Learning From Observation and Practice Using Primitives

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

[1]ATR Computational Neuroscience Laboratories, Department of Humanoid Robotics and Computational Neuroscience,Kyoto, Japan
[2]Computational Brain Project, ICORP, Japan Science and Technology Agency, Kyoto, Japan
[3] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We explore how to enable robots to rapidly learn from watching a human or robot perform a task, and from practicing the task itself. A key component of our approach is to use small units of behavior, which we refer to as behavioral primitives. Another key component is to use the observed human behavior to define the space to be explored during learning from practice. In this paper we manually define task appropriate primitives by programming how to find them in the training data. We describe memory-based approaches to learning how to select and provide subgoals for behavioral primitives. We demonstrate both learning from observation and learning from practice on a marble maze task, Labyrinth. Using behavioral primitives greatly speeds up learning relative to learning using a direct mapping from states to actions.

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. 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 (Arkin 1998; Schmidt 1988; Russell & Norvig 1995; Barto & Mahadevan 2003). (Bentivegna 2004) presents a more complete survey of relevant work, and provides additional detail on the work described in this paper.

There is a great deal of evidence that biological systems have units of behavior above the level of activating individual motor neurons, and that the organization of the brain reflects those units of behavior. There is evidence from neuroscience and brain imaging that there are distinct areas of the brain for different types of movements. Developing a computational theory that explains the roles of primitives in generating behavior and learning is an important step towards understanding how biological systems generate behavior and learn. A difference between work on behavioral primitives in biology and in robotics is that in biology the emphasis is typically on finding behavioral primitives that are used across many tasks, while in robotics the emphasis is typically on primitives that are only appropriate for a single task or a small set of closely related tasks.

In robotics, behavioral units or primitives are a way to support modularity in the design of behaviors. Primitives are an important step towards reasoning about behavior, abstraction, and applying general knowledge across a wide range of behaviors. We believe that **restricting** behavioral options by adopting a set of primitives is a good way to handle high dimensional tasks. We believe the techniques described in this paper will apply to many current challenges in robotics, such as grasping with an anthropomorphic hand, where object shape and pose can be used to select a grasping primitive. In manipulating deformable objects, such as putting on clothes, the task is quite complex, but simple actions such as pushing an arm through a sleeve or using gravity to orient the clothing are useful primitives to select from. We believe everyday tasks such as cooking, cleaning, and repair can be performed by robots using a library of primitives for specific behaviors and a learned primitive selection algorithm and subgoal generator.

Our current research strategy is to focus on how to best utilize behavioral primitives. Therefore, we manually define primitives. We are deferring the question of how to invent new primitives until we better understand how primitives are best used. Because of our emphasis on learning from observation, the definition of primitives is largely perceptual: the robot must know how to detect the use of a primitive in training data. Other aspects of the primitive, such as how to actually do it, can be learned.

We are exploring a three part framework for primitive use (Figure 1). A **classifier** selects the type of primitive to use in a given context. A **function approximator** predicts the appropriate arguments or parameters for the selected primitive (subgoal generation). Another **function approximator** predicts the necessary commands to achieve the subgoal specified by the parameters for the primitive (action generation). This framework supports rapid learning. The availability of subgoal information decouples action generation learning from the rest of the problem and will allow what is learned at the action level to be reused every time a primitive of the corresponding type is used. In what follows we describe our approach to learning how to select primitives and generate subgoals.

We have used two tasks to develop our thinking, the mar-

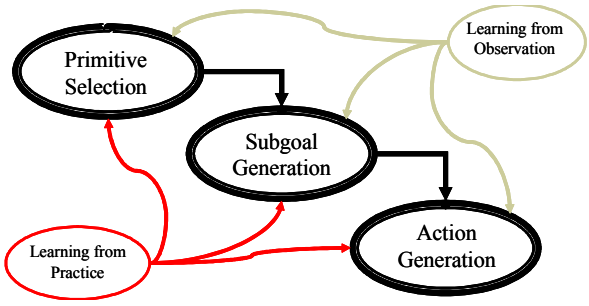

Figure 1: Three part framework for learning using primitives.

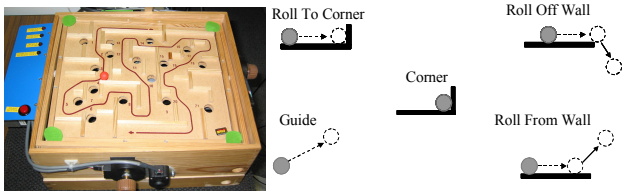

Figure 2: Left: The hardware marble maze setup. Right: Marble maze primitives (top view).

ble maze game Labyrinth (Figure 2) and air hockey (Bentivegna, Atkeson, & Cheng 2003; Bentivegna *et al.* 2002). We selected these tasks because 1) they are challenging dynamic tasks with different characteristics and 2) robots have been programmed to do both tasks, so they are doable, and to the extent there are published descriptions, there are other implementations to compare to. The marble maze task is similar to parts orientation using tray tilting (Erdmann & Mason 1988). The marble maze task will allow us to explore generalization across different board layouts, while air hockey will allow us to explore generalization across different opponents. We have developed versions of these games to be played by simulated agents and by actual 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.

This paper describes some of our work using the marble maze. In the marble maze a player controls a marble through a maze by tilting the board that the marble is rolling on. The actual board is tilted using two knobs and the simulated board is controlled with a mouse. There are obstacles, in the form of holes and walls. The primitives we have manually designed for the marble maze game are based on our observations of human players. The following primitives are currently being explored and are shown in Figure 2:

- Roll To Corner: The ball rolls along a wall and stops when it hits another 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.
- Roll Off Wall: The ball rolls along a wall and then rolls off the end of the wall.
- Roll From Wall: The ball hits, or is on, a wall and then is maneuvered off it.
- Guide: The ball is rolled from one location to another without touching a wall.

Learning From Observation

In learning from observation the robot observes a human or another robot performing a task, and uses that information to do the task. In learning from observation without primitives, the robot learns desired states or state trajectories and corresponding low level actions or action sequences (Atkeson & Schaal 1997). It is difficult to generalize much beyond the state trajectories or reuse information across tasks. It is also difficult to reason about what the demonstrator was trying to do at any time during the observation. Primitives provide a way to generalize more aggressively, and to reuse information more effectively. Furthermore, selecting the appropriate primitive can be interpreted as selecting a subgoal in that point of the task, a simple form of reasoning.

Finding Primitives In The Observation Data

In learning from observation using behavioral primitives, it is necessary to segment the observation (or training data) into instances of the primitives. The example tasks, marble maze and air hockey, have been chosen to some extent because it is relatively easy to segment an observation into distinct primitives. We use a strategy based on recognizing critical events. Examples of critical events for air hockey include puck collisions with a wall or paddle, in which the puck speed and direction are rapidly changed. In marble maze initiating and terminating contact with a wall are examples of critical events (Figure 3). A combination of geometric knowledge (proximity of the ball to walls) and violations of dynamic models (the ball does not accelerate in the direction of board tilt, sudden changes in ball velocity) are used to infer wall contact. Primitives are defined by creating algorithms that automatically segment the observed primitives by searching for sequences of critical events. Parameters or subgoals are estimated by observing the state of the task at the transition to the next observed primitive. In marble maze, the task state is given by the marble position and velocity, and the board tilt angles. There are cases where a primitive can not be identified in the training data (gaps in the figure).

Learning to Select Primitives and Choose Subgoals

In learning from observation using primitives the learner learns a policy of which primitive type and subgoal to use in each task state. The segmented observation data can be used to train this mechanism. There are many classifiers and function approximators we could have used. We have taken a memory-based approach, using a nearest neighbor scheme for the classifier that selects primitives, and kernel based regression for the function approximator that provides subgoals. This provides us with one mechanism, the distance function, that can be used to improve performance in

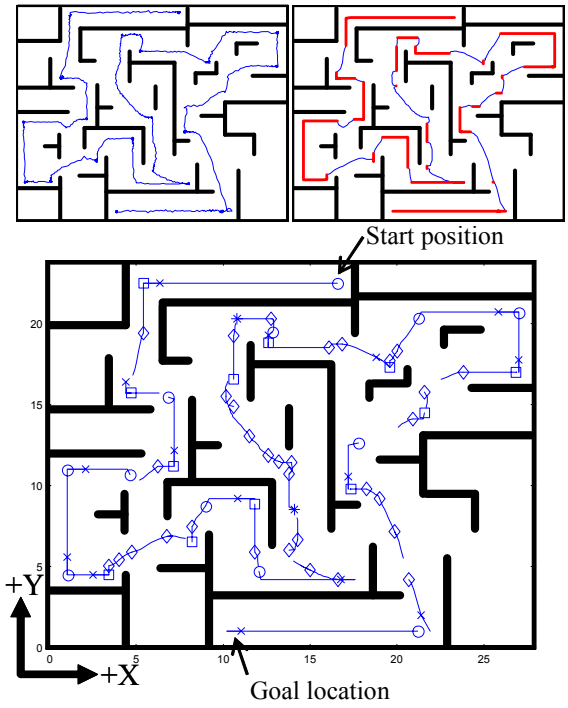

Figure 3: Top Left: Raw observed data. Top Right: Wall contact identified by thick line segments. Bottom: Symbols show the start of recognized primitives: ◯-**Roll To Corner,** □-**Roll Off Wall**, ◊-**Guide**, ∗-**Roll From Wall**, ×-**Corner**.

both primitive selection and subgoal generation. Our system learns by modifying the calculated distances between a query and the stored experiences in memory. Memory-based approaches have a number of other benefits, such as rapid learning, low interference, and control over forgetting (Atkeson, Moore, & Schaal 1997).

Each time a primitive is observed in the training data a corresponding data point is stored in a database that is indexed by the state of the task at the time the primitive was performed. In marble maze, the state is six dimensional, including the marble position (m_x, m_y), marble velocity $(\dot{m}_x, \dot{m}_y)$, and board tilt angles (b_x, b_y). During execution, the current state is used to look up in the database the closest or most similar primitive, which determines which kind of primitive to use. Distance is computed using a weighted Euclidean metric:

$$d(\mathbf{x}, \mathbf{q}) = \sum_j w_j (\mathbf{x}_j - \mathbf{q}_j)^2 \qquad (1)$$

$\mathbf{x}$ is a stored experience, $\mathbf{q}$ is the query to the data base, and j indexes components or dimensions. Each dimension is scaled to range from -1 to 1 based on all the data in the database, and then the following weights are applied: a weight of 100 on marble positions, 10 on marble velocities, and 1 on board angles. We will discuss the sensitivity to these parameters in a later section.

After it has been decided which type of primitive to use, the parameters of that primitive are also specified by the database of prior observations. The closest N stored experiences that involved the same primitive type are used to compute the subgoal for the primitive to be performed. We compute the parameters using kernel regression:

$$\hat{\theta}(\mathbf{q}) = \frac{\sum_{i=1}^{N} \theta_i \cdot K(d(\mathbf{x}_i, \mathbf{q}))}{\sum_{i=1}^{N} K(d(\mathbf{x}_i, \mathbf{q}))} \qquad (2)$$

$\mathbf{x}_i$ is the ith stored experience, $K(d)$ is the kernel function and is typically $e^{-d^2/s}$. As can be seen, the estimate $\hat{\theta}$ depends on the location of the query point, $\mathbf{q}$. We typically used the 4 closest experiences of the appropriate type to compute the subgoal, but N can range from 2 to 6 with little effect.

Results on Learning From Observation

Figure 4 shows the training data and performance using the training data on the hardware marble maze setup. Figure 5 shows the effect of the number of observed games on learning from observation for a large number of simulation runs. In the simulations we turn failures (falling into a hole) into a time penalty: if the marble falls into a hole or does not make progress for 15 seconds it is moved forward in the maze and the player is given a 10 second penalty. One to five games were randomly selected from a database of 50 human games with no failures and playing times of 20-26 seconds.

Learning from observation using this memory-based approach seems to work well. It became clear to us that learning from observation alone had several problems, however. Even under ideal conditions the learner just learns to imitate the performance of the teacher, not to do better than the teacher. In general, the learner did not match the teacher's performance. The same mistakes tend to be made over and over. To improve further, it became clear to us that additional learning mechanisms were necessary, such as the ability to learn from practice in addition to learning from watching others perform.

Learning From Practice

In order to learn from practice, the system needs to have a way to evaluate performance, or have a defined task objective function or criterion. This information is used to update primitive selection, and subgoal 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.

Our system learns from practice by changing the distance function used in the nearest neighbor lookup done in both selecting primitives and generating subgoals. Let's consider the simplest case, where a nearest neighbor classifier is used to select the primitive used in the experience most similar to the current context. If the closest primitive actually leads to bad consequences, we want to increase the apparent distance, so that a different experience becomes the "closest" and a different primitive is tried. If the closest primitive leads to good consequences, we want to decrease the apparent distance. In the nearest neighbor case, decreasing the

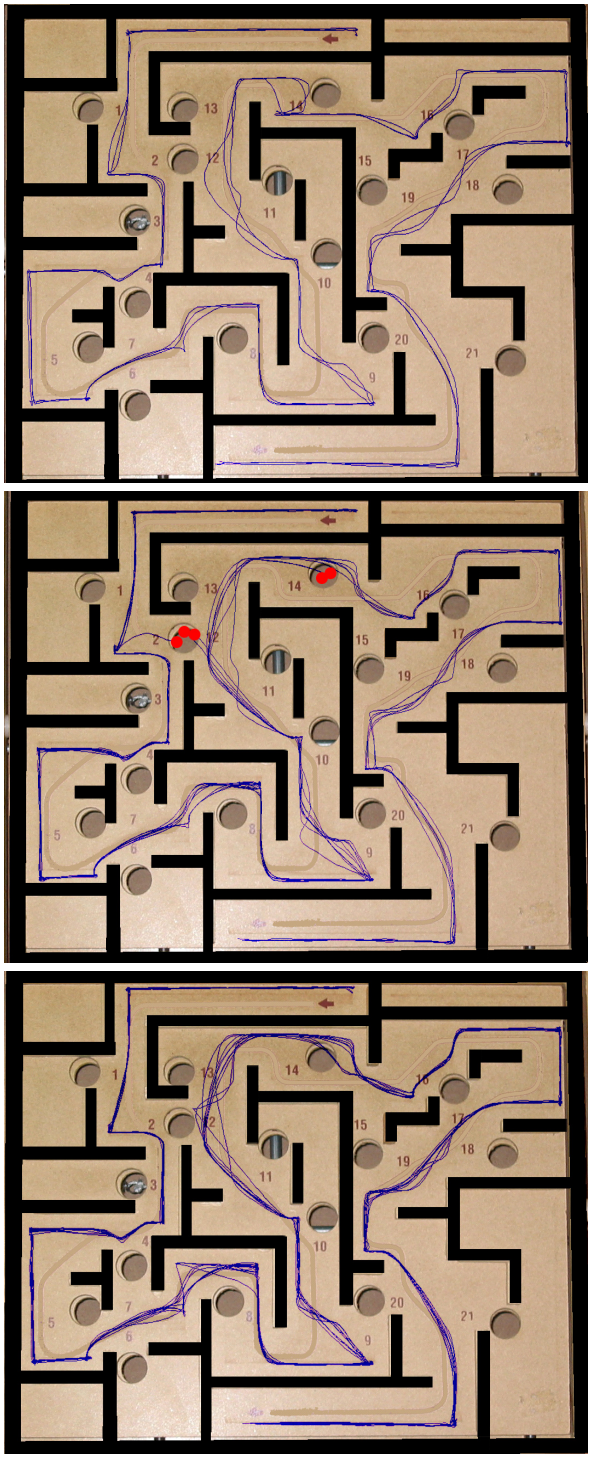

Figure 4: Top: The 3 observed games played by the human teacher. Middle: Performance on 10 games based on learning from observation using the 3 training games. The maze was successfully completed 5 times, and the circles mark where the ball fell into holes. Bottom: Performance on 10 games based on learning from practice after 30 practice games. There were no failures.

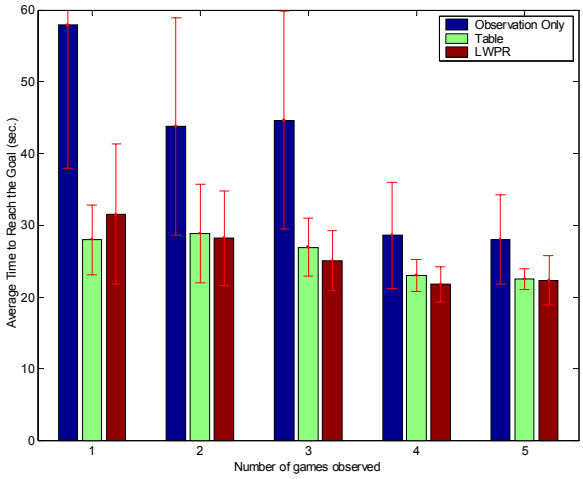

Figure 5: Performance in simulation based on one to five training games: learning from observation using one to five games (left bars), learning from observation and practice using tables (middle bars), and learning from observation and practice using LWPR (right bars). 30 trials using different randomly selected training games were averaged and the error bars are standard deviations of the results. The learning from practice results are based on practicing 300 games.

distance to the closest point has no effect, but in kernel regression it increases the weighting of that subgoal.

Let us continue to consider the simplest case, a nearest neighbor approach. 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 (Watkins & Dayan 1992). A Q function takes as arguments the current task state and an action. In our case, the task state is the query $\mathbf{q}$ to the database, and the action is choosing to use information from stored point $\mathbf{x}_i$, so we have $Q(\mathbf{q}, \mathbf{x}_i)$. We use this Q value as a scale factor applied to the distance, where C is a normalizing constant (version 1 of the scale factor):

$$\hat{d} = d(\mathbf{x}_i, \mathbf{q}) * \frac{C}{Q(\mathbf{q}, \mathbf{x}_i)} \tag{3}$$

This scale factor will have the effect of moving a stored experience in relationship to the query point. Scale factors C/Q greater than 1.0 will 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. For example, if the marble falls into a hole after a selected primitive is performed, the scale factors associated with the set of data points that voted for that primitive selection or contributed to the subgoal generation can be increased. The next time the agent finds itself in the same state, those data points will appear further away and will therefore have less effect on the chosen action. We have also explored using an alternative form of the scale factor (version 2):

$$\hat{d} = d(\mathbf{x}_i, \mathbf{q}) * \exp((C - Q(\mathbf{q}, \mathbf{x}_i))/\beta) \tag{4}$$

Version 2 of the scale factor will allow the Q values to be zero or negative and the influence of the Q value on the multiplier can be controlled by β (typically 20,000).

In our memory based approach, we associate the Q values, $Q_i(\mathbf{q}, \mathbf{x}_i)$, with our experiences, $\mathbf{x}_i$. In order to support indexing the Q values with the query $\mathbf{q}$, we actually associate a table of Q values with each stored experience, and use that table to approximate $Q_i(\mathbf{q}, \mathbf{x}_i)$. This also solves the problem that from one query point state the chosen experiences may be appropriate, but from a different query point, these experiences may not work very well. With a table associated with each stored experience, we can handle this effect. In the marble maze environment the state space is six dimensional. Each dimension is quantized into five cells. Each stored experience in the database has a table of size 5^6. For any query point in the state space, its position relative to the data point is used to find the cell that is associated with that query point. Since we expect only a small fraction of the cells to be used, the tables are stored as sparse arrays and only when the value in a cell is initially updated is the cell actually created. The size of the cells associated with a data point were chosen manually through trial and error with the cells near the center being smaller then those further away. To eliminate the need to specify a large number of parameters and to allow the agent to learn its own discretization of the state space we also explored the use of LWPR models to encode the Q values.

The Q values are initialized with C and then updated using a modified version of Q learning. For each of the data points chosen in the subgoal computation the Q-values are updated as follows:

$$Q(\mathbf{q}_t, \mathbf{x}_m) \leftarrow Q(\mathbf{q}_t, \mathbf{x}_m) + \alpha \cdot [r + \gamma Q(\hat{\mathbf{q}}, \hat{\mathbf{x}}) - Q(\mathbf{q}_t, \mathbf{x}_m)] \tag{5}$$

- α is the learning rate. Since multiple points are used, the weighting given by $\frac{K(d(\mathbf{x}_m, \mathbf{q}_t))}{\sum_{i=1}^{N} K(d(\mathbf{x}_i, \mathbf{q}_t))}$ is used as the learning rate. This weighting has the effect of having points that contributed the most toward selecting the subgoal having the highest learning rate.
- r is the reward observed after the primitive has been performed.
- γ is the discount factor (typically 0.8).
- $Q(\hat{\mathbf{q}}, \hat{\mathbf{x}})$ is the future reward that can be expected from the new state $\hat{\mathbf{q}}$ and selecting the data points $\hat{\mathbf{x}}$ at the next time step. This value is given by a weighted average:

$$Q(\hat{q}, \hat{x}) = \frac{\sum_{i=1}^{N} Q(\hat{\mathbf{q}}, \hat{\mathbf{x}}_i) K(d(\hat{\mathbf{x}}_i, \hat{\mathbf{q}}))}{\sum_{i=1}^{N} K(d(\hat{\mathbf{x}}_i, \hat{\mathbf{q}}))} \tag{6}$$

When each primitive ends the reward function is computed as the sum of the following terms:

- Moving through the maze: the distance in centimeters from the beginning of the primitive to the end location along the path (the line drawn on the board) toward the goal.
- Taking up time: $-10\times$ the amount of time in seconds from the time the primitive started to the time it ended.
- Not making any progress during the execution of the primitive: $-50,000$.
- Not completing the execution of the primitive within 4 seconds: $-20,000$. Primitive execution is terminated after 4 seconds.
- Rolling into, and being stable in, a corner: $30,000$. When a corner is reached, learning stops and restarts from the corner location. The playing agent only gets a reward the first time it goes to a corner. Subsequent visits to the same corner will also reset learning, but will result in no reward. This prevents the agent from returning to corners just to get high rewards.
- 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 location in the maze: $10,000$.

After learning from observation, the system now learns from practice using either tables or LWPR models to alter the distance function. Figure 4 shows individual games on actual hardware after learning from practice (using version 1 of the distance function scale factor and tables to represent the distance function). Figure 5 shows a more comprehensive set of simulation experiments exploring using version 2 of the distance function modification formula and either local tables or LWPR models to alter the distance function. Performance of learning from practice using local tables and LWPR models are about the same. Practice improves performance to human levels and also reduces variability. Figure 6 shows the effect of varying the distance function in simulation. The marble position is clearly the most important factor in learning from observation. However, after learning from practice, performance is relatively independent of the original distance function. The adjustment of the distance scale factors in learning from practice can compensate for poor selections of the original distance function.

Comparison: Learning Without Behavioral Primitives.

We also implemented learning from observation and practice using as direct a mapping from states to actions as we could practically implement. We implemented standard Q learning based on the description of Section 6.5 of (Sutton & Barto 1998). The Q function was represented as a table. We note that this table represents a different kind of behavioral primitive rather than no primitive at all, because the same action is used within each cell. We roughly optimized the size of the cells to maximize the learning rate, and the cell sizes were as follows: marble position cell width: 0.01m, marble velocity cell width: 0.1m/sec, board rotation cell width: 0.01radians, and three possible actions in each state, which change the rotation angle of the board by $(-0.01 radians, 0, 0.01 radians)$. With cells of this size and a board of size $28cm \times 23.5cm$, a maximum velocity of $0.5m/sec$ and maximum board rotation of 0.14 radians there were $(28 * 24) * (10 * 10) * (28 * 28) = 52,684,800$ states. This results in $52,684,800 * (3 * 3) = 474,163,200$ state-action pairs or Q values. Since many parts of the board are inaccessible due to walls and holes the actual number of possible Q values is smaller than this number. We used

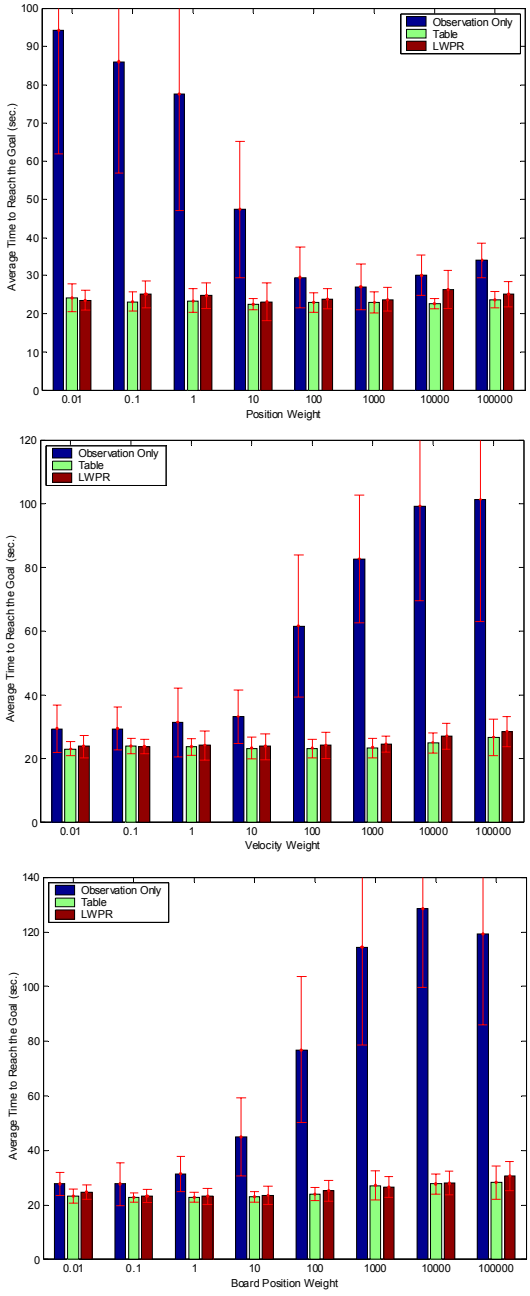

Figure 6: Effect of varying distance function parameters on performance (in simulation): learning from observation using five games (left bars), learning from observation and practice using tables (middle bars), and learning from observation and practice using LWPR (right bars). 30 trials using different randomly selected training games were averaged and the error bars are standard deviations of the results. The learning from practice results are based on practicing for 300 games. The first graph shows the effect of varying the weight on marble position, the second graph shows the effect of varying the weight on marble velocity, and the last graph shows the effect of varying the weight on board angle.

a sparse representation of this table, and typically created many fewer cells.

The agent uses the Q learning algorithm as described in Section 6.5 of (Sutton & Barto 1998). The Q values are updated as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (7)$$

α, the step-size parameter, is set to 0.2 and γ, the discount rate, is set to 0.999. r_{t+1} is the reward received by the agent when it moves from s_t to s_{t+1}. The reward function is $distTrav * 10 - 1$, where $distTrav$ is the distance in centimeters the marble traveled along the path to the goal. If the marble moves in the wrong direction, $distTrav$ will be negative. The -1 term is a penalty on taking time. The agent receives a reward of -1000 if the marble falls into a hole and a reward of 0 when the goal location is reached. The initial Q values, $Qinit$, are set to a pessimistic number (-500). We found that using an optimistic number caused the Q learner to learn much more slowly, since it would explore all possible states and actions until all optimistic values had been decreased.

The action-update cycle occurs at 60 times per second. The agent uses the soft-max function $\frac{e^{(Q_t(a)-Qinit)/\tau}}{\sum_{b=1}^{n} e^{(Q_t(b)-Qinit)/\tau}}$, where τ is set to 10.0, to select actions. The agent operates under the same conditions described previously: if the marble falls into a hole or does not make progress for 15 seconds it is moved forward in the maze and the player is given a 10 second penalty. Figure 7 shows the performance of this agent during 30 trials of $20,000$ games each. The graph shows the running average of the time to reach the goal across the 30 trials.

We show the learning curve of the direct mapping approach after initially training it on 5 randomly selected human-played games (Figure 7). The Q values were updated on the 5 randomly chosen games 20 times, as if the agent was playing the game. Learning from observation improves performance, and decreases the learning from practice necessary to attain maximum performance.

We also show the performance of our approach averaged across 30 trials, with learning from observation on 5 randomly selected human-played games followed by learning from practice. The line marked PRIMITIVES used the approach described in the previous section. The line marked PRIMITIVES2 used the same reward function as was used for the DIRECT case. We see that learning from observation using primitives is much more effective than learning from observation without using primitives. Learning without primitives, after more than 1000 games, eventually matches and sometimes exceeds the performance of the approach using behavioral primitives.

Discussion

Our goal is to find ways to simplify robot programming. In this paper we describe a way to program robots using learning from observation. We also develop an approach to learning from practice (reinforcement learning) tailored to improving learning from observation. To use this approach, a

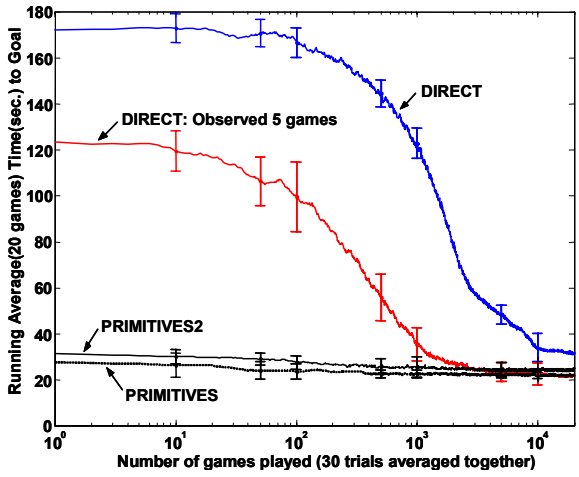

Figure 7: Performance in simulation versus number of games played. The top (DIRECT) line is learning from practice using the direct mapping of states to actions. The next line down (DIRECT: Observed 5 games) is learning from practice using the direct mapping after being initially trained on human games. The next lower line (PRIMITIVES2) is learning from practice using behavioral primitives after observing human games, using the same reward function as the direct mapping case. The bottom line (PRIMITIVES) is learning from practice using behavioral primitives after observing human games using the more complex reward function. Standard deviations are shown as error bars.

robot programmer must identify the task states and actions, define how to find primitives in training data, create a reward function for the task, and demonstrate how to do the task (potentially through teleoperation). There are several key ideas:

- Use behavioral primitives to improve generalization and speed up learning.
- Use memory-based approaches to learning how to select, provide subgoals for, and perform behavioral primitives.
- Use the observed human behavior to define the space to be explored during learning from practice.
- Useful subgoals are intermediate states or sets of states. These subgoals are easily observed, and policies can be learned to attain them.
- Use an existing library of behavioral primitives, rather than learn the primitives from scratch.
- Define primitives in perceptual terms, to enable the robot to detect their usage in training data. Humans describe how to find primitives in the training data, and robots learn how to perform them.

(Barto & Mahadevan 2003) reviews recent approaches to the use of primitives in reinforcement learning. Our emphasis on learning from observation gives our work a quite different focus. We use learning from practice (reinforcement learning) to alter how observed behavior is selected and combined, rather than trying out an arbitrary range of behavior. We also do not try to discover or invent primitives, a major emphasis of other work in reinforcement learning. Another distinguishing feature of our work is the use of memory-based approaches to learning (Atkeson, Moore, & Schaal 1997). We learn from observation by storing observations in a database. We learn from practice by altering the distance function. (Atkeson, Moore, & Schaal 1997) reviews the use of complex distance functions similar to what we have used here in learning from practice. (Ormoneit & Sen 2002; Santamaria, Sutton, & Ram 1998; Smart & Kaelbling 2002) are examples of work in reinforcement learning that use memory-based approaches to represent the value function itself. (Smart & Kaelbling 2002) performs learning from observation by training the Q function using observation data. Some of our current work includes implementing value functions for primitives, which will allow interesting comparisons with the approach described in this paper.

Learning using behavioral primitives in the marble maze task was much faster than learning using a direct mapping from states to actions. One reason for this is that our approach to learning using behavioral primitives **selects** actions from what the teacher did in the local context. Actions that the teacher did not apply are not explored in our version of learning from practice. This is clearly a two-edged sword: learning is greatly sped up, but ultimate performance might be limited. Even when restricted to selecting from demonstrated behavior, the system can learn new strategies. In the three observed games the human maneuvers the marble below hole 14 (Figure 4). During practice the agent falls into hole 14 and learns that it can more easily maneuver the marble around the top of hole 14 by selecting different primitives and generating different subgoals. We did not even know this action was possible until we observed the robot do it.

Another reason learning is sped up is that we have provided opportunities for generalization at the action generation level. Policies for each primitive are improved using data from many parts of the board. Interestingly, we have not taken advantage of another opportunity for generalization: generalization of primitive selection and subgoal generation across the maze and across different mazes. Because primitive selection and subgoal generation is done in terms of board coordinates, we cannot use primitive selection and subgoal generation learned on one maze to improve performance on another maze or in different parts of the same maze. We are exploring indexing primitive selection and subgoal generation using local features of the maze. Performance is currently less using local features than approaches using maze location, since information is lost about future consequences of actions.

What task or domain knowledge was used? Clearly the design of the primitives reflects a great deal of knowledge of the task. The design of a reward function for learning from practice also uses knowledge of the task. The reward function rewarded getting to the end of the maze, and penalized how long each primitive took, failing to make progress, and falling into a hole. The reward function used knowledge of the local direction to the goal to reward the amount of progress made by each primitive. This reflects the mar-

ble maze game, which typically has a desired path marked on the board (Figure 2). It is also easy to calculate such a path using dynamic programming or the A* algorithm, only considering marble positions and obstacle locations. The reward function also explicitly rewarded getting to corners, a very task specific element. Removing some of these features (corner reward, timeout penalty, and no progress penalty) reduced performance somewhat (PRIMITIVES2 line in Figure 7). In action generation learning we assume symmetry, that all corners and walls are dynamically the same. This could easily not be the case. We would have to extend learning to discover differences between similar situations. Interestingly, the only time information about wall location was used was in recognizing primitives in the training data. Wall location was not used by the system while playing the game. Wall and corner locations are implicit in the subgoals.

It is a common belief that the most important research issue in primitives is how they are invented for a new task. There are several arguments against this point of view: 1) For rapid learning, especially from a single observation, there is not enough data for statistical approaches to discovering primitives to work. 2) Our intuition is that some form of learning based on physical reasoning (understanding why corners and walls are useful) is key to obtaining human levels of learning performance. 3) Perhaps biological systems make use of a selection strategy as well. We are born with an existing library of biological eye movement primitives and gaits. It may be the case that biological systems utilize a fixed library of motor primitives during their lifetime. 4) It may be the case that we can manually create a library of primitives for robots that cover a wide range of everyday tasks, and that invention of new primitives is simply not necessary or quite rare. Hence the emphasis in this work is on exploring how to make effective use of existing primitives. It is our hope that this exploration will inform the design of a general set of primitives.

What properties should a set of primitives have? In order to learn from observation, primitives need to be recognizable from observable data. Good primitives divide learning problems up, so that each individual learning problem is simplified. For example, primitives that break input/output relationships up at discontinuities may lead to a set of simple learning problems with smooth input/output relationships.

There are many issues left to be worked on: How can we intermix learning from observation and practice? In this work, learning from practice follows learning from observation. How can we more effectively generalize across similar tasks? We are currently exploring how to generalize across different mazes. In this work we explored model-free learning approaches. We are currently exploring model-based approaches, which learn a model and use the learned model to accelerate policy learning.

Conclusions

We showed how to enable robots to rapidly learn from watching a human or robot perform a task, and from practicing the task itself. A key component of our approach is to use behavioral primitives. Another key component is to use the observed human behavior to define the space to be explored during learning from practice. We developed memory-based approaches to learning how to select, provide subgoals for, and perform behavioral primitives. We demonstrated both learning from observation and learning from practice on a marble maze task, Labyrinth. Using behavioral primitives greatly speeds up learning relative to learning using a direct mapping from states to actions.

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

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

Atkeson, C. G., and Schaal, S. 1997. Learning tasks from a single demonstration. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation (ICRA97)*, 1706–1712.

Atkeson, C. G.; Moore, A. W.; and Schaal, S. 1997. Locally weighted learning. *Artificial Intelligence Review* 11:11–73.

Barto, A., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Systems* 13:41–77.

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

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

Bentivegna, D. C. 2004. *Learning from Observation using Primitives*. Ph.D. Dissertation, Georgia Institute of Technology, Atlanta, GA, USA. http://etd.gatech.edu/theses/available/etd-06202004-213721/.

Erdmann, M. A., and Mason, M. T. 1988. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation* 4:369–379.

Ormoneit, D., and Sen, S. 2002. Kernel-based reinforcement learning. *Machine Learning* 49:161–178.

Russell, S. J., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Santamaria, J.; Sutton, R.; and Ram, A. 1998. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior* 6(2).

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

Smart, W. D., and Kaelbling, L. P. 2002. Effective reinforcement learning for mobile robots. In *IEEE International Conf. on Robotics and Automation*.

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

Watkins, C., and Dayan, P. 1992. Q learning. In *Machine Learning*, volume 8, 279–292.