From: AAAI-94 Proceedings. Copyright © 1994, AAAI (www.aaai.org). All rights reserved.

High Dimension Action Spaces in Robot Skill Learning

Jeff G. Schneider *
Department of Computer Science
University of Rochester
Rochester, NY 14627
schneider@cs.rochester.edu

Abstract

Table lookup with interpolation is used for many learning and adaptation tasks. Redundant mappings capture the important concept of "motor skill," which is important in real, behaving systems. Few robot skill implementations have dealt with redundant mappings, in which the space to be searched to create the table has much higher dimensionality than the table. A practical method for inverting redundant mappings is important in physical systems with limited time for trials. We present the "Guided table Fill In" algorithm, which uses data already stored in the table to guide search through the space of potential table entries. The algorithm is illustrated and tested on a robot skill learning task both in simulation and on a robot with a flexible link. Our experiments show that the ability to search high dimensional action spaces efficiently allows skill learners to find new behaviors that are *qualitatively* different from what they were presented or what the system designer may have expected. Thus the use of this technique can allow researchers to seek higher dimensional action spaces for their systems rather than constraining their search space at the risk of excluding the best actions.

Introduction

Memory-based models such as table lookup with interpolation have been used for many robotic learning tasks [Raibert 77, Atkeson 88, Atkeson 91, Mukerjee & Ballard 85, Moore 90, Moore 91]. The block diagram for a general learning task, and a specific task example (throwing a ball) are shown in Fig. 1. A table residing in the box marked "Skill" holds values for a mapping from a task parameter space to a plant command space. The plant command space is all possible vectors that could be stored in the table. The task parameters are used to index into the table. In the 1-d throwing task, the plant command space is the set

*This material is based on work supported by DARPA contract MDA972-92-J-1012. The government has certain rights in this material.

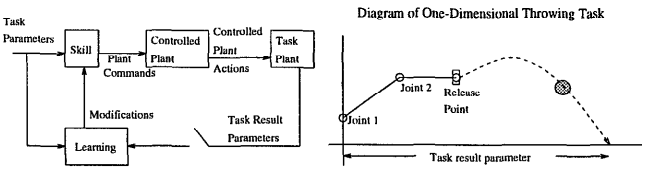

Figure 1: Skill learning system and 1d throwing task

of possible joint velocity sequences that can be sent to the robot controller and the single task parameter is the distance the ball travels.

We assume that our learning system operates in two modes: training and operational. It may train first and then remain in operational mode, or it may switch between the two frequently. In either case our goal is one of optimization: to minimize the amount of time required in training to attain a certain performance level in operation, or to maximize the performance level given a certain amount of training time. Performance may be measured with respect to accuracy, range of operation achieved, or a control effort metric.

Often, memory-based learning systems have relied on random search to fill in the table with the necessary information. This works when: the action space is inherently the same size as the task result space, the system designer has explicitly constrained the action space to be of moderate size using partial task models, or despite the size of the action space the system is interested in learning the results of all possible actions. Robot kinematic and dynamic learning systems often fall into the first and/or last categories. Skill learning systems often fall into the second category. In these kinds of system configurations random search is acceptable. Moore [Moore 90] considered tasks whose action space is of moderate size, but whose desirable actions make up a small portion of the space. He proposed an efficient search strategy for these tasks.

We consider tasks whose space of task parameters has low dimensionality (a small table), but whose space of plant commands has high dimensionality (large vectors stored in the table slots). Tasks of this type arise with open-loop control or planning, when an entire se-

quence of actions is derived from a single current state or goal. Each action in the sequence makes up a dimension of the space of possible action sequences, and different sequences can achieve the goal at different costs: there is redundancy in the mapping. Discrete closed-loop control avoids high-dimensional action spaces by choosing a single action at each time step. Three reasons for open-loop control are: 1) the action is fast with respect to sensor or control time constants. This problem could also be addressed by increasing sensing, computation, and control bandwidth. 2) there is a lack of sensors or controls for state during the task (e.g. during the flight of a thrown ball). 3) delay, which can destabilize a feedback system [Brown & Coombs 91]. The "Guided Fill In" algorithm given here addresses the problem of high-dimensional search to fill a small table. We test the algorithm with an open-loop robot throwing task both in simulation and on a real system.

In addition to standard table lookup methods, local function approximation methods like Kohonen maps [Ritter et al. 92], CMACs [Albus 75, Miller et al. 89], radial basis functions [Poggio & Girosi 89], and back propagation neural networks [Rumelhart et al. 86] store, retrieve, and interpolate between given data points. [Mel 90] combines a neural network approach with a depth first search of possible reaching strategies. Each method develops an efficient representation once a suitable set of input-output pairs has been found. However, none of these addresses the problem of efficiently obtaining the data to be learned. Often, the method is to let the system execute random plant commands and observe the results, which works well when the space of possible plant commands is not unreasonably large.

There is other work that attempts to perform the types of robot skills used to test the "Guided Fill In" algorithm. The use of global function approximation methods for robot skill learning was reported in [Schneider & Brown 93]. Work on throwing and juggling is reported in [Schaal & Atkeson 93, Rizzi 92]. In contrast to our work the task is usually constrained to remove redundancy or accurate models are used to approximate the desired mapping.

Guided Table Fill In

The Guided table Fill In algorithm is a modification of the SAB controller in Moore's thesis [Moore 90]. He was concerned with the efficient search of action spaces, but did not specifically address the issue of inverting a redundant action to task result mapping. Because of time constraints in real systems, it is often impractical to search the entire plant command space. Therefore, skills with redundant mappings have an increased need for search efficiency during training.

The unique inversion of a redundant mapping from some m-space of plant commands to an n-space of task parameters ($m > n$) requires a penalty function to optimize. For example, our 1-d ball throwing robot has two joints controlled via a sequence of six joint velocity settings ($m = 12$, $n = 1$). The penalty function measures the accuracy of the throw and the control effort (sum of squared joint accelerations). The goal of the system is to find the n-subspace of the plant command space that optimally maps onto the n-dimensional task parameter space.

Fields of a Table Entry			
p^{act}	p^{res}	p^{eff}	p^{good}
action	task results	control effort	goodness value

Table 1: Table entry for skill learning

1. Initialize table using approximate task model, teacher, or random choice.
2. Evaluate each point in memory according to its control effort. Assign a "goodness" value based on a comparison of a point's control effort value to its neighbors'
3. Randomly choose goal from desired portion of task parameter space.
4. Generate candidates from plant command space using these methods:
   - Find the table entry whose penalty function is lowest for the desired goal (considering accuracy of task result and control effort). Make random modifications to the action sequence of that entry.
   - For a n-d task space choose $n+1$ "good" samples from the table and interpolate or extrapolate to produce an action for the new goal.
5. Evaluate the probability of success for each of the candidate points.
6. Execute the action with the highest probability of success and store the results in the table (Optionally, readjust the desired range of performance). Goto 2.

Figure 2: See text for a detailed explanation

The result of each system execution is stored in a table. The fields of each entry are listed in Table 1. P^{act} is the action sequence tried and p^{res} the result in task space. P^{eff} is a measure of the control effort required by the action sequence. P^{good} is a goodness value for the entry (its computation is described below).

Guided table Fill In is summarized in Fig. 2. In the first step existing models or teachers can determine points from the plant command space to become the first entries into the table. Random choice is a worst case, but possible. There are two parts to a table entry's evaluation. First (done only once): given a point in plant command space, the system executes the corresponding action (p^{act}) and observes the output (p^{res}) and its control effort value (p^{eff}). The task out-

put parameters determine where the point is recorded in the table. Second (executed once each iteration): compare each point against its neighbors in the table (nearest points in the output space). A point's goodness (p^{good}) is the percentage of neighbors whose effort value is worse than its own.

Step 3 randomly chooses a goal from part of the task space. There are several ways the desired portion of the task space may be specified (discussed later). Step 4 generates candidate actions to accomplish the goal task result. Some of the actions are generated by local random modifications to existing "good" points. The table is searched for the entry that best accomplishes the desired goal considering both accuracy and control effort. The action for that entry is altered with small random changes. Several alterations are done to produce a set of candidate actions.

Moore advocates generating some actions from a uniform distribution over the entire space of actions. The purpose of these candidates is to keep the learner from converging to local minima. Experiments with redundant mappings showed that these candidates were not useful. The probability of a random action in a high dimensional space being useful proved to be too small. However, it is still necessary to generate candidate actions far from the existing set of actions in the table. This is done by using linear combinations of existing "good" points in the table. Several sets of points are chosen randomly with the "best" points having a higher probability of being chosen. Interpolation or extrapolation is done from the chosen points to generate new actions. These actions may be in completely unexplored regions of the action space, but are likely to be more useful than completely random actions because of the way they are generated.

Step 5 evaluates the probability of success as Moore suggests. For each candidate action, p^{act}, the table entry whose action, p^{act}_{near}, is nearest the candidate action is determined and used to estimate this probability. When several task result dimensions are considered, a probability is computed for each dimension and the product of them is the probability for the whole task result. When considering redundant mappings, reducing control effort is also a goal. Therefore, a goal effort is selected (usually a constant, unattainable, low value) and control effort is considered to be another dimension in the above computation. Finally, the candidate with the highest probability of success is executed at step 6. The results of the execution are recorded and a new table entry is made. Steps 2-6 are iterated during the training process.

Simulation Experiments

The table filling and lookup algorithms were tested on robot skill learning and performance on the task of throwing a ball. The results of these experiments are summarized in Table . The skill goal is a vector describing the position of a target and its output is sequences of joint velocities for a throwing robot. Here, the robot is the controlled plant and the forces affecting the ball's flight after it leaves the robot make up the uncontrolled plant. Skills are called n-dimensional where n is the number of parameters in the output space of the task: thus a 2-d throwing task has a target lying in a plane, such as the floor.

Experiments were simulated for 1-d and 2-d throwing tasks. For the 1-d task the robot consists of two joints in a vertical plane (Fig. 1). The control signal is a sequence of joint velocities to be executed over 6 time steps (also called a trajectory), thus making a total of 12 input command parameters. The single output dimension is the distance the ball lands from the base of the robot. In the terms of table lookup, a 12-d space must be searched to fill in a 1-d table. The 2-d throwing task is done with a three joint robot configured like the first 3 joints of a PUMA 760; a rotating base is added to the 1-d thrower. The additional joint yields an 18-d search space. The two task parameter dimensions are the x and y coordinates of the ball's landing position on the floor.

The penalty function includes the approximate amount of energy required to execute the throwing trajectories and the average task output error. The approximate energy measure has a second purpose. Robots have limits on their joint velocities and the metric tends to prefer trajectories that stay away from those limits. Later, the average value of the penalty function over the task parameter space will be referred to as the *performance* and the two terms will be referred to separately as *error* and *effort*.

Standard Table Lookup with Interpolation

Standard table lookup with interpolation using a Random Fill In (RFI) learning strategy was implemented to provide a baseline from which to compare the new algorithms. The fill in procedure is random. At each step a new random trajectory is formed by choosing ending joint velocities and positions from a random, uniform distribution over the range of valid values. The new command sequence is executed and recorded in the table. Retrieval from the table is done by finding two (three) points for linear interpolation in the 1-d (2-d) case. Extrapolation is never used for data retrieval. If no points can be found for interpolation, the nearest point in the task parameter space is chosen. When interpolation is possible, a scalar value is given to each possible pair (triple) of points to determine which should be used. It includes terms for the distance of the points from the goal in task parameter space, the average effort associated with each point's command sequence, and the distribution of the points about the goal point.

Fig. 3 shows some sample results of using RFI for 1-d throwing tasks. The desired range of operation is

1300-5000 mm (the robot's arm has a length of 1000 mm). The graphs represent averages over 20 runs. The x axis is the number of robot executions, or the number of trajectories that can be placed in the table. The y axis is an evaluation of the robot's progress learning the skill. To evaluate the robot's ability, ten targets evenly spaced in the desired range are attempted and the average effort and accuracy are recorded.

One characteristic of the graph is that effort appears to grow as the learning progresses. That happens because the robot is capable only of short throws initially (it is given the same sample short-throw trajectories that are given to the GFI algorithm in the next section). It uses those trajectories when long throws are requested and pays the penalty in accuracy (because it refuses to extrapolate). As it finds trajectories to throw greater distances, it uses them and accuracy is improved. These trajectories require greater effort thus causing the average effort to look worse. This initial decreasing error and increasing effort is a characteristic of many of the graphs presented here.

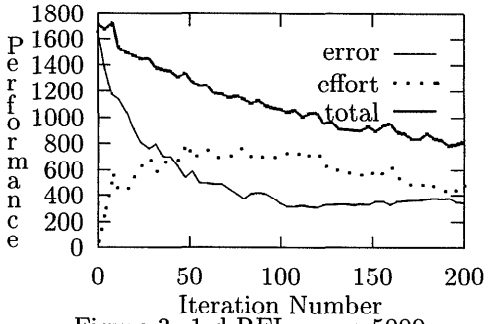

Figure 3: 1-d RFI: range 5000

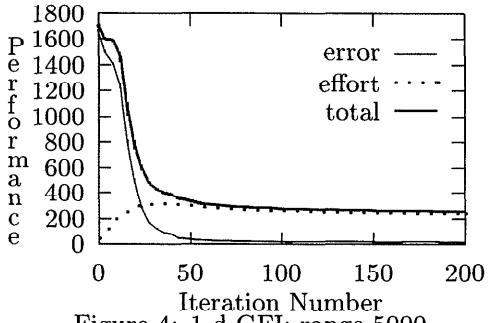

Figure 4: 1-d GFI: range 5000

Results with Guided Table Fill In

The 1-d throwing experiments of the previous section were repeated with Guided table Fill In (GFI). The evaluation and trajectory retrieval methods are the same as for RFI. Five initial trajectories capable of throwing distances from 1370 to 1450 were given in Step 1 (for fair comparison, the same five were given to the learners using RFI). Fig. 4 shows some sample results. The GFI execution attains a value of 254 after 200 iterations compared to 818 in the RFI run.

Improvements can be made when using GFI for longer distances. Automatic range expansion is a modification that allows the algorithm to choose its range of attempts according to its current abilities. Step 6 updates the current range of operation achieved by the system. Step 3 calls for choosing goal task parameters within some desired range. That range is set to be the current range of achieved results plus some distance farther. A parameter controlling how much farther trades off how quickly the range of operation will be increased with how many trials will be spent searching for low-effort actions within the current range. Fig. 5 shows standard GFI for a range of 10000 and Fig. 6 shows the results of using automatic range expansion to a distance of 10000. The additional set of points represent the maximum distance attained (divided by 10 to fit in the graph). Shorter distance throws are easier (there are more trajectories in the command space for them). The algorithmic modification allows the robot to learn the easy part first and then move on to more difficult throws. As the graph shows, it reaches a distance of 10000 after 200 iterations and attains a final performance value of 1072 (compared to 1448 with standard GFI). Traditional engineering practice calls for the range of operation to be pre-specified for system design. However, when dealing with complex plants this may be difficult. Therefore, this modification is also important when a system designer does not know the range of performance that can be achieved, but wishes to maximize it.

Task	Opt. value	GFI 200 tries	RFI 200 tries	RFI tries to catch up
1d 2.5k	24.4	55.0	267.6	over 100k
1d 5k	164.6	254.4	817.7	10k
1d 10k	467.1	1072.3	2443.8	10k
2d 2.5,1.2k	39.2	155.5	1246.9	over 100k

Table 2: Summary of experimental results: Optimal is numerically estimated. GFI is the performance of the new algorithm after 200 iterations using automatic range expansion when it produces improved results. RFI is from traditional table lookup with random trials. The last column indicates how many trials RFI needs to equal GFI's performance.

A 2-d throwing experiment was also done comparing SFI with GFI. The x distance was 2500 mm and the y distance was 1200 mm. GFI is significantly better than RFI for the 2-d task. It gets a final value of 155 compared to 1247 for RFI. This result is important because it shows that performance gains can also be seen in tasks of higher dimensionality.

A revealing statistic is the number of runs RFI requires to reach the performance attained by GFI. For a 1-d range of 2500 RFI requires an average of over 100000 iterations to reach the level GFI attains after only 200. For a 1-d range of 10000, RFI requires 10000 iterations to match 200 of GFI with automatic range

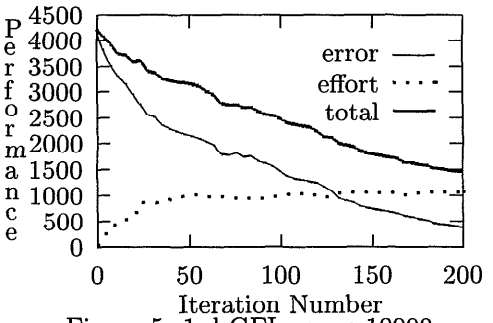

Figure 5: 1-d GFI: range 10000

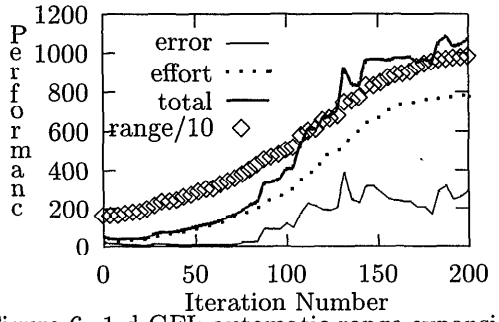

Figure 6: 1-d GFI: automatic range expansion

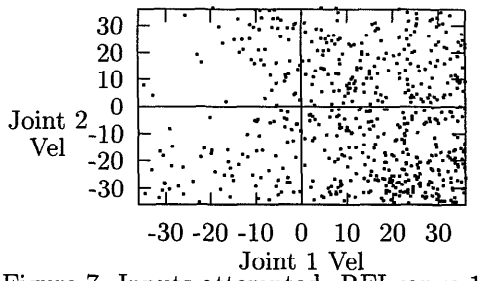

Figure 7: Inputs attempted: RFI range 1.3-5k

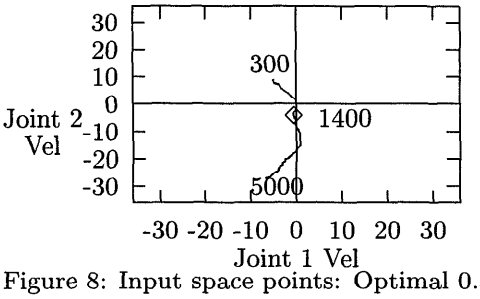

Figure 8: Input space points: Optimal 0.3-5k

expansion. The larger discrepancy between RFI and GFI on the shorter task shows the benefits of using a fill in algorithm that is guided by desired range of performance vs one that randomly tries valid trajectories. GFI is able to concentrate its trials on the portion of the space that throws in the range 1300-2500. Similarly, RFI for the 2-d case required over 100000 iterations to equal the GFI performance with 200 iterations. When skill learning is attempted on real robots, the number of executions required becomes important.

The actions attempted during 1-d GFI and RFI learning show how the algorithm works. Since the search space contains the velocities of two joints, the skill learner is looking for a curve through a 12-d space. Figs. 7 through 10 are projections of the space onto the plane defined by the ending joint velocities. Fig. 7 shows that the distribution of attempted points is fairly uniform throughout the search space when using RFI. The lower density at the left is caused by a higher number of invalid trajectories there. Fig. 8 shows a projection of the optimal curve for a range of 300-5000. A description of how the optimal curve was numerically estimated is in [Schneider & Brown 92].

Figs. 9 and 10 show the points attempted by GFI at different ranges. The figures show that the GFI concentrates its trials in a small part of the search space. The optimal curve verifies that GFI trials are concentrated in a good portion of the space. The dark spot near the center of the two GFI graphs is where the five initial points are located. A comparison of the two GFI graphs shows the algorithm starting from the initial points and working toward the optimal for the desired range.

Experiments on a PUMA

The algorithms presented here were tested on a PUMA 760 with a flexible link attached (a metal meter stick). At the end of the meter stick a plastic cup is attached to hold a tennis ball for throwing experiments. The release point of the ball is not directly controlled. It comes out whenever it is moving faster than the cup. A CCD camera watches the landing area and determines where the ball landed with respect to the base of the robot. Most of the parameters of the experiment were set the same as the 1-d throwing done in simulation. Two joints of the robot are controlled by specifying joint velocities over six 200 ms time steps. The low-level controller, RCCL, interpolates to produce a smooth velocity profile. As in the simulation, the effort function prefers trajectories that are far from the robot's physical limits.

The GFI algorithm was given three sample actions that resulted in throws ranging from 143 cm to 164 cm. Automatic range expansion was used because that option performed the best for 1-d throwing in simulation and because it was not possible to determine the robot's range of capability beforehand. After 100 iterations the robot had learned to throw accurately out to a range of 272 cm (a comparison execution of Moore's algorithm attained a maximum distance of 211 cm). Its accuracy is good enough that it consistently hits a 2 cm screw placed upright anywhere within its learned range of performance. The same cameras that watch the landing position of the ball during learning locate the target during performance mode.

The most interesting result of the learning was the

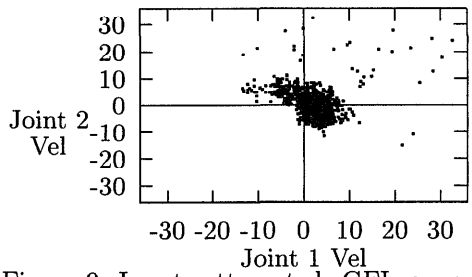

Figure 9: Inputs attempted: GFI range 0-1.5k

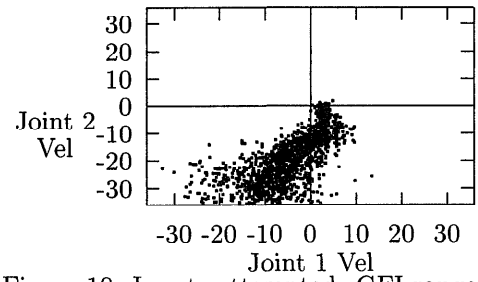

Figure 10: Inputs attempted: GFI range 1.3-5k

type of action found to produce long throws. The three sample actions smoothly accelerate both joints forward. It seems reasonable that longer throws can be obtained by accelerating more quickly. The learning algorithm tried this and it worked up to a distance of approximately 210 cm (this is also what Moore's algorithm did). It was unable to produce longer throws with that type of velocity profile, though, because of the joint velocity limits on the PUMA. It finally learned to do the following "whipping" motion (shown in fig. 11): The joints are moved forward until the meter stick begins to flex forward. Then the robot reverses the direction of its joints so that the stick is pushed forward past its flat state. Just as the stick begins to fall back again, the joints accelerate forward. This causes a large bend in the stick. Finally, the uncoiling of the stick combines with the large forward acceleration of the robot to produce a much higher ball release velocity than could be achieved by simple accelerating the joints forward.

The significant aspect of the long throws that are learned is that they are *qualitatively* different from any given to the system at the start. The sequence of events that led to the robot trying the action in Fig. 11 illustrates the GFI algorithm at its best. At iteration 15 the system was shooting for a goal of less than 200 cm (within its current range of operation). It chose an action created as a local random modification in step 3. That action had a significantly lower velocity at time step 2 for joint 3. The result was a throw for a distance of 164 cm with considerably lower control effort than any previous action for that distance. Later, at iteration 30, a similar thing happened with time step 3 of joint 5. The result was a low effort throw of 176 cm. Following that, the algorithm chose several actions that were generated by linear combinations of these unique actions. Large extrapolations from the new points created velocity profiles with the "whipping" motion shown in fig. 11. The penalty function, small random modifications, and extrapolation all worked together to find new, useful actions in unexplored portions of a high dimensional space without having to resort to brute force search.

Discussion and Conclusions

There are many important tasks with highly redundant plant command to task parameter mappings. When inverting redundant mappings it is necessary to optimize according to additional cost functions. This poses a problem for standard table lookup methods, which require a random or brute force search of the plant command space to optimize performance.

The Guided table Fill In algorithm extends lookup table learning methods to redundant mappings. It uses data already stored in the table to select plant command space candidates that are more likely to produce good data to add to the table. Linear interpolation and extrapolation between existing good points in the table will yield more good points if the mapping is reasonably smooth. The algorithm also allows natural modifications to learn the easy parts of a task first since it explicitly includes a desired range of task parameters in its decision process.

Experiments with robot skill learning show that Guided table Fill In can yield significant improvements in the number of training trials required to attain given levels of performance. They also demonstrate how GFI may be used to learn the easy part of a task first and the performance benefits of doing so. Many sequential task learners [Watkins 89] must operate closed-loop because of the exponential explosion of action possibilities that occurs when a sequence of actions is considered. The results presented here demonstrate one way to deal with the large number of potential actions and thus offer an open loop alternative for these problems. The gains can be significant when the cost of perception is considered or feedback delay in a real-time system becomes a problem.

Experiments using a flexible manipulator for throwing demonstrate the power of the new learning algorithm. Previously, researchers applying learning to robotics attempted to constrain the action space to make the problem tractable. With efficient techniques for searching high dimensional spaces that step may not be necessary. More importantly, the ability to handle high dimensional spaces enables the learner to generate *qualitatively* different behaviors. Often these are the behaviors that the researcher would have eliminated by applying constraints based on poor intuition.

One of the disappointing aspects of work in learning is that it is often applied to tasks where the system designer "already knows the answer." In these situations

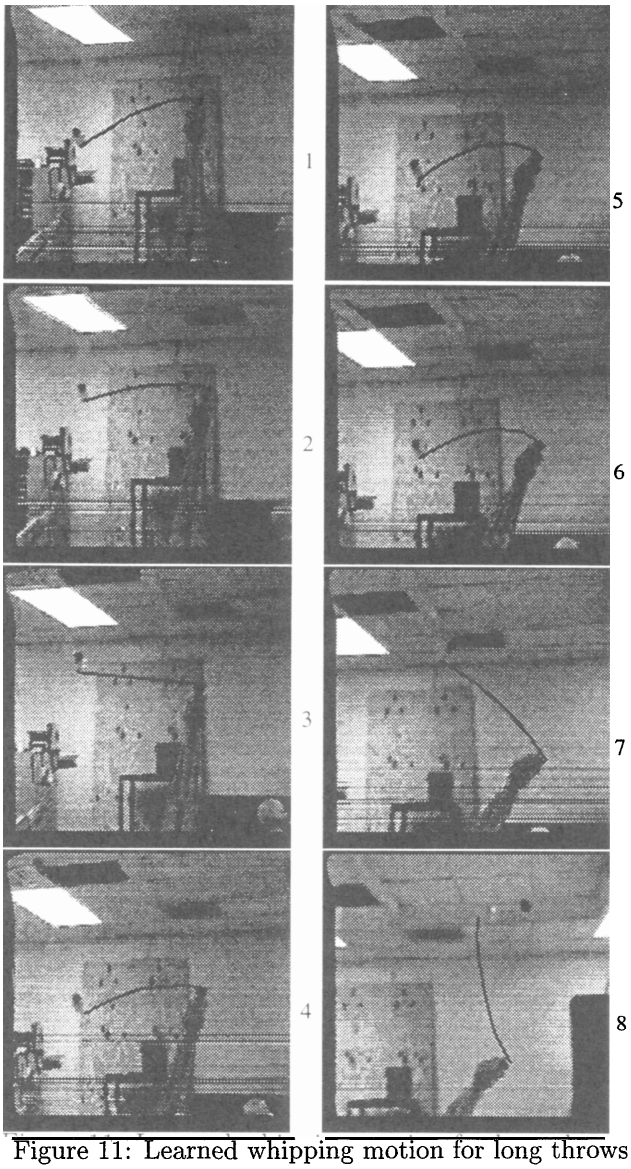

Figure 11: Learned whipping motion for long throws

learning functions more as a fine-tuner to improve accuracy or to fit model parameters. In the throwing experiments presented here, we had speculated that improvements could be made by storing energy in the manipulator. However, it was assumed that this would be done by making an initial backward motion, followed by a forward motion. Only through the use of the GFI algorithm was it revealed that a forward-backward-forward motion was the way to attain a high release velocity, given the constraints on joint velocity, the length of time allocated for the throwing motion, and the natural frequency of the meter stick.

References

[1] J. Albus. A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Journal of Dynamic Systems, Measures, and Controls*, 1975.

[2] C. Atkeson. Using associative content addressable memories to control robots. In *Proceedings of the 27th Conference on Decision and Control*, December 1988.

[3] C. Atkeson. Using locally weighted regression for robot learning. In *Proceedings of the 91 IEEE Int. Conference on Robotics and Automation*, April 1991.

[4] C. Brown and D. Coombs. Notes on control with delay. Technical Report 387, University of Rochester, 1991.

[5] B. Mel. *Connectionist Robot Motion Planning: A Neurally Inspired Approach to Visually Guided Reaching.* Academic Press, 1990.

[6] W. Miller. Real-time application of neural networks for sensor-based controlof robots with vision. *IEEE Trans on Systems, Man, and Cybernetics*, July 1989.

[7] A. Moore. *Efficient Memory-Based Learning for Robot Control.* PhD thesis, University of Cambridge, November 1990.

[8] A. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Proceedings of the 8th International Workshop on Machine Learning*, 1991.

[9] A. Mukerjee and D. Ballard. Self-calibration in robot manipulators. In *Proceedings of the 85 IEEE Int. Conference on Robotics and Automation*, 1985.

[10] T. Poggio and F. Girosi. A theory of networks for approximation and learning. Technical Report 1140, MIT AI Lab, 1989.

[11] M. Raibert. Analytical equations vs table look-up for manipulation: a unifying concept. In *Proceedings of the IEEE Conference on Decision and Control*, 1977.

[12] H. Ritter, T. Martinetz, and K. Schulten. *Neural Computation and Self-Organizing Maps.* A-W, 1992.

[13] A. Rizzi and D. Koditschek. Progress in spatial robot juggling. In *Proceedings of the 92 IEEE Int. Conference on Robotics and Automation*, 1992.

[14] D. Rumelhart, G. Hinton, and R. Williams. *Learning Internal Representations by Error Propagation.* MIT Press, 1986.

[15] S. Schaal and C. Atkeson. Open loop stable control strategies for robot juggling. In *Proceedings of the 93 IEEE Int. Conf. on Robotics and Automation*, 1993.

[16] J. Schneider and C. Brown. Robot skill learning and the effects of basis function choice. Technical Report 437, University of Rochester, September 1992.

[17] J. Schneider and C. Brown. Robot skill learning, basis functions, and control regimes. In *Proceedings of the 93 IEEE Int. Conf. on Robotics and Automation*, 1993.

[18] C. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.