] that can generalize to similar tasks. Robots that have learned a primitive execution sequence for manipulation tasks have the ability to generalize to a slightly different initial configuration of the work space [6]. Other systems have been created that support generalization by providing gestural and verbal feedback to the robot to guide the sequence of primitives [7]. We have found that in dynamic tasks, where primitive execution sometimes fails and other agents may interfere, we need more complete policies to tell the agent what action to perform in any state.

In this paper we investigate a method to learn policies for selecting primitives and generating subgoals that will generalize to similar tasks. We will show how local features support generalization, but global features are much better in supporting learning from practicing the same task than local features. We will use the marble maze task as an example. In this task, a player tilts a platform to roll a marble to a goal, avoiding hazards such as holes. The objective is not to find a path to the goal, as the path to the goal is printed on the board, but to perform actions that will quickly move the marble along the path to the goal location. The robot observes a human as they control the board to move the marble through a maze layout. The robot then controls the board to move the marble from the start to goal location on a different layout.

Software and hardware versions of the marble maze task have been created as testbeds (Fig. 1). We created a performance evaluation criterion that measures elapsed time so that the performance of various players can be compared. If the marble falls into a hole or does not make progress for 10 seconds, the marble is moved forward in the maze and the player is given an additional 10 second penalty. We used a manually defined library of primitives so we can focus on learning to *select* primitives and *generate* subgoals from observing a task and practicing a similar task. The manually defined library of primitives for the marble maze task is (Fig. 2):

- **Roll To Corner**: The marble rolls along a wall and stops when it arrives at a corner.
- **Roll Off Wall**: The marble rolls along a wall and then rolls off the end of the wall.
- **Roll From Wall**: The marble rolls along a wall and is then maneuvered off the wall.
- **Guide**: The marble is rolled from one location to another without touching a wall.
- **Corner**: The ball is moved into a corner and the board is then almost leveled, so that the next primitive can move the ball out of the corner.

A. Extending Previous Research

A major focus of our research is to investigate methods that will allow learned information to be reused as much as possible. We created a framework for learning from observation

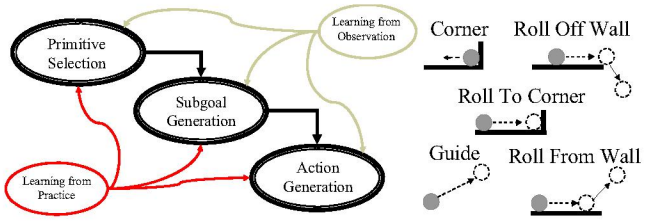

Fig. 2. Left: Our framework. Right: The primitives being explored in the tilt maze environment.

and practice using primitives (Fig. 2). This framework enabled robots to learn how to perform the marble maze task [8], [9] and play air hockey [10]. *Action Generation* was designed to be used at multiple locations within the environment and could also be used in similar environments. The *Primitive Selection* and *Subgoal Generation* modules use a memory-based approach to learning when to use which primitive and with what arguments. The *Learning from Observation* module supports learning from watching others perform a task. When using only observed information the robot has a fixed policy based on the observed information. The *Learning from Practice* module evaluates the robot's performance during task execution and uses a variety of algorithms to improve the skill of the robot.

When selecting a primitive to perform, the robot observes the environment and asks the question, "What primitive did the teacher perform when the marble was at this board location, with the observed velocity and the board at the observed angles?" The answer is the primitive that the teacher performed when they were in the state closest to the observed state. We use the term "global representation" for a state that incorporates board location since the state includes information in global board coordinates. This type of global representation implicitly encodes the board layout and general situation. Because of this implicit encoding of context, the learned primitive selection and subgoal generation policies can not be used on boards with different layouts.

A robot whose goal is to use information in a more general way may ask the question, "What did the teacher do when the marble was in a situation where there is a wall to the left, with a corner ahead, a hole to the right, the marble has a velocity in the direction of desired movement, and the board is rotated to increase the marble's velocity in the desired direction?" If the robot previously observed and recorded the teacher in this situation *and stored the information in an appropriate way* it would be able to answer this question. We will refer to this type as a "local representation" as these representations typically emphasize features spatially local to some object or event, in this case the marble.

The research presented in this paper focuses on the use of local features in direct policy learning. In our previous work we found that global representations supported fast learning from observation and practice on a fixed task. However, generalization to similar tasks was poor. This paper explores how to more effectively generalize by using both global and local representations. Local representations help transfer skill to a new board, and global representations help improve skill with practice on the same board.

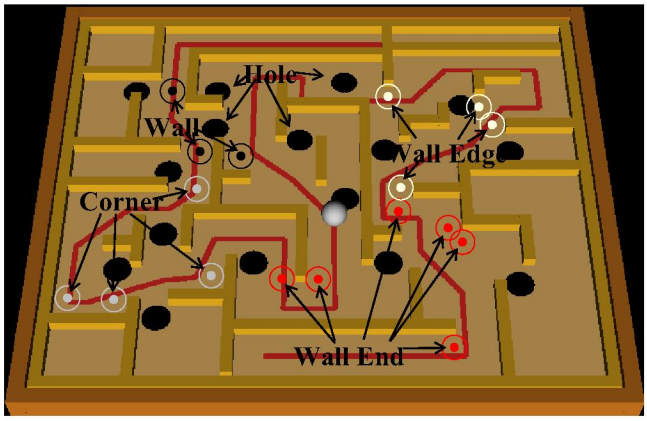

Fig. 3. The location of some of the features in the tilt maze environment.

II. Local Representation

Among many possible options, we designed a local representation for the marble maze that records the features that would be seen if one was at the location of the marble and looked around. The approximate path the marble should take is printed on the board, and we use it to initialize a 2D table of all possible locations the marble can be in with the distance to the goal recorded in each cell. The desired movement direction is computed from any point on the maze by observing the values in the adjacent cells.

Ball movement directions have been discretized into four directions that are parallel and perpendicular to the maze walls: forward (the desired movement direction), reverse, left, and right. For this initial implementation we used features in the four directions along with the ball's velocity and board angles in relationship to the desired movement direction. Fig. 3 shows various locations of the features (holes, walls, corners, wall ends, and wall edges) within the maze, and Fig. 4 and Table I shows the encoding at various locations in the maze. The side directions, left and right, are treated differently than the forward and reverse directions. The *Wall End* feature is only computed for the side directions. If a wall is seen on the sides within 2*BALL_RADIUS, the feature (*Corner*, *Hole*, or *Wall End*) that is along that wall in the direction of desired movement is also recorded.

This local representation is used to store experiences and encode the actions that were taken while observing a player perform the task. For each observed primitive performance, the local state information along with the type of primitive that was performed, the movement of the marble, and the marble's velocity and board angles at the end of the primitive are recorded.

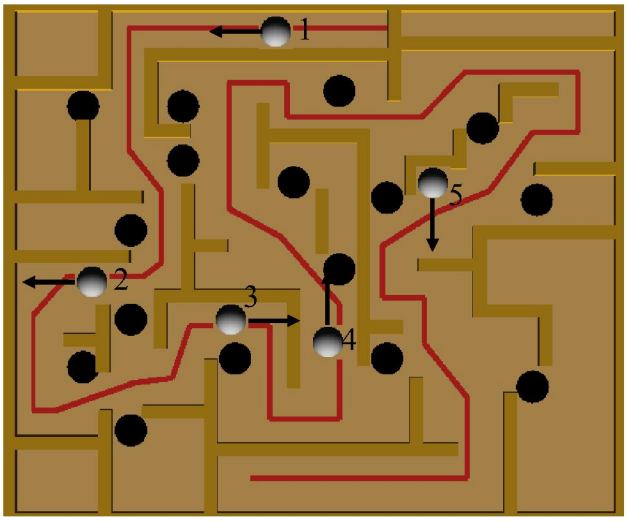

Fig. 4. The marble at various locations within the maze. The arrows show the desired movement direction. Table I lists the board features for each of these positions. The marble's velocity and the angles of the board in relationship to the desired movement direction are also recorded local features.

Pos.	Forward	Left	Reverse	Right
1	Corner	Wall/End	Corner	Wall/Corner
2	Corner	Wall Edge	Corner	Wall/Corner
3	Corner	Wall/Corner	Corner	Hole
4	Hole	Wall/End	Wall	Wall/Hole
5	Corner	Hole	Corner	Wall/End

TABLE I
FEATURES RECORDED FOR THE MARBLE IN THE POSITIONS SHOWN IN FIG. 4.

A. Computing Behavior

The robot uses memory-based approaches to computing behavior, looking up past experiences and using that information to guide decisions [11]. The primitive type is selected using nearest neighbor, and the primitive parameters are generated using kernel regression. The robot's behavior is determined by how data points are selected and weighted. In the global representation a Euclidian 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 w allows each dimension to be weighted. A query to the database in the global representation is the current state of the environment: marble position (x, y), velocity (V_x, V_y) and board tilt angles (θ_x, θ_y). Typical weights used in the global representation in previous research are 100.0, 1.0, and 1.0 on the position, velocity and tilt angles respectfully.

A distance is similarly computed for the local representation scheme so the same memory-based algorithms can be used. A distance value is computed using the marble's velocity, board angles, and features in the four directions. A value that represents the difference in features between the two situations is computed by comparing the number of features that match. The number of matching features is then divided by the total number of possible matches; for this representation there are 4. Therefore the difference in feature space can be 0.0, 0.25, 0.5, 0.75, or 1.0 where 0.0 signifies that there are no matches and 1.0 signifies that the features in all directions match. To compute a query vector, $\mathbf{q}$, the robot computes the current desired movement direction, locates the features in all directions, and transforms the marble's velocity and board angles to the local reference frame. The number of data points that are considered is reduced by checking if the the primitive type specified by the data point is applicable to the situation the marble is currently in. For example, if a data point specifies a **Roll to Corner** primitive and there is no corner available in the current situation, this data point will not be considered.

The primitive specified by the closest data point determines which primitive type to use for the current situation. The robot then computes the subgoal information such as where the ball will move to and what its velocity should be at the completion of the primitive. The location of local features currently available, such as wall ends and corners, are used to identify the end location for the **Roll off Wall** and **Roll to Corner** primitives. The closest data points of the same primitive type are used to compute other subgoal parameters including the desired marble's velocity and board rotation angles. The outcomes of multiple data points are combined in the following kernel regression:

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

$K(d)$ is a kernel function and is typically $e^{-d^2/\delta}$ and N is the number of data points used in the regression. The estimate for $\hat{y}$ depends on the location of the query point, $\mathbf{q}$.

III. Improving Performance Through Practice

To learn while practicing this task we added a mechanism to change the distance function used in the nearest neighbor lookup done in both selecting primitives and generating subgoals.

We use an estimate of the value function, or actually a Q function, to represent the consequences of choosing a particular primitive in the current task state [12]. A Q function takes as arguments the current task state, $\mathbf{s}$, and an action. In our case the action is choosing to use information from stored point $\mathbf{x}_i$, Choosing data point $\mathbf{x}_i$ at the global state $\mathbf{s}$ has a direct relationship to the action that is taken at this state and therefore $Q(\mathbf{s}, \mathbf{x}_i)$ encodes a Q value. In the case we are maximizing the reward, we use this Q value to compute a scale factor on the distance:

$$\hat{d}(\mathbf{x}_i, \mathbf{q}, \mathbf{s}) = d(\mathbf{x}_i, \mathbf{q}) * \exp((C - Q(\mathbf{s}, \mathbf{x}_i))/\beta) \quad (2)$$

where C is the initial Q value and β controls the influence of Q on the multiplier. $\mathbf{q}$ is the query state in the local representation and $\mathbf{s}$ is the state in the global representation. The data points, $\mathbf{x}$, are encoded using the local representation. By replacing $d(\mathbf{x}_i, \mathbf{q})$ in equation 1 with $\hat{d}(\mathbf{x}_i, \mathbf{q}, \mathbf{s})$, the scale factor has

the effect of moving a stored experience in relationship to the query point. Scale factors greater than 1.0 have the effect of moving the data point farther away from the query point and scale factors less then 1.0 have the effect of moving the data point closer to the query point. This equation gives the the robot the ability to use both local and global information when choosing an action.

Q learning lends itself to updating values by taking into account the result of actions taken in the future. The Q values are initialized with a constant and then updated using a modified version of Q-learning [13]. For each of the data points, $\mathbf{x}_m$, chosen at state, $\mathbf{s}$, the Q-values are updated at the completion of each primitive performance as follows:

$$Q(\mathbf{s}, \mathbf{x}_m) \leftarrow Q(\mathbf{s}, \mathbf{x}_m) + \alpha \cdot [r + \gamma \max Q(\hat{\mathbf{s}}, \hat{\mathbf{x}}) - Q(\mathbf{s}, \mathbf{x}_m)] \tag{3}$$

- α is the learning rate. Since multiple points are used, the weighting given by $\frac{K(\hat{d}(\mathbf{x}_m, \mathbf{s}))}{\sum_{j=1}^{N} K(\hat{d}(\mathbf{x}_i, \mathbf{s}))}$ is used as the learning rate. This weighting has the effect of having points that contributed the most toward selecting the primitive having the highest learning rate.
- r is the reward observed after the primitive has been performed.
- γ is the discount rate.
- $\max Q(\hat{\mathbf{s}}, \hat{\mathbf{x}})$ is the future reward that can be expected from the new state $\hat{\mathbf{s}}$ and selecting the data points $\hat{\mathbf{x}}$ at the next time step. This value is given by:

$$\sum_{i=1}^{N} \left[\max Q(\hat{\mathbf{s}}, \hat{\mathbf{x}}_i) \cdot \frac{K(\hat{d}(\hat{\mathbf{x}}_i, \hat{\mathbf{s}}))}{\sum_{j=1}^{N} K(\hat{d}(\hat{\mathbf{x}}_j, \hat{\mathbf{s}}))} \right]$$

IV. Performing the Marble Maze Task

In this section we describe the performance of the above algorithms on the hardware marble maze task. The robot observes the teacher maneuver the marble through the maze configuration shown in Fig. 1 and then performs the task on the maze configuration shown in Fig. 6. We report results on learning from observation followed by learning from practice.

The training data for learning from observation is created for this task from observing a human maneuver the marble through the hardware maze shown in Fig. 1 during three games. During the three games the robot observed 34 **Roll to Corner**, 36 **Roll off Wall**, 12 **Roll From Wall**, 105 **Guide**, and 20 **Corner** primitives. The primitives are located by observing a sequence of critical events such as the ball just hit a single wall or the ball is against two walls. For more details on our method of segmenting observed data into primitives see [14]. The policies used in the *Action Generation* module have previously been learned from observing and practicing similar mazes and were also used in the research of [8], [9].

Execution of a primitive ends when the subgoal location is reached, if the marble rolls into a hole or outside a bounding box that contains the start and end location, three seconds have elapsed, or when the final goal is reached. In addition to only considering data points that are usable for the local configuration as described in Section II-A, the robot can not use the same data points consecutively and, if a **Corner** primitive was just performed, it can not use **Corner** primitive data points again until the marble has moved at least $2cm$ away from the corner. This helps the robot to not make errors such as repeatedly choosing data points that make no progress or going to a corner and staying there.

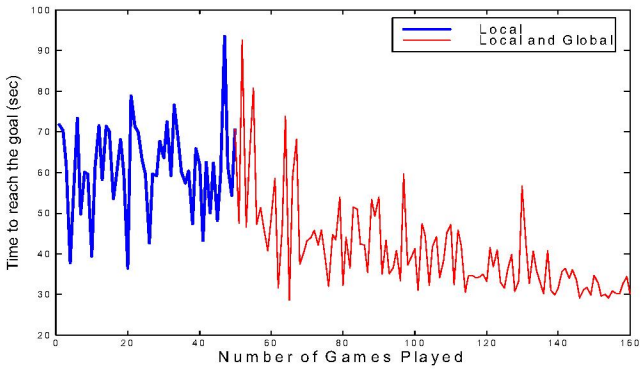

Fig. 5. The time to reach the goal during 160 games played by the robot. During the first 50 games the robot is only using the observed local information, learning only from observation. Starting at game 51 the robot is using both local and global information, learning from observation and practice.

Fig. 5 shows the time to reach the goal as the robot maneuvers a marble through the maze configuration shown in Fig. 6. During the first 50 games the robot is using only local information and completes the maze 9 times without being penalized for a failure. The performance varies a lot from game to game due to the noise of the system. The two main sources of noise are the vision sensing system and the irregularities of the marble. The position information provided by the vision system is filtered and a velocity is calculated from the filtered information. Therefore errors in the marble's position result in larger errors in its observed velocity. The marble is painted so it can be easily seen by the vision system and due to the numerous times it is used, the paint chips away. When the marble is stopped on a chipped surface, it takes a larger board angle to get the marble moving again. This results in a large acceleration that must be handled when the marble does move. The irregular surface also affects the path of the ball when it is moving slowly.

Fig. 6 shows the path of the marble during a typical game. During this game the marble fell into the bottom left hole and restarts the game from the bottom left corner. The robot had difficulty maneuvering the marble through portions of the maze. In the upper right corner it took the robot two tries to get out of that area. Just before reaching the goal, it took the robot many tries to get the marble to make the last turn. If a situation like this exists for more than 10 seconds, the robot is stopped, given an additional 10 second penalty, and play begins again at a location a little further ahead in the maze.

From this information we can see that the robot can perform this task, and sometimes complete the maze without a failure, from only observing a similar task. But the robot should also

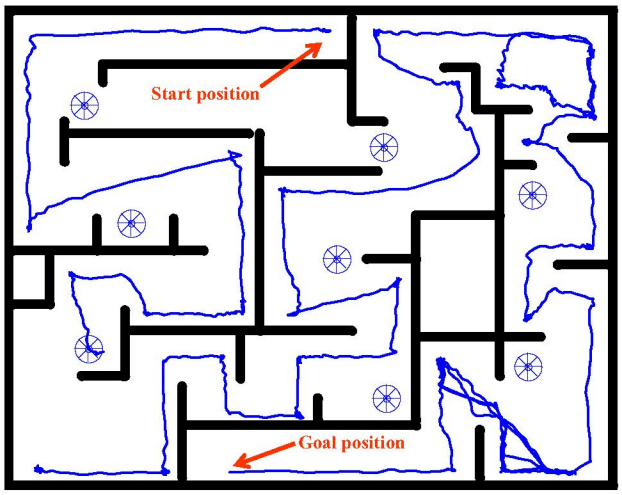

Fig. 6. The path of the marble in the hardware maze during a single trial performed by the robot, after learning from observation only.

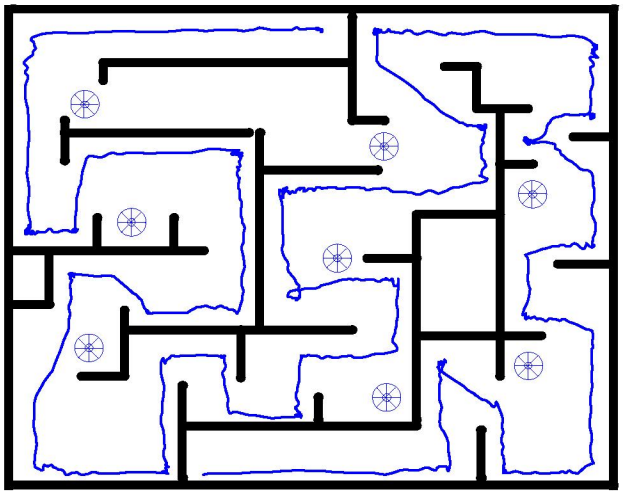

Fig. 7. The path of the marble in the hardware maze during a single trial performed by the robot after practicing 80 games.

be able to perform better by decreasing the number of its failures, performing actions that make the largest progress in the shortest time, and eliminating actions that cause the marble to go backwards in the maze. When a human performed this task the marble had an average velocity of $6.3cm/sec$ through the observed maze. During 5 games played by the robot in the new maze, the marble had an average velocity of $2.8cm/sec$; indicating that the robot should be able to improve its performance with practice.

A. Increasing Performance Through Practice

This same robot is now given the ability to improve its performance through practice. When the primitive ends the reward is computed as follows:

- Moving through the maze: $500\times$ the distance in centimeters from the beginning of the primitive to the end location. Movement along the path toward the goal is positive distance and movement away from the goal is negative distance.
- Taking up time: $-1000\times$ the amount of time in seconds from the time the primitive started to the time it ended.
- Falling into a hole: $-50,000$, the Q-value of the hole state is 0. When the playing agent falls into a hole, learning is stopped and restarted from the location the marble is placed at when the game begins again.
- Reaching the goal: $10,000$.

Fig. 5 shows the improved ability of the robot to quickly reach the goal location as it begins to use the global information starting at game 51. The robot improves significantly after practicing for only 20 games. After practicing about 60 games it has almost entirely eliminated actions that cause the marble to fall into holes. The performance continues to improve as the robot tries to find actions that will allow the marble to traverse slow areas of the maze more quickly. During the first five trials of practicing the marble has an average velocity of $3.1cm/sec$ and during the last five trials it is $4.5cm/sec$; not including the added time penalty and distance due to failures. There are no failures during the last 24 games, and only 5 failures in the last 50 games, compared with 34 failures during the first 50 games practiced (games 51 to 101 of Fig. 5) and 56 failures during the first 50 games played when the robot was not using global information to learn from practice (games 1 to 50 of Fig. 5). Fig. 7 shows a typical path taken by the marble after the robot practices for 80 games.

The values for the various parameters were chosen using trial and error and experience gained during previous research. For the results reported in Fig. 5, they had the following values, weight on the velocity: 10.0, weight on the feature vector 1.0, weight on the board angles: 1.0, kernel function δ: 10.0, number of data points used in the kernel regression of equation 1 (N): 3, initial Q value (C): 0.0, scaling function β: 1000.0, and the discount rate (γ): 0.1. The research presented in [14] shows how the learning from practice algorithm does a good job of compensating for poorly chosen parameters.

B. Comparing Approaches Using the Simulator

To evaluate our algorithms statistically we implemented them on the marble maze simulator. This allows us to quickly conduct many trials. For the software implementation we created a database of 20 games played by a human using the mouse to tilt the board shown in Fig. 1. Every game in the database was completed by the human with no failures. When the software agent performs a learning trial, it first randomly selects three games from the database and creates a primitive selection and subgoal generation database. A trial consist of 500 games played on the simulator where the first 100 games use learning from observation only and during games 101 to 500 the agent is learning from practice.

We compared agents using local and global representations when learning from observation and practice. Because global learning from observation is not able to generalize to another maze layout we used a single board layout (Fig 1) for these

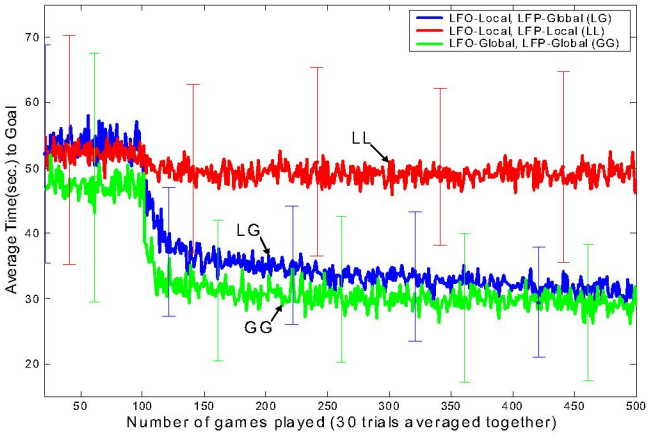

Fig. 8. Time to reach the goal during 30 trials under various learning configurations. LFO-Local and LFO-Global use local and global features respectively to learn from observation. LFP-Local and LFP-Global use local and global features respectively to learn from practice. During each trial, until game 100 the agent is only learning from observation. Starting from game 101 the agent is also learning from practice.

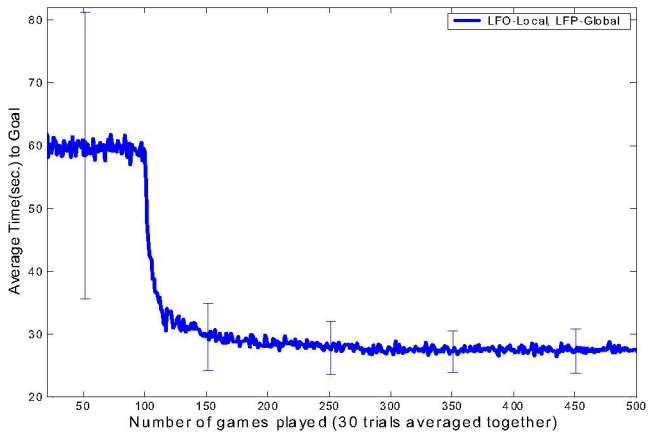

Fig. 9. Time to reach the goal during 30 trials using local features for learning from observation and using both local and global features for learning from observation and practice. The agent is performing the maze shown in Fig. 6.

comparisons. The "LFO-Local, LFP-Global" line in Fig. 8 shows the result when the agent is learning from observation (LFO) using a **local** representation and then learning from practice (LFP) using a **global** representation (condition LG). This is the configuration of the robot in Section IV with the exception that the robot played a different board than it observed. For the "LFO-Local, LFP-Local" line the agent is learning from observation using a **local** representation and then learning from practice also using a **local** representation (LL). In this case the state, s, in equation 3 is the state in the local representation. The Q values are stored in a table of possible local states and indexed using the local state representation. The "LFO-Global, LFP-Global" is for the agent that is learning from observation using a **global** representation and then learning from practice also using a **global** representation (GG). This is the configuration used in our previous research [8], [9]. The graph presents the average of 30 trials. The error bars in these figures represent the standard deviation of the 30 trials.

Looking at the performance during the first 100 games shows how well each agent can perform using only the observed information. The GG agent's policy for learning only from observation is better than that of the LG and LL agents. However, the LG and LL agents have the advantage that they can learn from observing play on another board. The LL agent, using only a local representation, is very poor at increasing its performance through practice. The LG and GG agents can increase their performance significantly with the GG agent performing slightly better.

The GG policy can only be used on the same board that it observed and therefore it can not be tested in other maze configurations. Fig. 9 shows the result of the LG agent using the observed information from the board in Fig. 1 to perform on a different maze configuration shown in Fig. 6. The results are similar to that of the robot on the hardware maze, Fig. 5. These results support the claim that local features support generalization, but global features are much better in supporting learning from practicing the same task than local features.

V. Discussion and Future Research

How well our system can generalize may be determined by the design of the local representation. There are typically many possible local representations one could use for a particular task. Other possible representations include representing the marble maze state with a bird's eye view image, with various types of image features describing the current situation. We could have used a 2D grid around the ball, with each grid cell indicating whether it was empty, had a wall, or had a hole. We could have measured the distances to walls and holes in all directions around the marble, and used a histogram of the distances as the state. Desirable features for a local representation are 1) fixed number of elements 2) supports a distance measure that works for the task. 3) adjustable level of detail or scope. It might be possible to use feature selection or weighting algorithms to find better local representations during practice of a task. There may also be algorithms to invent new types of local representations based on learned data. The focus of this paper is not to present an optimal local representation, but given a possible local representation present how generalization and learning using global parameters can be combined.

VI. Conclusion

We explored using local and global features in learning from observation and learning from practice. The use of local features is a significant extension to our previous research and gives our robot the ability to perform similar tasks. The local policy supports task transfer by providing the robot with an initial guess of which primitive and what subgoals should be used in the new task. Global information, along with a

performance evaluation, allows the robot to improve while performing the task. In the marble maze task, local features supported generalization to similar tasks, but did not support learning from practice. Global features supported learning from practice, but not generalization. Local and global features working together supports task transfer and effective learning from practice.

REFERENCES

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

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

[3] H. Asada and Y. Asari, "The direct teaching of tool manipulation skills via the impedance identification of human motions," in *IEEE Int'l Conf. on Robotics and Automation*, 1988, pp. 1269–1274.

[4] W. L. Wooten and J. K. Hodgins, "Simulating leaping, tumbling, landing and balancing humans," in *IEEE International Conference on Robotics and Automation*, vol. 1, 2000, pp. 656–662.

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

[6] Y. Kuniyoshi, M. Inaba, and H. Inoue, "Learning by watching: Extracting reusable task knowledge from visual observation of human performance," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 6, pp. 799–822, 1994.

[7] J. J. Steil, F. Rothling, R. Haschke, and H. Ritter, "Situated robot learning for multi-modal instruction and imitation of grasping," *Robotics and Autonomous Systems, Special issue on Robot Learning from Demonstration*, vol. 47, pp. 129–141, June 2004.

[8] D. C. Bentivegna, C. G. Atkeson, and G. Cheng, "Learning to select primitives and generate sub-goals from practice," in *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, vol. 1, Las Vegas, Nevada, USA, October 2003, pp. 946–953.

[9] D. C. Bentivegna, G. Cheng, and C. G. Atkeson, "Learning from observation and from practice using behavioral primitives," in *Robotics Research The Eleventh International Symposium Series: Springer Tracts in Advanced Robotics, Vol. 15*, P. Dario and R. Chatila, Eds. Springer-Verlag GmbH, 2004.

[10] D. C. Bentivegna, C. G. Atkeson, A. Ude, and G. Cheng, "Learning to act from observation and practice," *International Journal of Humanoid Robotics*, vol. 1, no. 4, pp. 585–611, December 2004.

[11] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, pp. 11–73, 1997.

[12] C. Watkins and P. Dayan, "Q learning," in *Machine Learning*, vol. 8, 1992, pp. 279–292.

[13] R. Sutton and A. Barto, *Reinforcment Learning: An Introduction*. MIT Press, 1998.

[14] D. C. Bentivegna, "Learning from observation using primitives," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, USA, July 2004, http://www.cns.atr.jp/~dbent/bentivegna-thesis.pdf.