Rapid Reinforcement Learning for Reactive Control Policy Design in Autonomous Robots

Andrew H. Fagg[*] ahfagg@robotics.usc.edu
David Lotspeich[#] lotspeic@robotics.usc.edu
Joel Hoff[*] hoffj@robotics.usc.edu
George A. Bekey[*] bekey@robotics.usc.edu

Center for Neural Engineering
[*]Department of Computer Science and [#]Department of Industrial Systems Engineering
Henry Salvatori Building #300
University of Southern California
Los Angeles, California 90089-0781

Abstract

This paper describes work in progress on a neural-based reinforcement learning architecture for the design of reactive control policies for an autonomous robot. Reinforcement learning techniques allow a programmer to specify the control program at the level of the desired behavior of the robot, rather than at the level of the program that generates the behavior. In this paper, we explicitly begin to address the issue of state representation which can greatly affect the system's ability to learn quickly and to apply what has already been learned to novel situations. Finally, we demonstrate the architecture as applied towards a real robot that is learning to move safely about its environment.

Introduction

Traditional methods of constructing intelligent robotic systems, employing artificial intelligence-based techniques, have met great difficulties in application to real-world problems. Such systems have often required a tremendous amount of computational power in order to make control decisions, and thus have sacrificed the ability to make decisions in real time. In addition, these systems must make many assumptions about the information that is supplied by the sensory processing subcomponents or about the results of actions that are taken. When these assumptions become invalid (which is typically the case when presented with a dynamic environment), these systems become brittle, and ultimately are unable to reliably accomplish their mission.

Reactive or behavior-based control systems have been offered as alternative methods to these more traditional techniques of designing robotic control systems (Arbib, 1989; Arkin, 1990; Bekey & Tomovic, 1986; Brooks, 1986; Brooks, 1991). These approaches embody two key principles. First of all, the control problem is decomposed into a set of simple computing modules, each of which may be designed and implemented separately. Secondly, each module only extracts the information that it needs in order to make a reasonable decision. This approach contrasts significantly with traditional AI approaches, in which the sensing system is used to update a global world model, from which decisions are then made.

However, despite some of the recently reported successes of reactive or behavior-based approaches, hiding behind most successes is a graduate student who spends many hours carefully designing, testing, and redesigning the set of control modules, until the desired behavior is achieved (exceptions to this include (Maes & Brooks, 1990; Mahadevan & Connell, 1992)). One reason that this process is so laborious is that it is typically very difficult for a programmer to put herself *in the shoes of the robot*, and truly understand the information that is being provided by the (often imperfect) sensing subcomponents, as well as understand the range of possible outcomes of actions taken by the robot. In addition, when a robot is picked up and placed into a new environment, there is no guarantee that the control program will continue to function as desired (Verschure & Kröse, 1992).

One possible approach to these difficulties is the application of a reinforcement-based learning technique (Barto & Bradtke, 1991; Barto, Sutton, & Anderson, 1983; Samuel, 1967; Sutton, 1988; Watkins & Dayan, 1992; Williams, 1987). Here, the programmer (or teacher) provides the robot with only an evaluation of its behavior. In our case, this evaluation is a scalar score that is potentially (but not necessarily) given for each control decision that is made by the controller. We refer to the manner in which the reinforcement information is computed as the *reinforcement policy*. Based upon this abstract representation of the desired behavior, the task of the learning system is to infer a reactive control strategy that satisfies the reinforcement policy provided by the teacher.

However, many reinforcement learning techniques suffer from the amount of time required to learn an effective control strategy. This difficulty is due in part to the manner in which states are represented in these systems. Purely localist representations (Barto & Bradtke, 1991; Barto, et al., 1983; Watkins & Dayan, 1992) store the relevant information for each possible state that the system might encounter. Such an approach suffers because it scales poorly with an increasing number of state variables. In addition, neighboring states are not able to share information with one-another, requiring that the system visit all states during the learning process in order to ensure an optimal control policy.

On the other end of the spectrum are the completely distributed representations, such as backpropagation-based techniques (Williams, 1987), where every (hidden) unit participates to some degree in each learned mapping. Although such approaches allow for generalization between states, the very same mechanism that gives us this feature also causes a significant amount of interference between dissimilar states. The result is a rapid slow-down in learning as the state space becomes larger or more complicated. In this work, we seek a state representation that sits somewhere between these two extremes - that allows us to capture some degree of generalization, but is still localist enough such that learning in different regions of the state space will not interfere with one-another.

In the remainder of this paper, we first present a specific neural architecture for the representation of reactive control programs. We next show how this model can be updated in light of reinforcement-based information from a teacher, and how a module that predicts future reinforcement can be used to solve the temporal credit assignment problem. Finally, we illustrate the behavior of the architecture when it is presented with several tasks to be learned.

Problem Description

The Robot

The robotic system used in these experiments is an adapted radio-controlled car. The vehicle is equipped with two tactile bumpers that are mounted on the Front and the Rear of the vehicle. In addition, there are five sonar sensors which are oriented in different directions: Left, Forward, Right, Up (forwards), and Rear.

The Task

The world in which the robot is situated is a standard laboratory environment, with a large variety of obstacles, some of which (such as chair and table legs) are rather difficult to detect, especially with a moving sonar platform. In work to date, we have experimented with three different tasks that the robot is to learn:

1. Avoid collision with obstacles.
2. Environmental exploration.
3. Wall following.

For each different task, the teacher chooses an appropriate reinforcement policy. For example, one possible reinforcement policy for task 1 is to punish the robot when it collides with an obstacle (reinforcement = -1), and otherwise no information is given (reinforcement = 0). From this sparse information, the learning control system must learn a control policy that reliably keeps the robot from receiving the negative reinforcement.

Network Model

The general network architecture is depicted in Figure 1, and has evolved from our work on modeling of primate visual/motor conditional learning (Fagg & Arbib, 1992). The goal of the network is to map current sensory inputs into appropriate actions. Inputs from five sonar units, and two bumpers (I) activate a particular pattern activity across the feature detector units (F). The feature detector units then interact with one-another to contrast-enhance the activity pattern across the vector of feature detectors (G). The feature detector units that continue to be active vote for a favored set of actions at the *action-selection layer* (A). The votes from the set of active feature detector units are gathered together at each action through a summation operation. The one action with the highest activity is chosen to be executed for the current time-step, after which the process is repeated. The six possible actions are movements in the following directions: Left Forward, Straight Forward, Right Forward, Left Reverse, Straight Reverse, and Right Reverse.

In the interest of computational efficiency, the unit dynamics are implemented as a one-pass system. The sensory vector (I) contains one neuron for each bumper sensor (active if touching something), and three neurons each for the sonar inputs (corresponding to Near, Mid, and Far sonar ranges). The mapping from sensors (I) to feature detector units (F) is implemented as a simple matrix-vector operation:

$F = W * I + Noise$

where

I and F are vectors representing the input unit activities and the feature detector inputs.

W is a weight matrix (initially random) that maps from I to F.

$Noise$ is a vector of random signals that are injected into the feature detector units.

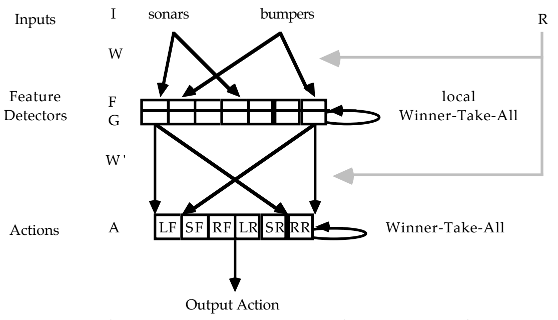

Figure 1: The general network architecture. The sensor inputs activate a set of feature detector units. Those remaining after the local Winner-Take-All competition instantiate votes for one of six actions (LF = left forward, SF = straight forward, ... , RR = right reverse). The one with the highest number of votes is output for execution.

The contrast enhancement function at the feature detector layer is implemented by a *local winner-take-all* operation. Unlike the standard *winner-take-all algorithm*, a particular unit only has an effect over a small neighborhood of the feature detector field. In fact, the interaction between the units is best described as a *mexican-hat connectivity*, with near neighbors supporting the activity of one-another, and neighbors that are further out inhibiting each other.

The interaction amongst the feature-detector units is implemented according to the rule:

$$Winner_i = \begin{cases} 1 & if\ F_i = \underset{i-N \le j \le i+N}{Max} \{F_j\} \\ 0 & otherwise \end{cases}$$

where:

N defines a local neighborhood of feature detectors.

The output activity values (G) are then computed in two steps:

1. $G_i = Winner_i * F_i$
2. If $Winner_i = 0$ and **i** is a close neighbor of a winner (call it unit **j**), then :

   $G_i = Max(F_i, \gamma F_j)$ where $0 < \gamma < 1$

The Winner vector determines the location of the peaks of activity within the feature detector layer (as implemented by equation 1). Close neighbors of the winning feature detectors are also activated to some degree (equation 2). *Close neighbor* is smaller than the neighborhood size (N), used to compute the Winner vector.

The contrast-enhancement operation serves a vital role in the assignment of credit during the learning process. Through its application, it is guaranteed that only a small number of units in the feature detector layer actually become active at any one time (and thus instantiate their votes at the action selection layer). This is important because when reinforcement information becomes available, identifying those feature detector units that are at fault (so that learning can occur) is a much more precise computation. This will be elaborated further in the Discussion section.

The votes are then collected by the action selection units :

$A = W' * G + Noise'$

The action that is output is action **i** such that:

$A_i = \underset{j}{Max}\{A_j\}$

Once the selected action is executed, a new set of sensory inputs is presented to the network and the process of selecting a new action is repeated.

Network Learning

In parallel to the execution process described above, the learning system makes updates to the weight matrices W and W' based on the reinforcement information (R) that is received from the teacher. The sign of this signal indicates the appropriateness ($R > 0$) or inappropriateness ($R < 0$) of the *recent behavior* of the robot, and the magnitude of the signal represents the degree of this (in)appropriateness. Inherent in this definition is the fact that an entire sequence of actions may contribute to the final reinforcement signal that is provided by the teacher. Thus, the learning algorithm is faced with propagating current reinforcement information backwards through time in such a way that the recent actions are updated appropriately.

We will first consider the structural credit assignment problem, which states that given some instantaneous state of the network and a reinforcement signal, R, how is blame assigned to the individual weights of W and W'.

Suppose that the execution of the last action yielded a positive reinforcement signal from the teacher. In order to increase the probability of making the same decision the next time the same situation arises, two things must be done. First of all, we must insure that the same set of features are recognized (i.e. the same set of feature detector units are turned on). This may be accomplished by increasing the strengths of the synaptic weights from the currently active input units to the currently active feature detector units. Secondly, given that the same set of features are recognized, the same action must be taken. This is captured by increasing the strength of the synaptic weights from the currently active feature detector units to the selected action.

On the other hand, suppose that a negative reinforcement signal is received from the teacher. The selection of the incorrect action may be due to one of two cases. First of all, the incorrect set of feature detector units may have been selected. If this is the case, then the connection strengths from the currently active input units to the currently active feature detector units should be decreased. The next time that the same situation arises, the total input to these feature detector units will be weaker, giving them less of a chance to win the local winner-take-all competition. In the second case, the set of feature detector units is correct, but the selected action is incorrect. For this case, the connection strength from the active feature detector units to the selected action is decreased, thus reducing its future probability of being selected. The only difficulty is that the system does not know which of the two cases are the correct assessment of blame. Therefore, both sets of weights are updated.

The update rule for both the positive and negative reinforcement cases may be expressed by:

$$\Delta W_{ij} = \alpha R I_i G_j W_{ij}$$

$$\Delta W'_{jk} = \alpha' R G_j \hat{A}_k W'_{jk}$$

where:

α and α' are learning rate constants.

$\hat{A}$ is a vector in which all elements are 0, except for the kth element, where k is the winning action.

$I_i G_j W_{ij}$ and $G_j \hat{A}_k W'_{jk}$ are measures of the participation of specific synapses in the last decision (synapses between input unit i and feature detector j and feature detector j and action unit k, respectively).

In order to approach the problem of temporal credit assignment, we make use of the concept of eligibility, which was first introduced by Klopf (Klopf, 1982) and later used by others, including (Barto, et al., 1983). The eligibility of a weight is defined as a temporal memory of a synapse's participation in recent action decisions, and is computed as follows:

$$\tau \frac{d\,e_{ij}}{dt} = -e_{ij} + I_i G_j W_{ij}$$

$$\tau' \frac{d\,e'_{jk}}{dt} = -e'_{jk} + G_j \hat{A}_k W'_{jk}$$

where

e_{ij} and e'_{jk} are the eligibility measures.

τ and τ' determine the temporal width of the memory.

Finally, the weight updates become:

$$\Delta W_{ij} = \alpha R e_{ij}$$

$$\Delta W'_{jk} = \alpha' R e'_{jk}$$

By this definition of eligibility, the assignment of credit or blame for a reinforcement signal is given with the highest weight to the most recent decision that was made, and exponentially decreasing weight for decisions that were made further back in time. One key aspect of this definition of memory is that the number of memory elements does not depend upon the size of the time window over which the memories are stored, and only depends upon the number of synapses in the network.

Reinforcement Prediction

Eligibility provides a simple means by which credit can be assigned to a sequence of control decisions. However, the propagation of reinforcement information is limited to a fixed window of time defined by the decay of the eligibility memory. It is thus possible that a critical decision occurs outside of this window, and therefore would not receive any reinforcement information. As a result, the control network cannot learn to behave properly when an action and the corresponding reinforcement signal are separated by a length of time that is greater than that of the eligibility memory.

This problem is approached in this work through the method of reinforcement prediction (Sutton, 1988). First of all, suppose that we have a prediction network P(x(t)) that maps the current state of the system (x(t)) into a measure of the expected future reinforcement (relative to a fixed control policy). More explicitly:

$$P\big(x(t)\big) = E\left\{\sum_{\tau=t}^{\infty} \lambda^{\tau-t} R(\tau)\right\}$$

where:

$R(t)$ is the reinforcement received at time t.

λ is the discount factor for future reinforcement.

The function P(x(t)) can be viewed as the *goodness* of being in state x(t). Now observe that:

$$\begin{aligned} P\big(x(t)\big) &= E\left\{\sum_{\tau=t}^{\infty} \lambda^{\tau-t} R(\tau)\right\} \\ &= E\left\{R(t) + \lambda \sum_{\tau=t+1}^{\infty} \lambda^{\tau-t-1} R(\tau)\right\} \\ &= E\Big\{R(t) + \lambda P\big(x(t+1)\big)\Big\} \end{aligned}$$

And define:

$$R'(t) = R(t) + \lambda P(x(t+1)) - P(x(t))$$

$R'(t)$ can be interpreted as a measure of the deviation of the actual reinforcement received from that which was expected by the prediction network (for an individual time-step). In other words, if R' > 0, then the system performed better in the last time-step than it expected to perform; and when R' < 0, the system performed worse than expected. This measure can be used as an internally-generated reinforcement signal, that has the potential for delivering more meaningful reinforcement information at *every instant* that a decision is made, even when the teacher is providing a very sparse reinforcement signal.

It is important to note that the function $P\big(x(t)\big)$ is also relative to the control policy implemented by the control network. As this control policy adapts through experience, so must this prediction function. In this work, we make use of the method of *Temporal Difference Learning* (Sutton, 1988) to acquire the prediction function. This learning process is performed in parallel with the adaptation of the control network.

The R' that we compute from the non-stationary control policy has several important properties:

- The measure rewards incremental improvements in performance. When the controller discovers an action that leads to a higher level of performance than what was expected, the control network adjusts itself such that the probability of executing the same action given a similar situation is increased. This results in an overall improvement in the system's performance. The prediction network quickly adapts itself such that it expects this new level of performance. At this point, execution of the same action yields a null internal reinforcement signal, and a positive signal is only received if a new action further improves the performance.
- The above implies that even when the teacher only provides negative reinforcement as a way of specifying the desired behavior, the internal reinforcement signal can provide positive reinforcement information for actions that cause the system to receive less negative reinforcement than it was receiving earlier.
- More effective propagation of reinforcement information through time. Consider a controller

that has learned for all points in the set A (Figure 2) the correct sequence of actions to drive the system to point p, where it receives positive reinforcement from the teacher. Outside of set A (e.g. point z), however, the system does not know how to get reliably to point p, and in these cases, generates random actions. Once the system has visited the points in set A a number of times, the expected future reinforcement will become that which is received at point p (with some discount in the number of steps required to reach p). Thus, the internal reinforcement for moving from any of these points towards p will effectively be zero.

For the case of point z, the expected future reinforcement will be small, if not zero, and expected internal reinforcement will also be zero (because from point z, the system will tend to wander aimlessly outside of the set A). However, there is a some probability that an action chosen at state z will take the system into a point within A - i.e. from a state with low expected future reinforcement to a state with high expected future reinforcement. As a result, the internal reinforcement signal will be high, thus rewarding the action that brought the system into A. This is the case even though the actual reinforcement from the teacher is still several time-steps away.

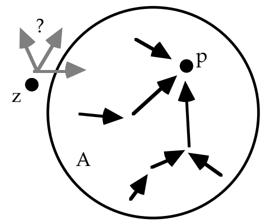

Figure 2: State space representation of a controller's actions. For those points within set A, the controller knows the correct sequence of actions to drive the system to point p, where positive reinforcement is received. However, outside of set A, the system has not yet learned the correct actions. By using R' as the reinforcement signal, the system is able to deliver a large amount of positive reinforcement to network when the boundary is crossed from point z to a point in set A. This is the case even though reinforcement from the teacher is still several time-steps away.

• Prevention of over-learning. If the external reinforcement signal is used to update the control network, weight updates will continue to be made when positive reinforcement is received, even if the network is already completely committed to the correct action. This can cause problems in terms of unnecessary interference with learning in other nearby regions of the state space. By using R' as the reinforcement signal, the system only adjusts its weights as long it is necessary to ensure commitment to the correct action(s), and then learns no further. Thus, other regions of the state space can be learned more easily, and more neural hardware can be reserved for learning during later experiences.

The updated network architecture is shown in Figure 3. The prediction network is a linear neural network, with the feature detector activity vector (G) serving as the input. The environmental reinforcement signal (R) is now combined with system's prediction of reinforcement to generate an internal reinforcement signal (R'). This signal is not only used to update the connections involved in the action selection process (W and W'), it also serves as an update signal for the prediction network (P).

Experimental Results

In this section, we illustrate the behavior of the system through a series of learning experiments. At this time, we are especially interested in understanding how the reinforcement policy affects the learned behavior.

Experiment 1: Collision Avoidance

Up to this point, the collision avoidance problem has been the primary focus of our experimentation. Several different reinforcement policies have been explored:

1. Punish running into an obstacle. When this policy is used, the network has a difficult time determining which action is appropriate, because it is only being told when an action was a bad choice. As a result, the system must spend a significant amount of time searching for the action that will produced the desired behavior.

2. Reward not running into an obstacle. The network often quickly discovers a one-move local minimum that satisfies the basic requirements of the behavior. An example of this is always making a forward right turn, causing the robot to move in a circle. However, the network very quickly and completely commits itself to this simple strategy, and when later faced with a new situation (such as an obstacle in the path), it is very difficult for the robot to explore alternative actions.

3. Punish for running into a wall and reward for not. Two sub-cases are possible:

3A. $|$**Punishment**$| < |$**Reward**$|$. This type of policy enables the robot to develop short cycles of movements, which result with overall positive reinforcement. When faced with an obstacle, a very common behavior that is learned is one in which the robot moves forward, collides with the obstacle, backs up one step, and then repeats the process. Even though this strategy receives negative reinforcement for the collision, it makes up for it by insuring that it receives a higher degree of positive reinforcement at the next time-step.

3B. |Punishment| > |Reward|. This policy produces control programs that perform the best for the collision avoidance problem. This is due to a balance between policy 1 (from above), where learning requires a long time, and policy 2, where learning happens so quickly that the system does not have adequate opportunity to explore the control space.

Figures 4 and 5 show the learning results of using policy 3B to solve the collision avoidance problem. Figure 4 tabulates a number of the feature detectors that were commonly learned over several experiments, and the actions that these feature detectors supported. In many cases, the input units that activate the feature detectors logically match the preferred actions.

Figure 5 shows the actual and internal reinforcement signals as learning takes place. We see a significant improvement in performance up through about 300 time-steps (as measured by the actual reinforcement curve). Also note that the internal reinforcement begins to deviate from the actual reinforcement curve at about 100 time-steps, demonstrating that the reinforcement predictor has begun to learn something useful. By the 600th time-step, this curve has leveled off, indicating that the predictor has learned to anticipate reinforcement to the best of its ability, and that the controller learning has stopped.

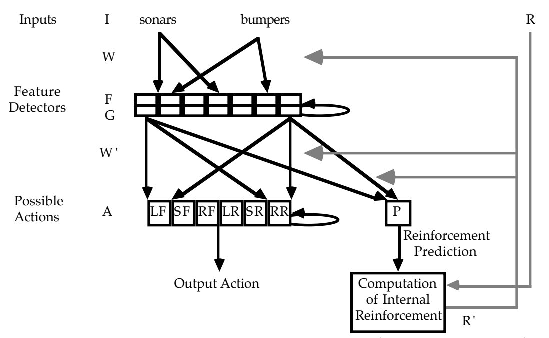

Figure 3: Modified network architecture. The reinforcement information from the teacher is first combined with the output of the Predictor Network to produce an internal reinforcement signal, R'. It is this signal that is used to update the weights in the control network. The Predictor Network (P) is implemented as a linear function of the feature detector outputs (G).

Sensory Inputs	Supported Actions
Mid Range Left Sonar Far Range Left Sonar Far Range Up Sonar	Straight Forward
Front Bumper Near Range Forward Sonar Mid Range Up Sonar Far Range Up Sonar	Straight Reverse
Far Range Up Sonar	Straight Forward
Rear Bumper Far Range Up	Right Forward Left Forward
Near Range Forward Sonar Mid Range Front Sonar Near Range Right Sonar	Left Reverse

Figure 4: Common feature detectors and the actions for which they vote. Over a set of several learning experiments (policy 3B), these rules (with some variation) were consistently learned.

Experiment 2: Environmental Exploration

In this experiment, the goal is to develop a behavior that causes the robot to cover a large area of its environment. The policy reinforces this behavior by rewarding the robot for moving in a straight forward direction, and punishing reverse movements. As it is still important to avoid obstacles, this rule is combined with policy 3B.

It is possible to assign different weights to each of these two sub-policies to reflect learning priorities. It was observed that the sub-policy given higher priority was learned faster than the other and typically dominated the other.

Experiment 3: Wall Following

A reinforcement policy for this behavior which has been posited is to reward the network when it moves in the straight forward direction while an obstacle is detected in the mid range of the left sonar. This policy is designed to encourage the robot to follow straight sections of walls. It is our belief that the reinforcement predictor network can provide sufficient information to internally reward the robot for properly turning corners when they are encountered. This is the subject of our ongoing experiments.

Discussion

This work has drawn on the results on TD (Temporal Difference) Learning of Sutton (Sutton, 1988), as well as those on Q-Learning (Barto & Bradtke, 1991; Watkins & Dayan, 1992). The primary difference with our algorithm is in its more neural orientation, and in the type of predictive information that is stored at each state. In our case, we only store the expected future discounted reinforcement (Watkin's V() function), as opposed to the expected future reinforcement relative to the next action to be taken (Q-values). The information provided by the Q-values is given to some degree by the activity levels of the output units, in that those actions that have the higher Q-values will tend to acquire higher and higher activity levels as learning progresses.

Our primary contribution is an attempt to identify more efficient coding schemes for state space representation. In this case, efficiency constitutes the amount of experience necessary to yield a competent reactive policy, and the amount of time required to actually learn the policy. For the experiments described above, learning was performed in real time (as the actions were being executed by the robot), and robot was allowed at most 30 minutes to acquire its program (about 1000 time-steps). These results are possible by a combination of the reinforcement predictor (as discussed earlier), and the input state coding scheme used at the feature detector layer.

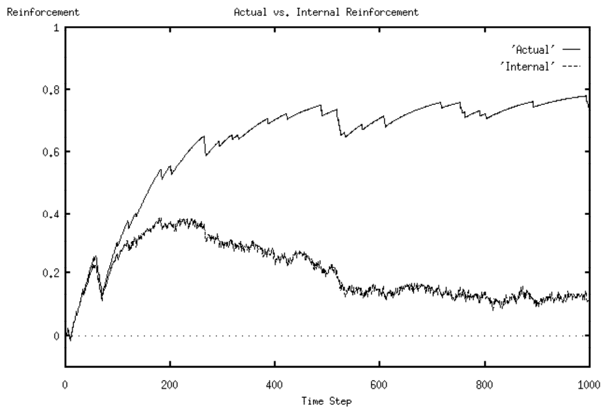

Figure 5: Actual (upper curve) and internal reinforcement (lower) over the course of a single learning collision avoidance experiment.

The local winner-take-all operation at the feature detector layer forces moderately different input activity patterns to activate different sets of feature detector units. As a result, learning that is performed in one region of the state space tends not to interfere with learning in other regions of the space, as is the case with Backpropagation-based reinforcement learning algorithms (Williams, 1987).

One the other hand, local support given by winning feature detectors to nearby neighbors tends to create a topological mapping of the input space (von der Malsburg, 1987). As a result, points that are nearby each other in the input space will activate largely overlapping sets of feature detectors, allowing these points to share their common experiences, yielding some degree of generalization.

State space representation for reinforcement learning as applied to robots has also been addressed in (Mahadevan & Connell, 1992) using statistical clustering techniques.

Conclusions

Reactive or behavior-based systems have been offered as alternative approaches to the more traditional AI-based techniques for the design of intelligent control systems for autonomous robots. Their ability to deal with uncertainties in sensing and in action generation, as well as their limited computational requirements have allowed them to demonstrate some level of success beyond that of AI systems. However, these systems are constructed in a trial-and-error fashion, and can often require long periods of programming time to reliably achieve the desired behavior.

This paper has presented an approach to constructing reactive control systems that allows the programmer (or teacher) to specify only the desired behavior of the program. This specification comes in the form of a reinforcement policy (rewards and punishments for certain behaviors that the robot exhibits), from which the learning algorithm must infer a control policy that attempts to maximize the rewards and minimize the punishments that are received. Specifying programs in this way is useful because the specification happens at a level that is easy for the programmer to express and understand, and yet the programs are evaluated within the environment in which they must ultimately perform.

The problem of programming however has not gone away. It has only shifted from specifying sets of reactive rules to specifying reinforcement policies that lead the learning system to discover sets of rules that accomplish the desired task. As was demonstrated in our experiments described above, this second process is also an iterative one, where the final behavior is observed, the reinforcement policy is adjusted, and the learning process is then repeated.

The current challenge is to construct principles from which reinforcement policies can be designed, such that the time required to learn a desired behavior is minimized. With this goal in mind, two directions are being pursued:

- **Staged Learning** is a technique in which partial solutions to a problem are first taught to the robot before it is expected to solve the entire problem (Lewis, Fagg, & Solidum, 1992; Lin, 1993). This is accomplished by first presenting the robot with a reinforcement policy that encourages the development of the partial solutions. Once the robot has obtained this intermediate level of capability, the reinforcement policy is changed such that the robot is then required to solve the entire problem in order to receive positive reinforcement. This technique can be used to lead the robot along a specific path of learning, significantly reducing the amount of search that is necessary to discover a program that produces the desired behavior. The behavior-based decompositions for the learning system described in (Mahadevan & Connell, 1992) also has some interesting parallels to staged learning.

- **Learning by Demonstration** is an approach in which motor programs are demonstrated to the robot by the human (Fagg, 1993; Handleman & Lane, 1993; Lin, 1993; Liu & Asada, 1992). The robot first learns to mimic the behavior of the human through a more supervised learning approach. Reinforcement learning is then applied to further refine the learned motor programs such that they are better matched with the robot's own sensory and actuation abilities.

Acknowledgments

The authors would like to thank Dr. Michael Arbib, Gaurav Sukhatme, Mike McHenry, Amanda Bischoff, Nicolas Schweighofer, Tony Lewis, Jason Bagley, Parag Batavia, Anand Ramakrishna, and Mathew Lamb for their comments, suggestions and help during the course of this work. The neural model has been implemented in NSL (Neural Simulation Language (Weitzenfeld, 1991)), which is available via anonymous ftp from yorick.usc.edu. More information may be obtained by contacting Alfredo Weitzenfeld (alfredo@rana.usc.edu). This work has been supported in part by grants from the University of Southern California Graduate School, School of Engineering, and Computer Science Department, and by the Human Frontiers Science Program.

References

Arbib, M. A. (1989). Schemas and Neural Networks for Sixth Generation Computing. Journal of Parallel and Distributed Computing, **6**, 185-216.

Arkin, R. C. (1990). Integrating behavioral, perceptual, and world knowledge in reactive navigation. Robotics and Autonomous Systems, **6**, 105-122.

Barto, A. G., & Bradtke, S. H. (1991). Real-Time Learning and Control using Asynchronous Dynamic Programming (TR No. 91-57). Department of Computer Science, University of Massachusetts, Amherst.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. IEEE Transactions on Systems, Man, and Cybernetics, **SMC-5**, 834-46.

Bekey, G. A., & Tomovic, R. (1986). Reflex Control of Robot Actions. In IEEE International Conference on Robotics and Automation, (pp. 240-247). San Francisco.

Brooks, R. A. (1986). A Robust Layered Control System for a Mobile Robot. IEEE J. Robotics and Automation, **RA-2**(RA-2), 14-23.

Brooks, R. A. (1991). Intelligence Without Reason (A.I. Memo No. 1293). MIT.

Fagg, A. H. (1993). Developmental Robotics: A New Approach to the Specification of Robot Programs. In G. A. Bekey & K. Y. Goldberg (Eds.), Neural Networks in Robotics (pp. 459-86). Norwell, Massachusetts: Kluwer Academic Publishers.

Fagg, A. H., & Arbib, M. A. (1992). A Model of Primate Visual-Motor Conditional Learning. Journal of Adaptive Behavior, **1**(1), 3-37.

Handleman, D. A., & Lane, S. H. (1993). Fast Sensorimotor Skill Acquisition Based on Rule-Based Training of Neural Nets. In G. A. Bekey & K. Y. Goldberg (Eds.), Neural Networks in Robotics (pp. 255-70). Norwell, Massachusetts: Kluwer Academic Publishers.

Klopf, A. H. (1982). The Hedonistic Neuron. Washington, D. C.: Hemisphere.

Lewis, M. A., Fagg, A. H., & Solidum, A. (1992). Genetic Programming Approach to the Construction of a Neural Network for Control of a Walking Robot. In Proceedings of the 1992 IEEE Conference on Robotics and Automation, (pp. 2618-23). Nice, France:

Lin, J. J. (1993). Hierarchical Learning of Robot Skills by Reinforcement. In Proceedings of the 1993 IEEE Conference on Neural Networks, (pp. 181-6). San Francisco, California: IEEE.

Liu, S., & Asada, H. (1992). Transferring Manipulative Skills to Robots - Representation and Acquisition of Tool Manipulative Skills Using a Process Dynamic Model. Journal of Dynamic Systems Measurement and Control-Transactions of the ASME, **114**(2), 220-8.

Maes, P., & Brooks, R. A. (1990). Learning to Coordinate Behaviors. In AAAI, (pp. 796-802). Boston, MA.

Mahadevan, S., & Connell, J. (1992). Automatic Programming of Behavior-Based Robots Using Reinforcement Learning. Artificial Intelligence, **55**, 311-365.

Samuel, A. L. (1967). Some Studies in Machine Learning Using the Game of Checkers. IBM Journal of Research and Development, **11**, 601-17.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. In Machine Learning 3 (pp. 9-44). Boston: Kluwer Academic Publishers.

Verschure, P. F. M. J., & Kröse, B. J. A. (1992). Distributed Adaptive Control: The Self-Organization of Structured Behavior. Robotics and Autonomous Systems, **9**, 181-196.

von der Malsburg, C. (1987). Ordered Retinotectal Projections and Brain Organization. In F. E. Yates, A. Garfinkel, D. O. Walter, & G. Yates (Eds.), Self-Organizing Systems - The Emergence of Order (pp. 265-78). New York: Plenum Press.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-Learning. Machine Learning, **8**(3), 279-92.

Weitzenfeld, A. (1991). NSL - Neural Simulation Language Version 2.1 (TR No. TR 91-05). Center for Neural Engineering, University of Southern California, Los Angeles, CA.

Williams, R. J. (1987). Reinforcement Learning Connectionist Systems (TR No. NU-CCS-87-3). Northeastern University, College of Computer Science.