SAND2009-1808C

Model-free Learning and Control in a Mobile Robot

Brandon Rohrer
Intelligent Systems, Robotics, and Cybernetics Group
Sandia National Laboratories
Albuquerque, NM, USA

Abstract

A model-free, biologically-motivated learning and control algorithm called S-learning is described as implemented in an Surveyor SRV-1 mobile robot. S-learning demonstrated learning of robotic and environmental structure sufficient to allow it to achieve its goals (finding high- or low-contrast views in its environment). No modeling information about the task or calibration information about the robot's actuators and sensors were used in S-learning's planning. The ability of S-learning to make movement plans was completely dependent on experience it gained as it explored. Initially it had no experience and was forced to wander randomly. With increasing exposure to the task, S-learning achieved its goals with more nearly optimal paths. The fact that this approach is model-free implies that it may be applied to many other systems, perhaps even to systems of much greater complexity.

1. Introduction

S-learning is a general learning and control algorithm modeled on the human neuro-motor system [2], [9], [10]. It is model-free in the sense that it makes no assumptions about the structure or nature of the system being controlled or its environment. S-learning accomplishes this using previous experience to help it select actions. This paper describes the implementation of S-Learning in computer code and the application of S-learning to a mobile robot.

1.1. Relation to Previous Work

Most approaches to robot control assume the existence of an explicit system model, such as the changes that a motion command will have on Cartesian or joint positions. The majority of machine learning algorithms take the form of a search in parameter space, with the underlying structure of the space reflecting detailed knowledge of the system. Other methods make a less constraining assumption: that the vectors of state information occupy a metric space. These include finite state machines [12], variants of differential dynamic programming [8], [13], the Parti-game algorithm [7], and probabilistic roadmaps [4]. But even this seemingly benign assumption implies a good deal about the system being modeled. It is violated by any sufficiently non-smooth system, such as one containing hard-nonlinearities or producing categorical state information.

There are still a number of algorithms that are similar to S-learning in that they make no assumptions about the system being learned and controlled. These include Q-learning [14], the Dyna architecture [11], Associative Memory [5], and neural-network-based techniques including Brain-Based Devices [6] and CMAC [1]. These approaches, together with S-learning, can be categorized as reinforcement learning (RL) algorithms, or solutions to RL problems. However, these all assume a static reward function, where S-learning does not.

1.2. Dynamic Reinforcement Learning Problem Statement

To be more precise, S-learning addresses a general class of reinforcement learning (RL) problem, referred to hereafter as the dynamic RL problem: how to maximize reward in an unmodeled environment with time-varying goals. More specifically, given discrete-valued action (input) and state (output) vectors, $a \in \mathcal{A}$ and $s \in \mathcal{S}$, and an unknown discrete-time function f, such that

$$s_t = f(a_{i \leq t}, s_{i<t}, t), \tag{1}$$

(where the notation $a_{i \leq t}$ denotes the set of all a_i such that $i \leq t$) and a scalar reward, r, and known reward function, g, such that

$$r_t = g(s_{i \leq t}, t), \tag{2}$$

maximize the total reward over time:

$$V = \sum_{i=0}^{\infty} r_i \tag{3}$$

Equation 3 shows an infinite-horizon formulation, but finite- and receding-horizon variations of the dynamic RL problem are similarly structured.

The dynamic RL formulation is relevant to a large class of problems. It is applicable in instances where 1) the model is unavailable and 2) the reward function varies with time. Models may be unavailable for a number of reasons. Systems may be too complex to model accurately with the resources available. Also, systems may have characteristics that vary with age, such as joint friction or tire pressure, or may even have non-catastrohpic sensor and actuator failures. Time varying reward functions are introduced whenever the system's goals are modified, as in response to an operator command. Despite the importance of the dynamic RL problem, no other published solutions exist. The nature of the dynamic RL problem—that the only information available is the robot's action-state history—suits it well to an experience-based approach.

2. Method

S-learning operates by recording sequences of state-action pairs. The resulting libraries contain a reduced version of the system's history, a system memory. The memory can then be used to make predictions and guide the selection of the system's actions. When the system encounters a previously-experienced state, it retrieves sequences beginning with that state. The system can then re-execute the actions of recalled sequences that are likely to result in an increased reward.

2.1. S-learning algorithm

S-learning handles state-action (s-a) pairs, σ. An ordered sequence of state-action pairs is called a *sequence*, ϕ, and an unordered collection of ϕ is a *library*, κ. Both ϕ and κ may have any length of one or more, given by n_σ (number of state-action pairs) and n_ϕ (number of sequences), respectively.

An S-learning implementation can be broken into three main function blocks: the Agent, the Environment, and the Sequence Library. (Figure 1.)

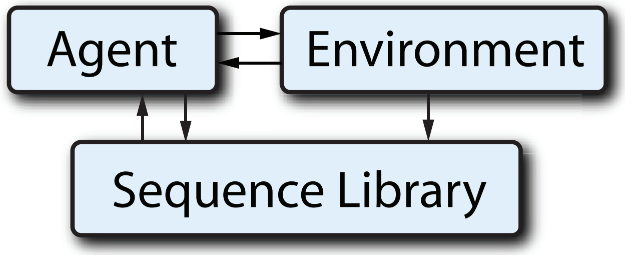

Figure 1. Block diagram of S-learning. The Environment represents the system dynamics, f, and the Agent contains the reward function, g. The Sequence Library is created from the time history of s-a pairs.

2.1.1. Environment. The Environment is the embodiment of the system dynamics, f (Equation 1). It receives action commands from the Agent and reports its state to the Sequence Library and back to the Agent. In practice the Environment may be a continuous-time system, as long as it includes a means to execute discrete-time commands, a, and to report discrete-time sensor information, s.

The formulation of the dynamic RL problem places no constraints on the Environment. It may contain its own internal control system, stochastic elements, and learning capabilities. The Environment may do a large amount of pre-processing on its sensor data and return highly-interpreted information. Alternatively, it may return nearly raw sensor data, binned and discretized in time. It may be physical or simulated, and there are no explicit limits to the complexity it can have.

2.1.2. Agent. The Agent contains the reward function, g, and uses it to evaluate the plan candidates it receives from the Sequence Library. It executes the plans it selects by passing the corresponding actions to the Environment. The procedure the Agent follows during its operation is outlined below:

1) Define a target, τ, consisting of the most recent σ.
2) Query the Sequence Library for sequences that begin with τ, $\phi(\tau)$. The set of these form $\kappa(\tau)$, a collection of candidate plans.
3) Select a plan to execute from $\kappa(\tau)$:
   a) Select the candidate plans that maximize the expected reward, $\overline{r}$, from the states that follow τ in each $\phi(\tau)$.
   b) If there are more than one of these, select the shortest among them, that is, minimize n_σ.
   c) If there is still more than one candidate, randomly select from among the remaining candidate plans such that a single plan, $\hat{\phi}$, is selected.
4) Execute the actions, a, associated with each element of $\hat{\phi}$.
5) Return to step 1.

The Agent also passes copies of the actions it executes, a, to the Sequence Library, so that it can assemble each a-s pair into a σ.

2.1.3. Sequence Library. The Sequence Library is at the heart of S-learning. It allows S-learning to learn from its experience, use new learning as it is gained, generalize that learning to unfamiliar situations, make predictions, and attain goals. It has two primary functions: to pass candidate plans to the Agent and to record state space trajectories as they are observed. Candidate plans, $\phi(\tau)$ are selected on the basis of whether they begin with the target subsequence, τ, passed in by the Agent. The set of $\phi(\tau)$, $\kappa(\tau)$, is returned to the Agent. The process for recording newly observed states in the library is described below.

Due to the fact that S-learning is an experience-based learning algorithm, there is no distinction between memory and learning. Both are accomplished by the storage of sequences. As the Agent passes in actions, a, and the Environment passes in output states, s, the Sequence Library assembles them into a-s pairs, σ. A working memory of the most recently observed states is maintained. Sequences, ϕ, of length n_σ are stored in the library, κ. For ϕ_j that begins with σ_i, ϕ_{j+1} will begin with σ_{i+1}, that is, the subsequent sequences overlap by $n_\sigma - 1$ states. Through this accrual process, κ becomes the repository of the system's experience.

2.2. Robot implementation

The S-learning algorithm was coded in Java and demonstrated with a Surveyor SRV-1 mobile robot (Surveyor Corporation, San Luis Obispo, California, USA). The SRV-1 is a tracked robot with a frontward-mounted color CCD and a Bluetooth radio. (Figure 2) It is relatively small, at 12 cm $\times$ 10 cm $\times$ 8 cm and weighs approximately 350 g. The onboard software that drives the robot is entirely open source. Wireless communications with the robot were accomplished via a software socket, with the robot acting as server in a client-server architecture. The S-learning algorithm was implemented in Java on a remote laptop that received images from the robot's camera and issued movement commands.

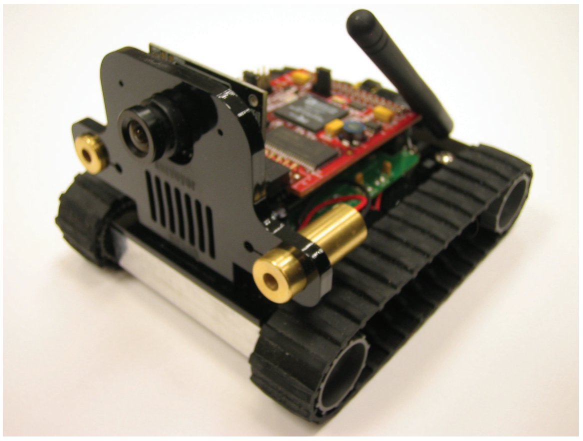

Figure 2. The Surveyor SRV-1 robot.

The robot occupied a 102 cm $\times$ 72 cm room with black walls 40 cm high and a black floor. In the center of each wall was a white stripe 13 cm wide extending the height of the wall. (Figure 3a) At each time step the robot returned an image from its camera to the controlling computer. (Figure 3b)

In order to interface with the S-learning algorithm, the image was preprocessed and binned with a great reduction in the information content. An x-y coordinate system was defined with the origin at the upper left corner of the image, with the positive x direction down and the positive y

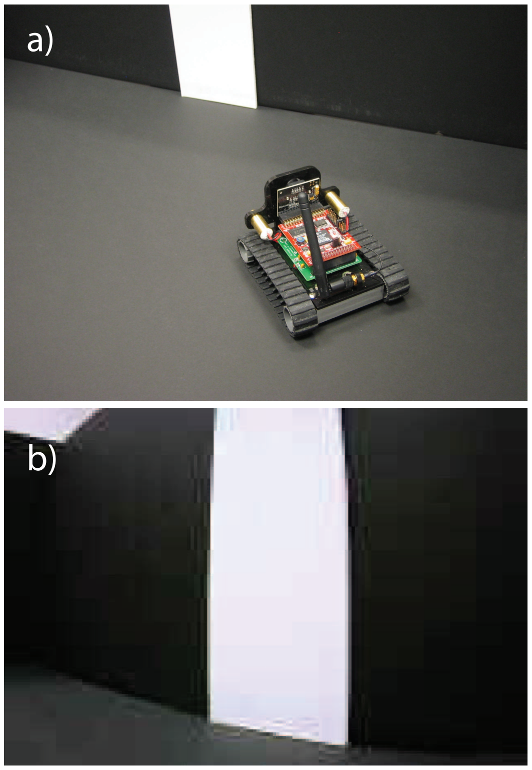

Figure 3. The room that served as the robot's environment. a) Viewed from above. b) Viewed from the robot's camera.

direction to the right. The height of the image was $x_{max} = h$ and the width was $y_{max} = w$. The width-wise center strip of the image from $h/4 < x < 3h/4$ was partitioned into five overlapping vertical strips, each with a width of $w/3$ and an overlap of $w/6$ with its neighboring strip(s). The average pixel value in each strip was calculated by summing the value for each of the red, green, and blue channels over all the pixels in the strip and dividing by three times the number of pixels, resulting and a value between 0 and 255.

Differences between adjacent strips were calculated by subtracting their average pixel values and taking the absolute value. This difference had a theoretical range of 0 to 255, but because adjacent strips shared half their pixels, these differences fell between 0 and 127. The four difference values ($\Delta_1 - \Delta_4$) provided a very rough representation of the amount of contrast between different portions of the image.

2.2.1. Action-state pair vector, σ. The σ vector at each timestep was composed of binary elements representing the command issued and the sensory state after executing the command. Each of the four difference values were binned into 11 bins of width 23.2, covering the range from 0 to 255. The portion of the state vector corresponding to each difference value consisted of 11 binary elements, with each element corresponding to a bin. If an element's bin contained the sensed difference value at a particular time step, that bin would be set to one, as would all bins corresponding to lesser values. In this way, the visual information available to S-learning was greatly reduced with only the minimum necessary retained.

The action vector had four binary elements, corresponding to *forward*, *reverse*, *spin right*, and *spin left* commands. Non-zero elements indicated which commands were issued at that time step. *Forward* and *reverse* commands typically produced a linear motion of approximately 7 cm. *Spin right* and *spin left* commands typically produced a rotation of approximately 5 degrees. Simultaneous *forward* and *reverse* commands produced no action, as did simultaneous *spin right* and *spin left* commands. When both a rotational and linear command were issued simultaneously, only the rotational command was executed.

2.2.2. Reward. The goal of the system emerged from the nature of the reward. A reward vector, ρ, was created with the same length as σ, such that the total reward, r_c, was maximized by high-contrast visual fields. At each time step, r_c was calculated by multiplying σ by ρ element-wise and summing the resulting values. Sensory states were rewarded or penalized by assigning higher or lower values of ρ. The ρ vector used in the simulation was constructed to reward high-contrast near the center of the image moreso than near the edges. The ρ values corresponding to Δ_2 and Δ_3 bins were all set to 2, those corresponding to Δ_1 and Δ_4 bins were set to 1. In practice, this ρ maximized the reward when a white strip was centered in the visual field and occupied about 1/4 the image's width. Also, a reward for low-contrast images, r_d, was derived by subtracting r_c from a constant, 20.

2.2.3. Sequence library creation. At each timestep, the action that was executed and the state that resulted from that action were combined into a state-action pair, σ. The sequence of $n_\sigma^{\max}$ most recently observed sequences was maintained, where $n_\sigma^{\max}$ was the maximum sequence length, a parameter manually set in software. As described above, the longest sequence not in the library already (up to the maximum sequence length) was added to the Sequence Library. Due to the simplicity of the system, all the information necessary to make reasonably accurate predictions about the system was available at each timestep. In this case a maximum sequence length of $n_\sigma^{\max} = 3$ was sufficient. More complex systems would benefit from a greater $n_\sigma^{\max}$, as it would be able to compensate somewhat for partial or noisy state information.

As sequences were added to the library, they were assigned an initial strength value, 10^6. At each timestep, the strength was decreased by 1. The strength of each sequence was multiplied by 10 after each repeat observation. If strength ever decayed to 0, the sequence was dropped from the library. This provided a mechanism for rarely observed sequences to be forgotten. Due to its relatively short run time, the robot did not make use of this feature in this experiment, but it is a feature of S-learning that suits it for use with more complex systems as well. It can also be seen that after several repeated observations a sequence's existence in the library would be assured for the life of the system. This is analogous to recording an experience in long-term memory.

2.2.4. Action selection. The Agent referred to the Sequence Library to help determine which action command to send at each time step. All sequences that began with the most recent state were used as a set of predictions. (The most recent state might be contained in multiple σ's, since several actions may have resulted in that state in the system's history. Sequences beginning with all σ's matching the most recent state were returned.) Each sequence represented a possible future. The Agent compared the reward at the final state of each sequence to the reward at the initial state, and the sequences with the greatest increase in reward were selected as the most promising.

The actions pertaining to each sequence defined a plan. By executing the actions in the same order, it was possible to create the same sequence of states. However it was not guaranteed to do so. Some state information, such as distance to the walls, was not directly sensed and so introduced some variability into the effects produced by a given series of actions. The most promising sequences found in the Sequence Library represented the best case scenarios for each plan. In order to make a more informed decision, the expected value of the final reward for each plan (up to 50 of them) was calculated in the following way.

The library was queried for all the sequences starting from the most recent state and executing each plan. The final rewards for the sequences executing a given plan were averaged, weighted by the log of the strength of each sequence:

$$\overline{r} = \frac{\sum_i r_i \log(\omega_i + 1)}{\sum_i \log(\omega_i + 1)} \tag{4}$$

where $\overline{r}$ is the weighted average reward and r_i is the reward and ω_i is the strength associated with each sequence. One was added to ω_i to ensure that the log remained non-negative.

$\overline{r}$ represented the expected value of the reward a given plan would produce. The plan with the highest value of $\overline{r}$ was selected for execution, given that $\overline{r}$ was greater than the reward at the most recent state.

2.2.5. Neighboring states. When S-learning's prior experience did not provide a plan by which it could expect to improve its reward, it broadened its search to states that were only similar to the most recent state, rather than an exact match. The measure of similarity between states, β, was carefully defined so as not to require any knowledge of the robot hardware, sensor modalities, or the nature of its environment. The introduction of β did not violate the algorithm's model independence.

Shared members between state vectors represent exact matches of *parts* of the states. The Jaccard Similarity Index [3] is a useful measure of the extent to which the sets of active elements in two states intersect. Alternatively, it represents the number of dimensions of the binary subspace in which the active elements of the two states match identically. Given two binary vectors, σ_a and σ_b, $\overline{n}_{\theta^a}$ is the number of active elements in σ_a that are not active in σ_b, and $\overline{n}_{\theta^b}$ is the number of active elements in σ_b that are not active in σ_a. $\overline{n}_{\theta^{ab}}$ is the number of elements that are active in both. The Jaccard similarity between σ_a and σ_b is given by the following:

$$\beta = \frac{\overline{n}_{\theta^{ab}}}{\overline{n}_{\theta^a} + \overline{n}_{\theta^b} + \overline{n}_{\theta^{ab}}} \quad (5)$$

Jaccard similarity is roughly similar to the L^2 norm. In the case that the two states are completely identical ($\sigma_a = \sigma_b$), both are equal to one. In the case where σ_a and σ_b share no common active elements, both are equal to zero. A threshold, β_c, can be set sharply delimiting what represents a match and what does not. For the experiment shown here, $\beta_c = 0.9$.

The use of a similarity measure also implies the existence of a distance metric, $1/\beta$. The introduction of a distance metric would seem to subject S-learning to some of the limitations of other approaches cited in the introduction. However, there is are two important distinctions in how S-learning creates and handles the distance metric. First, the state vector to which the distance metric is applied betrays no information about the state itself, and thus the distance metric is completely model-independent. Second, S-learning uses the distance metric only as a source of guesses for which states are similar. The similarity of any two states is not taken as an axiom within the algorithm. As the robot gains experience, two states that may be identified as similar by β may be revealed as being very different. (For example, while two locations may seem very close in Cartesian space, they may mean the difference between an inmate being inside or outside a prison wall.) S-learning would then base its future actions on its experience, rather than on the nominal similarity of states.

2.2.6. Exploration. If no plans were expected to increase the reward, then the robot generated a random exploratory plan. Exploratory plans were also initiated at random intervals (on average one out of sixty time steps). In addition, a *boredom* condition was met if the same state was observed more than five times within the last fifty time steps. This also resulted in the creation of an exploratory plan. Exploratory plans were of random length, up to 4 actions long. Each action was randomly generated with each of the 4 elements of the action vector having an independent, 50% chance of being active. Exploration provided some random variation to S-learning's operation, allowing it to explore its environment and avoid getting caught in highly non-optimal behavior patterns, such as infinitely-repeating cycles.

2.2.7. Task structure. The robot alternated between two goals: seeking high- and low-contrast visual fields. It was able to increase the contrast of the image by orienting itself toward the white strip from a distance of about 20 cm. It was able to decrease the contrast of the image by orienting itself toward the black wall and zooming in close. A threshold for the reward was manually set for both the high-contrast ($r = 24$) and low-contrast tasks ($r = 12$), based on observed values during exploration. Whenever the reward exceeded the relevant threshold, the goal was considered reached, and the other goal was adopted. In this manner, the robot alternated between contrast-seeking and contrast-avoidance tasks. The measure of the performance in each trial was the number of time steps required to complete the task.

The robot performance was compared under two conditions: 1) active learning and 2) random exploration. In the learning condition, the robot operated according to the algorithm and parameters described above. The robot was allowed to operate for the life of its battery, approximately 4 hours, and the performance for each run was recorded. In the exploration condition the robot did not make use of past experience when selecting actions, but rather executed random exploratory actions at each time time step.

3. Results

The block-averaged results of the experiment are shown in Figure 4 and individual trial results are shown in Figure 5. Initially in the learning robot, the Sequence Library was empty and all movements were random and exploratory. Learning gained during the first movements was used as soon as it was applicable. The earliest runs consisted mostly of exploration and were relatively lengthy. As the Sequence Library became more complete and the state space was better explored the number of time steps required to complete eaach trial decreased rapidly.

Comparison of random exploratory behavior with learning behavior in Figure 4 shows that robot behavior based on its prior experience is clearly superior. The experience-based controller completed the first 25 trials in just over an average 50 moves per trial, whereas the randomly exploring robot required approximately five times as long. This indicates that even the earliest portions of the robot's experience were very quickly applied to forming plans. While the performance of the randomly behaving robot stayed consistently poor (typically 250 or more movements per trial), the performance of the learning robot continued to improve to fewer than ten movements per trial.

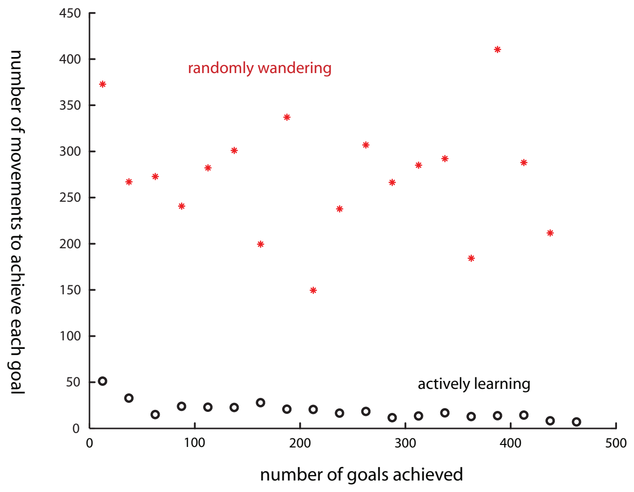

Figure 4. Robot performance in both exploration and learning conditions, averaged in blocks of 25.

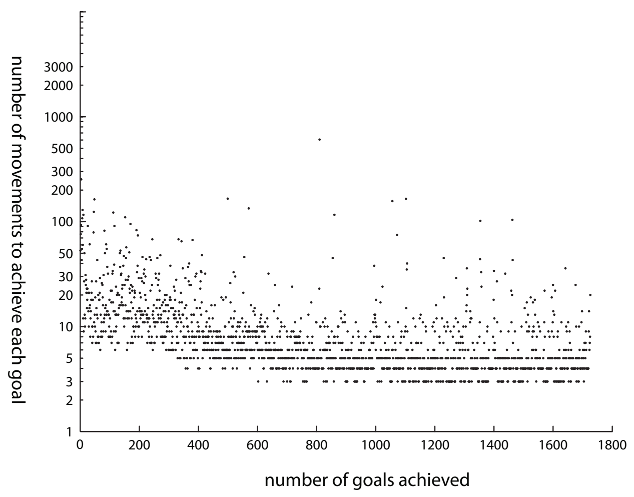

Figure 5. Robot learning performance, logarithmically scaled.

A significant amount of variation can be seen overlying the trends in both the random and learning robots in Figure 4. One major source for the variation in both cases was the fact that in some portions of the robot's environment, goals were easier to reach than in others. The robots' sensory suites were not sufficiently sophisticated to infer their position with the enclosure, and so they were not able to identify and maintain those most advantageous positions. When the randomly behaving robots stumbled into them, however, their success rates increased temporarily, until they drifted into less fruitful positions. When learning robots found themselves in the prime locations within the environment, their performance increased as well, until they also moved.

Figure 5 shows an extended learning interval including more than 1700 trials. Toward the last half of the session, performance regularly reached as low as 3 movements per trial. Although these were interspersed with longer exploratory trials, they show that the robot learned how to achieve its goals in an efficient manner. The logarithmic scaling of the plot de-emphasizes the variation between the very short trials and the very long trials (many longer than 100 movements). It also de-emphasizes the dramatic improvement in performance that occurred from the beginning of the session to the end.

4. Discussion

This work has demonstrated the implementation and operation of S-learning, a model-free learning and control approach. S-learning was able to learn to control a mobile robot in a simple environment. S-learning is capable of addressing some dynamic reinforcement learning problems, including the alternating high- and low-contrast tasks described above. For other examples of S-learning solving dynamic RL problems, see [2], [9].

The two degree-of-freedom, non-holonomic mobile robot used here could be modeled with a small amount of effort. However, S-learning didn't make use of such a model, but treated the robot and its environment as a black box. The key aspect of S-learning's operation is that it relied only on the system's prior experience, rather than on *a priori* knowledge.

4.1. Limitations of S-learning

The robustness and model-independence of S-learning comes at a price. The largest cost is in long learning times. Significant training time, approximately 19,000 time steps, was required to learn to control a relatively simple system. This raises the question of when it would be appropriate to use S-learning. In any implementation where a model is available, the trade-off between which portions to learn and control with S-learning and which to control with a more conventional model-based controller is a trade-off between learning time (short-term performance) and robustness (long-term performance). This question can only be answered based on the specific goals and constraints of each implementation.

Some of the details of S-learning's implementation are specific to the system. One of these details is the maximum sequence length, $n_\sigma^{\max}$. As described previously, $n_\sigma^{\max} = 3$ was known to be appropriate to the simulation due to its relative simplicity. However, other systems may benefit from larger values of $n_\sigma^{\max}$. Humans' capability to remember 7 ± 2 chunks of information suggest that $n_\sigma^{\max} = 7$ is an estimate with reasonable biological motivation. Similarly the dynamics of sequence strength, underlying consolidation and forgetting of sequences, may need to be varied to achieve good performance on different systems. Initial tests show that the most critical design decisions in an S-learning implementation are the discretization of sensor data and the assignment of reward vectors that produce desirable behaviors. Some primary considerations when discretizing sensors are discussed in [9], [10], but additional work is required to fully identify the trade-offs involved.

4.2. Implications

Due to its model agnosticism, S-learning's approach to robot control is potentially applicable to hard problems, such as bipedal locomotion and manipulation. In the case of locomotion, the system model can be extremely, if not intractably, complex, and environments may be completely novel. In addition, extra-laboratory environments can be harsh, and insensitivity to sensor and actuator calibration may be desirable as well. In the case of manipulation, mathematical modeling of physical contact is notoriously difficult and requires a lot of computation to perform well. It also requires high-fidelity physical modeling of the entire system, which is not possible when handling unfamiliar objects. The difficulties involved in modeling both of these applications suggest that locomotion and manipulation are two examples of hard problems to which S-learning may provide solutions. Both problems benefit from S-learning's ability to let the world serve as its own model.

Acknowledgements

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] J. Albus. A new approach to manipulator control: Cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97:220–227, 1975.

[2] S. Hulet, B. Rohrer, and S. Warnick. A study in pattern assimilation for adaptation and control. In *8th Joint Conference on Information Systems*, 2005.

[3] P. Jaccard. Étude comparative de la distribution florale dans une portion des alpes de des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabalistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[5] S. E. Levinson. *Mathematical Models for Speech Technology*. John Wiley and Sons, Chichester, England, 2005. pp. 238-239.

[6] J. L. McKinstry, G. M. Edelman, and J. L. Krichmar. A cerebellar model for predicitive motor control tested in a brain-based device. *Proceedings of the National Academy of Sciences*, 103(9):3387–3392, 2006.

[7] A. W. Moore and C. G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233, 1995.

[8] J. Morimoto, G. Zeglin, and C. G. Atkeson. Minimax differential dynamic programming: Application to a biped walking robot. In *Proceedings of the IEEE/RSJ Intl. Conference on Intellignet Robots and Systems*, pages 1927–1932, 2003.

[9] B. Rohrer. S-learning: A biomimetic algorithm for learning, memory, and control in robots. In *Proceedings of the 3rd International IEEE EMBS Conference on Neural Engineering*, 2007.

[10] B. Rohrer. Robust performance of autonomous robots in unstructured environments. In *Proceedings of the American Nuclear Society 2nd International Joint Topical Meeting on Emergency Preparedness and Response and Robotics and Remote Systems*, 2008.

[11] R. S. Sutton. *Planning by incremental dynamic programming*, chapter Proceedings of the Eighth International Workshop on Machine Learning, pages 353–357. Morgan Kaufmann, 1991.

[12] D. C. Tarraf, A. Megretski, and M. A. Dahleh. A framework for robust stability of systems over finite alphabets. *IEEE Transactions on Automatic Control*, June 2008. To appear as a regular paper in the IEEE Transactions on Automatic Control (scheduled for June 2008).

[13] Y. Tassa, T. Erez, and B. Smart. *Advances in Neural Information Processing Systems*, chapter Receding horizon differential dynamic programming, pages 1465–1472. MIT Press, Cambridge, MA, 2008.

[14] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.