SAND2007-7568C

# ROBUST PERFORMANCE OF AUTONOMOUS ROBOTS IN UNSTRUCTURED ENVIRONMENTS

Brandon Rohrer

*Sandia National Laboratories: MS 1010, PO Box 5800, Albuquerque, NM 87185-1010, brrohre@sandia.gov*

*The problem of navigating in and interacting with an unstructured environment presents challenges to traditional learning and control approaches. However, the nature of emergency response situations requires that autonomous robots' performance be robust to unmodeled environments and unexpected challenges. One approach to providing this capability is presented here: S-Learning.*

*S-Learning, an experience-based learning algorithm, is implemented in the control of a seven degree-of-freedom robotic arm. S-Learning stores sequences of discretized (discrete in time), quantized (discrete in magnitude), and categorical (uninterpreted) sensor data and actuator commands. Handling the data in this way removes explicit models about the environment, robot kinematics, dynamics, and structure. Instead, a bootstrapped model is generated on the fly by observing sequences of sensory and command events. S-Learning is based on a neuro-psychological model of learning and movement control in humans and seeks to mimic the strategies used by the brain to solve this problem.*

## I. INTRODUCTION

Emergency response environments are rarely well-characterized, level, and free from obstruction. The ability of an autonomous robot to handle novel environments is essential to robust performance in an emergency situation. Additionally, the very environmental hazards that require robotic intervention (e.g. unstable rubble, sharp debris, explosive materials, radiation) can damage the robot. Ideally, robot responders would be robust to failed sensors, frozen actuators, misaligned cameras, and joint obstructions to the greatest degree possible. This work describes a robot control algorithm designed to achieve this goal.

The field of "learning to learn," also termed *generalization* or *bias learning*, takes machine performance a step further than many learning algorithms.[1] Generalization algorithms seek to improve system performance not just on tasks for which the systems have explicitly trained, but also on novel, unrelated tasks. Humans are often able to learn a task after only one or two exposures due to the ability to generalize from previously learned tasks. Generalization algorithms attempt to imbue automated systems with this same ability. Common approaches include connectionist networks,[2,3] statistical (including Bayesian, memory-based, and Markovian) methods,[4,5,6] dimensionality reduction,[7] and modified reinforcement learning techniques.[8,9] Within this set of generalization algorithms, a subset is explicitly biologically-motivated. These mimic the human brain, which serves as an existence proof for solutions to daunting perception and control problems. S-Learning falls into this category.

### I.A. Relation to Temporal-Difference techniques

S-learning is a variant of temporal-difference (TD) learning. It is superficially similar to Q-learning,[10] another TD algorithm, but involves *sequences* of discrete events (hence the *S*). TD algorithms are typically effective at discovering optimal sequences of actions in unknown environments. However, existing algorithms only address the static TD problem, in which the states that result in reward or punishment are fixed. This is equivalent to a control system that has a fixed goal that does not vary over time. And while multiple instances of a static TD algorithm, such as Q-learning, can be employed to account for multiple goal states, the experience gained while training one does not transfer to others in a straightforward way. Such an approach typically requires a separate training period for each instance of the algorithm. Even when this multiple-instance approach is successful, it still does not aid the system in reaching unfamiliar goal states.

The distinguishing characteristic of S-learning is that it continually records recurring patterns to build a library of past experiences. This library allows a goal-seeking agent to piece the patterns together to form a complete path to a goal. The strength of this approach is that the goal can be any previously-visited state, not just one or a few that were hard-coded from the start. Thus S-learning can also handle changing goals, multiple goals, and even conflicting goals and provides a potential solution to the dynamic TD problem.

### I.B. Relation to Markov Models

In an S-Learning sequence library, a set of sequences of length two can be accurately represented in a Markov

model. The likelihood of transitioning from state A to state B can be inferred from the sequence set and could alternatively be represented in matrix form. Similarly, longer sequences could be represented as higher-order Markov models. It is accurate to describe an S-Learning sequence library as a shorthand way of representing a series of Markov models of order one to order *N-1*, where *N* is the maximum sequence length. The advantage of a sequence library is that it is concise. A first order Markov model in a system with *M* possible states can be represented by a *M* x *M* matrix, a second order Markov model by a $M^2$ x *M* matrix, and an *N-1* order Markov model by a $M^{N-1}$ x *M* matrix. For the system simulated in this paper, in which $N = 7$ and $M = 2^{4141}$, this representation quickly becomes computationally burdensome. In this sense, a sequence library is a sparse matrix coding for a multi-order Markov model.

## II. METHOD

Note: A description of the S-Learning algorithm has been previously published, e.g. [11], but is briefly presented here for clarity.

Initially, the controller has no experience on which to draw. In simulation, the S-Learning algorithm issues random commands until the goal is achieved, at which point it resets the simulation and attempts to complete the task again. Each time the goal is achieved, the sequence of states (sensor readings and commands issued) leading up to the goal are stored in a sequence library. Each sequence can be envisioned as a trail of discrete states that result in a goal. During future attempts, each state encountered during exploration is compared against previous successful state-trails. If there is a sufficiently close match, S-Learning issues the same sequence of commands that had previously proved successful.

### II.A. Architecture

S-Learning is at the core of a biomimetic Brain-Emulating Cognition and Control Architecture (BECCA, Fig. 1). BECCA consists of an Agent, a Planner, a World, and an S-Learning Engine, each of which is briefly described below.

*II.A.1. Agent*

The Agent sets goals for the system. The goals are expressed in terms of the sensory state information available from the World. Goals can be a specific state, a set of states, or a portion of a state. Multiple, even conflicting, goals can exist. Goals can change over time, and the Agent can use new state information to decide when and how to change them. The current set of goals is available for use by the Planner.

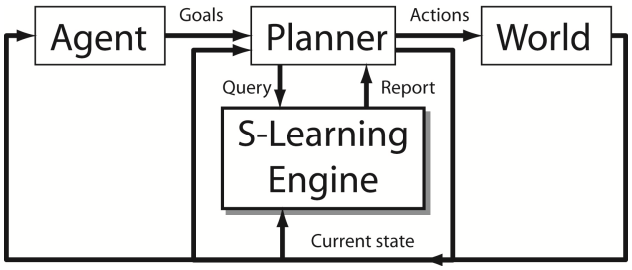


Fig. 1. Brain-Emulating Cognition and Control Architecture (BECCA), featuring S-Learning: a block diagram representation. The S-Learning algorithm is used as an engine to bootstrap a model of the World. This model is referenced by the Planner and uses new state information to refine its World model in order to achieve goals provided by the Agent.

*II.A.2. Planner*

The Planner determines which (if any) actions to take at any given point in time. It takes in goals from the Agent and current state information to inform its decisions. The Planner queries the S-Learning Engine in order to predict the results of possible courses of action. Exploratory actions are also considered, particularly if the current state is unfamiliar and the S-Learning Engine cannot predict a path to a goal state. After a course of action is determined, the Planner issues commands to the World and reports those actions in a state vector.

*II.A.3. World*

The World is the external system that is being learned and controlled. It is analogous to the Plant and environmental disturbances in classical control system formulations. The World can either be simulated or instantiated in hardware, but in either case, the only information it provides back to the rest of BECCA is through its sensors. In simulations, BECCA does not have direct access to the World's internal and state variables.

*II.A.4 S-Learning Engine*

The S-Learning Engine uses the regularly-updated stream of state information to bootstrap a model of the World. There is no explicit model, assumed dynamics, or implied structure. Instead, the S-Learning Engine observes repeated state sequences, particularly those that result in a goal state. These state sequences are stored in a library, which is referenced by the Planner during action planning. The S-Learning Engine also keeps track of the sequences that the Planner selects as action plans. If a sequence leads to a goal, as predicted, it is reinforced by weighting more heavily in the sequence library. If a sequence fails to lead to a predicted goal, its weighting in the library is reduced. After a number of failed

predictions, a sequence becomes sufficiently weak that it is removed from the library.

### II.B. S-Learning Algorithm

S-Learning provides a single mechanism for handling learning, memory, and prediction in BECCA. The learning and memory behavior of S-Learning emerge from the way new states are incorporated into the bootstrapped world model. Initially, the sequence library has no prior experiences and contains no state sequences. When a goal state is achieved (presumably though the exploratory efforts of the Planner) the sequence of events leading up to the goal are stored in the library.

Control in S-Learning is straightforward. All sequences that contain the most recently observed state(s) and terminate in a goal state are candidates for plans. The Planner selects one plan from the candidate set (if there is more than one) on the basis of some criterion, say distance to goal or past success rate.

Other, more sophisticated control methods based on the sequence library are possible as well. For instance, daisy-chaining sequences together, creating trees of possible plans, would allow the Planner to create novel plans and generate a series of sub-goals.

If the Planner finds an appropriate sequence from the library to serve as a basis for a plan of execution, then it executes the sequence of commands contained in that sequence. The planner has an expectation that a goal will be achieved at the end of that sequence. If a goal is not achieved when expected, that sequence of events is appended to the library, allowing future prediction of the same failure.

If a goal is achieved when expected, then the successful sequence is compared against the sequence library. If the observed sequence is significantly different from any sequence in the library, the observed sequence will be appended to the library.

### II.C. Simulation

S-Learning was implemented in a simulated seven degree-of-freedom robot arm, based on physical PowerCube hardware (Amtec GmbH, Germany). MATLAB (Mathworks, Natick, MA) served as the computation engine for the simulation. The robot consisted of six serial rotary links, which terminated in a parallel-finger gripper. (Fig. 2) The robot was mounted on a table, within reach of a salt shaker-sized block.

The simulation consisted of two parts, a physical contact model and a visual representation. In both cases, each link of the robot was treated as a rigid body, and its position relative to the other links was described completely by the kinematic constraints and position of each joint. Due to the high mechanical impedance and non-backdrivability of the joints, links were considered to

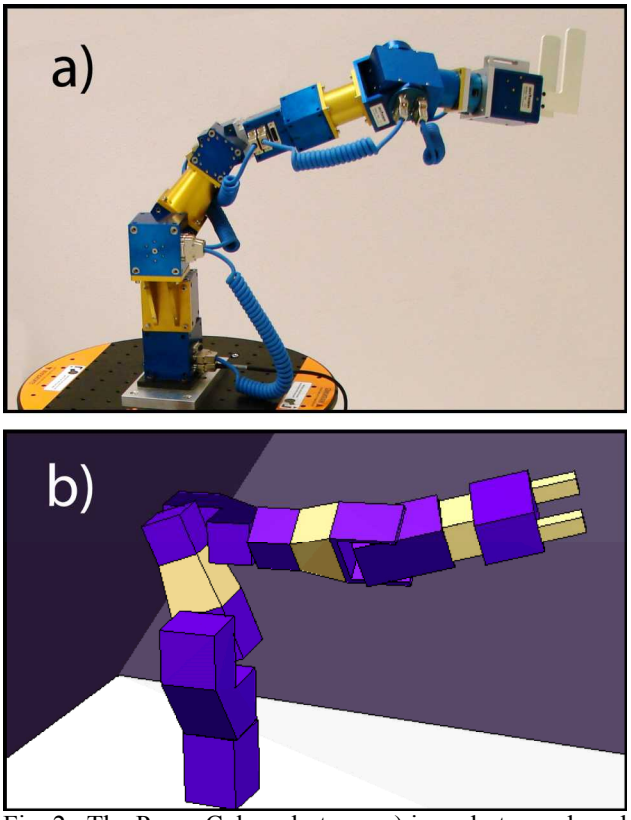

Fig. 2. The PowerCube robot arm a) in a photograph and b) in an image captured from the MATLAB simulation.

have no inertia; inertial effects were negligible in determining movement dynamics. As a result, the set of seven joint positions provided a complete state description of the arm. The visual representation of the model showed the configuration of the arm in the current state in relation to the target block.

The physical contact model used of a number of discrete spheres to represent the physical volume occupied by each rigid link. When the contact spheres from one link impinged on those of another, contact forces were generated. These forces were computed over the entire link and propagated, link by link, down the kinematic chain to the base. If the forces or torques at any given joint exceeded a threshold in the direction of that joint's movement, they prevented the joint from moving against that load. The net effect of this was that the arm was not capable of driving its gripper into the table, or of "crushing" the target block.

Discrete commands were issued to each joint consisting of a position (angle) step in one of 33 magnitudes. This yielded $33^7$ ($>10^{10}$) possible commands. The commands were checked to ensure that they did not attempt to drive any joint past its position limits. This approach was motivated by the physical hardware; discrete position commands are also the accepted command format for the actual PowerCube arm. When a command was issued, a small amount of stochastic

command noise was added, resulting in a non-deterministic system. This jitter provided a means of exploring a local neighborhood of the state space. In addition, when a sequence of commands was being executed, two subsequent commands would, on random occasions, be executed simultaneously. This "carelessness" served to drive learned sequences toward their optimal length. If a learned sequence could be made shorter, it eventually would be.

*II.C.1. Handling sensory information*

The vector of sensory information supplied to the World Model contained joint position, a "goal achieved" flag, and coarse vision from a fixed overhead camera. (Table I) All of the sensor data used in simulation is readily available on the physical robotic hardware platform. In all, there were over 4000 sensor channels feeding information to the S-Learning Engine at each time step. Each sensor channel carried a 1 (signifying "active") or a 0 (signifying "inactive"), making them superficially similar to the afferent neurons that supply sensory information to the brain.

Table I. Sensory Data Channels

| Sensory Modality | Number of Channels |
|---|---|
| Vision: plan view of table | 2500 |
| Position: joint 1 | 600 |
| Position: joint 2 | 300 |
| Position: joint 3 | 200 |
| Position: joint 4 | 200 |
| Position: joint 5 | 200 |
| Position: joint 6 | 100 |
| Position: joint 7 | 40 |
| Flag: goal achieved | 1 |
| Total | 4141 |

The fact that S-Learning does not have a set interpretation of its sensory information means that goals were required to be expressed in terms of raw sensor data. In the case of the PowerCube robot arm simulation, the goal state was considered achieved when the gripper was trying to close, but couldn't (i.e., there was significant positive current flowing to the actuator and significant distance between the two fingers). In terms of the robot's simple set of sensors, this constituted a successful gripping of the target. In the simulated environment, there was only one object within the robot's workspace that could prevent the fingers from closing together—the target block. If there had been a large number of objects—if the target had been located on top of a pile of other objects, for example—then additional sensors would have to be consulted to determine whether the task had been successfully achieved. But in this case (by design), the sensors were adequately suited to the task.

With a sensory state space of $2^{4141}$ ($>10^{1240}$) possible states, exhaustive exploration of the space was prohibitive. A random walk through the state space was also unlikely to reach the target in reasonable time. Two strategies were employed to handle the enormity of the state space. First, a model-independent distance metric for the space was used. For any two states, the fraction of active channels that they shared determines their similarity. More specifically, the similarity, $\sigma$, between two states, A and B, was given by the number of shared active channels, $N_S$, divided by the least number of active channels of the two states, $\min(N_A, N_B)$.

$$\sigma = \frac{N_S}{\min(N_A, N_B)} \qquad (1)$$

This similarity measure yielded a 0 if none of the channels were shared, and a 1 if all the channels were shared or if the active channels in one state were a subset of the active channels in another. In the software implementation, a threshold of $\sigma = 0.93$ was used as a cutoff to determine whether two states were sufficiently close to be considered a match for planning purposes. This value was tuned empirically and was likely specific to the particular system simulated.

The second strategy used to cope with the large state space was to start with an easy task (i.e. success in the task could be achieved with a random walk in reasonable time) and to incrementally increase the difficulty as the robot became more adept. In this way, the robot began the task just outside the border of a familiar region of state space. Each time the robot discovered a path to the goal, this resulted in an incremental increase in the size of the familiar state space. Initially, the robot's position was set such that it was prepared to grasp the target and only needed to close its grippers. Each time the robot successfully grasped the target, the task was reset. Once the robot was able to successfully grasp the target more than five times in 100 moves, the initial distance from the target was increased. This process continued until the maximum initial distance was achieved (shown in Fig. 3a, first panel).

After a certain number of movements without reaching the goal (in this case, 10), the task was reset, and the arm was re-initialized to a starting position as described above. This prevented the robot from spending large amounts of time in the expanses of state space that were far from the goal.

*II.C.2. Task Progress*

The initial starting position was characterized by a quantity called "progress," which ranged from 0 (at the closest starting position) to 100 (at the maximum initial

distance). The heuristic by which progress is modified is given by Eq. 2.

$$\hat{p} = p + \alpha(\min(n_s, 2n_t) - n_t) \quad (2)$$

In Eq. 2, $\hat{p}$ is the new progress value, $p$ is the most recent progress value, $n_s$ is the number of successful target reaches in the last 100 movements, $n_t$ is a constant threshold value, and $\alpha$ is a constant adaptation rate. In this example, an $n_t$ of 5, and an $\alpha$ of 0.0025 produced well-behaved, largely monotonic increases in p, but these values are likely to be specific to this application and may require adjustment for others. The effect of $\min(n_s, 2n_t)$ was to put a ceiling on the rate of change of $p$, preventing large jumps into uncharted regions of the state space.

The initial starting position described by $p$ was used as an upper bound; it was the maximum distance that the robot could use for an initial position at each task reset. The actual starting position was randomly selected from the positions corresponding to a uniform distribution over the interval [0, p], often starting the robot within a familiar region of the state space. This allowed knowledge of the familiar portion of the state space to be continually refined and renewed.

Two additional sources of stochastic variation ensured that the state space was appropriately explored in the neighborhood relevant to the task. Positional variation in both the target and the robot joints were tied to $p$, that is, the magnitude of the variation was 0 when $p = 0$ and maximal when $p = 100$. The maximum magnitudes of these positional variations were $\pi/4000$ radians in the robot joints and 20 cm in the x- and y-positions of the target. The variations in joint position provided only a slight positional jitter. The target position variations were large enough to change the nature of the task.

## III. RESULTS

Initially, the simulated PowerCube arm moved randomly, in an exploratory fashion. Its sequence library was empty, as it had no previous experience. The S-Learning algorithm had no basis upon which to form plans for achieving its goal of grasping the target. Typically, after resetting 5-20 times, the robot finally discovered a sequence of movements that allowed it to grasp the target. It repeated this, subject to the stochastic variation inherent in its motor commands, until its task progress was increased, pushing the robot's initial position out into an unfamiliar portion of the state space. Again, the robot made random movements until it got sufficiently close to a familiar position and was able to reach the target again. This process repeated until the robot was able to reach the target 5 times in 100 moves at a task progress, $p$, of 100. A well-behaving reaching movement from a fully-trained robot is contrasted with random movements in Fig. 3.

Early in learning task progress increased relatively rapidly. Then, later in the process, it slowed considerably. (Fig. 4) This illustrates the increase in difficulty with increased progress, attributable to increased uncertainty in the robot's initial position, the target position, and the magnitude of movement commands. As task progress increased, the magnitude of variation in the robot's initial position increased as well. This occasionally resulted in the robot starting in previously unvisited portions of the state space, requiring additional learning time to reach the goal.

Increases in the magnitude of target position variation resulted in a dramatic increase in task complexity. The robot was, in effect, being presented with a family of tasks rather than just one. Without target position variation, vision information (the majority of the sensory input) would have been superfluous and could have been be ignored. However, when target position was varied, vision information needed to be incorporated and correctly applied to allow successful completion of the task.

Finally, as longer sequences of movements were required to reach the target, the cumulative stochastic variation in those movements had a greater chance of causing significant deviation from the desired end position. While this was helpful in exploring the local state space, it presented one more challenge to the efficiency of the task performance.

Often, the robot was unable to reach the target 5 times within 100 movements. In these cases $p$ was decremented according to Eq. 2. The alternating incrementing and decrementing of $p$ resulted in a jitter that can be seen superimposed on the general upward progression of $p$.

The training time required to reach $p = 100$ for the first time was approximately 280,000 movements, although improvements in performance were still apparent through 500,000 movements.

## IV. DISCUSSION

The simulation presented here demonstrated the S-Learning algorithm controlling a complex robotic arm to achieve a specific task. The algorithm did not use any information about the environment, or the nature of the goal, or the size, range, or topology of the robot. It contained no Cartesian representation of its workspace, no explicit representation of joint angles, and no calibration or interpretation of its rudimentary vision pixels. It learned the relation between all these through self-experimentation or "playing" in its environment.

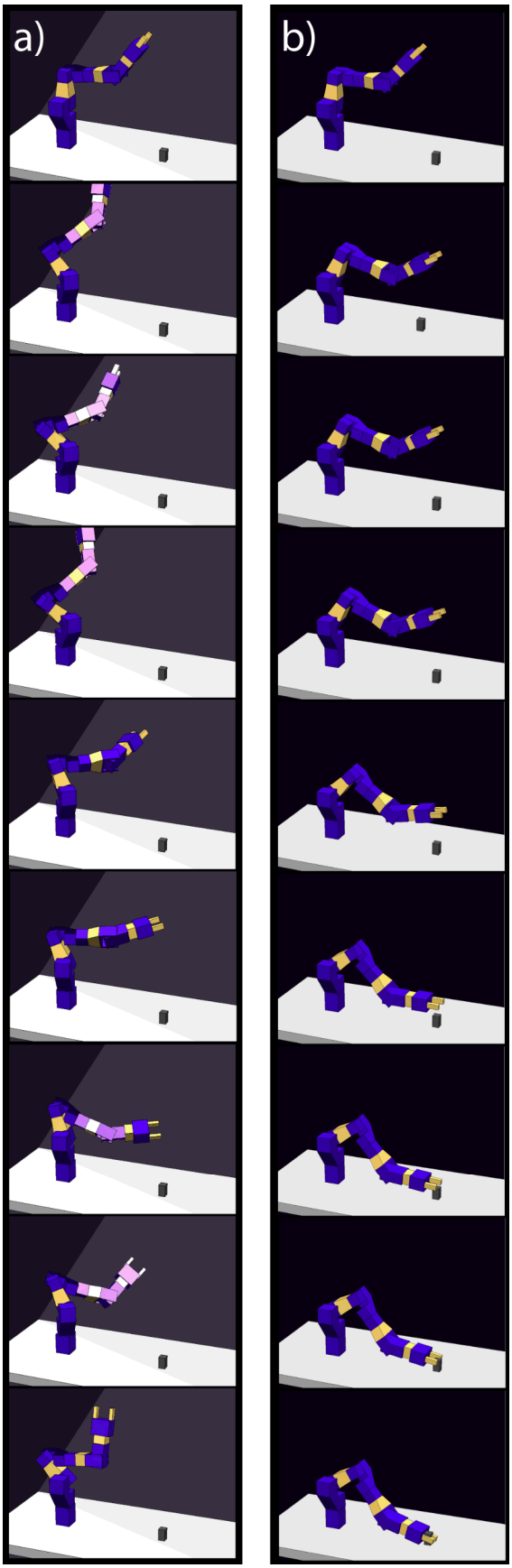


Fig. 3. Video frames from the animations of a) novice or "newborn" reaching performance and b) experienced reaching performance. In novice reaching, there was no experience (i.e. no sequences stored in the S-Learning Engine) to direct the arm to the target or to aid in the interpretation of sensory inputs. After a body of experience had been amassed, sequences linking the initial state to the goal state was used to plan a path through state space to the target.

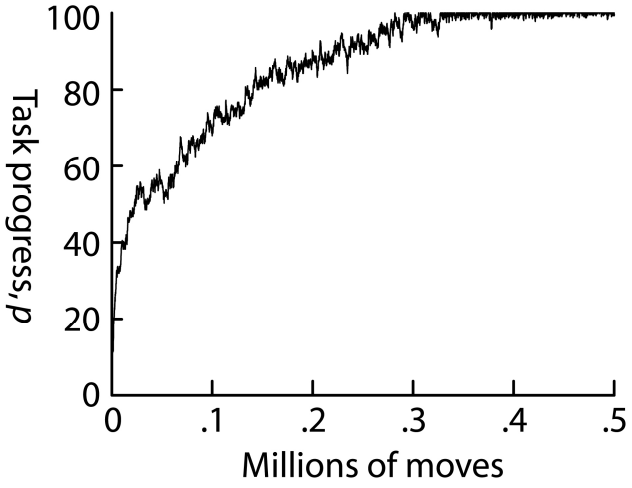


Fig. 4. Progress of the robot performance over time. "Progress," as shown here, represents the average distance from the target at which the arm will begin the task. Initially, the arm was constrained to begin the task very near the target, but as it achieved some success it is gradually started further away. The maximal starting distance, corresponding to progress of 100, is the starting position shown in the first panel of Fig. 3a.

The training time shown of about one-half million moves is not out of line with training time for skilled human movements. Each attempt at the target consisted of as many as ten moves, meaning that approximately 100,000 separate attempts on the target were made. This compares favorably with the number of repetitions it takes for humans to learn a task well. For example, significant improvement in cigar-rolling performance is observed, even after the one millionth cigar.[12]

S-Learning is an extremely general learning approach. The same S-Learning and BECCA algorithm shown here could have been applied to stabilizing an inverted pendulum, learning to grasp with a many degree-of-freedom robotic hand, or steering an unmanned vehicle. As long as the state contains appropriate sensor information and the system has adequate actuation, S-Learning can be used to learn its behavior, store it in

memory, and recall it for use in control. Many other methods have been used to control seven degree-of-freedom robots; it is not a new control problem. The significance of S-Learning accomplishing the task is that it did so without knowing *a priori* how to interpret any of its sensor data or how to reach its goal.

While S-Learning is in most respects a very general tool, there are several parameters that were specifically adjusted to produce desirable performance. In addition to the constants identified in the description of the simulation, the selection and quantization of sensory data was found to be critical to the algorithm's performance. Using irrelevant sensory information can artificially increase the dimensionality of the state space and dilute the algorithm's ability to identify similar states. Quantizing sensory data too finely makes the state space unnecessarily large. Quantizing sensor data too coarsely can prevent the algorithm from distinguishing between qualitatively different states. While the underlying algorithm of S-Learning remains broadly applicable, addressing a number of diverse applications, its application must be engineered to a certain extent, fitting it to the specific application at hand. This combination of general and optimized tools is found in human neurophysiology as well: brains are extremely general in their capabilities and show remarkable plasticity, while the structure, sensitivity, and distribution of sensory neurons appears to be highly optimized. In future work it may be feasible for S-Learning to implement evolutionary algorithms to optimize the implementation-specific parameters, modeling not just its ontologic development, but also its phylogentic development on biological processes.

By basing its function on observed psychophsyiology, S-Learning is able to recreate some of the salient features and strengths of human motor behavior. This paper has demonstrated a simulation of S-learning in a reaching task. However, the general nature of the algorithm suggests that it may also be capable of solving more complex motor control problems, including grasp, bimanual manipulation, visual tracking, balance, and bipedal locomotion. The absence of explicit models makes S-Learning robust to changes in the environment and changes in robotic hardware, such as sensor and actuator failures, that are likely to occur in emergency situations.

## ACKNOWLEDGEMENTS

The generous technical and programmatic support of Kristopher Klingler, Fred Rothganger, Larry Shipers, and Patrick Xavier are gratefully acknowledged.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

## REFERENCES

1. S. Thrun and L. Pratt, "Learning to Learn: Introduction and Overview," *Learning to Learn*, S. Thrun and L. Pratt eds., Kluwer Academic Publishers, pp.3-18, (1998).
2. L. Pratt and B. Jennings, "A Survey of Connectionist Network Reuse Through Transfer," *Connection Science*, **8**:2 (1996).
3. R. Caruana, "Multitask Learning," *Learning to Learn*, S. Thrun and L. Pratt eds., Kluwer Academic Publishers, pp.95-134 (1998).
4. J. Baxter, "Theoretical Models of Learning to Learn," *Learning to Learn*, S. Thrun and L. Pratt eds., Kluwer Academic Publishers, pp.71-94 (1998).
5. D. Shepard, "A Two-dimensional Interpolation Function for Irregularly Spaced Data," *Proc of the 23rd Association for Computing Machinery National Conference*, August 27-29 (1968).
6. J.B. Tenenbaum, "Bayesian Modeling of Human Concept Learning," *Advances in Neural Information Processing Systems 11*, MIT Press, Cambridge, MA (1999).
7. N. Intrator and S. Edelman, "Making a Low-dimensional Representation Suitable for Diverse Tasks," *Connection Science*, **8**:2 (1996).
8. J. Schmidhuber, J. Zhao, and N.N. Schraudolph, "Reinforcement Learning with Self-Modifying Policies," *Learning to Learn*, S. Thrun and L. Pratt eds., Kluwer Academic Publishers, pp.293-310 (1998).
9. R. Maclin and J.W. Shavlik, "Creating Advice-Taking Reinforcement Learners," *Learning to Learn*, S. Thrun and L. Pratt eds., Kluwer Academic Publishers, pp.311-347 (1998).
10. C.J.C.H. Watkins and P. Dayan, "Technical Note: Q-Learning," *Machine Learning*, **8**:2-4, pp.279-292 (1992).
11. B. Rohrer, "S-Learning: A Biomimetic Algorithm for Learning, Memory, and Control in Robots," IEEE EMBS Conference on Neural Engineering, Kohala Coast, HI, May 2-5 (2007).
12. E.R.F.W. Crossman, "A Theory of the Acquisition of Speed-Skill," *Ergonomics*, **2**:2 pp.153-166 (1959).