Emergent Control and Planning in an Autonomous Vehicle

To appear in *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*

Lisa Meeden[†] and **Gary McGraw**[*†] and **Douglas Blank**[†]
[†]Department of Computer Science
[*]Center for Research on Concepts and Cognition
Indiana University
Bloomington, Indiana 47405
meeden@cs.indiana.edu gem@cogsci.indiana.edu blank@cs.indiana.edu

Abstract

We use a connectionist network trained with reinforcement to control both an autonomous robot vehicle and a simulated robot. We show that given appropriate sensory data and architectural structure, a network can learn to control the robot for a simple navigation problem. We then investigate a more complex goal-based problem and examine the plan-like behavior that emerges.

1. Autonomous agents

An *autonomous agent* can be abstractly defined as a mapping from a sequence of sensory inputs to an appropriate action in response to these percepts. Such an agent is *autonomous* to the extent that its behavior is determined by its immediate inputs and past experience, rather than by its built-in control [Russell and Wefald, 1991]. We are interested in investigating the cognitive capabilities of autonomous agents. We believe that cognitive behavior can emerge from the reactive, situated activity of autonomous agents.

Some consensus exists about how to design autonomous agents. There should be a relatively direct coupling between perception and action, control should be distributed and decentralized, and most importantly, there should be a dynamic interaction between the environment and the agent [Maes, 1990]. However, no such consensus exists on the best method to implement these design features. Connectionist networks can easily accommodate all these design features and we believe they can be effective mechanisms for controlling autonomous agents.

This paper focuses on exploring connectionist designs for controlling simple navigation in an autonomous vehicle. We conclude by applying the successful design features to a more difficult problem and examine the plan-like behavior that emerges.

2. Methodology

Given that a connectionist controller will be used, there are still a number of implementation questions to be resolved. First, what sorts of sensory abilities will the controller need to simply react effectively to the environment? Second, what sort of memory or training subtasks would enable the controller to produce more complex responses to the environment? Third, how should the controller be trained so that the behavior emerges rather than being specified explicitly?

We examine these implementation questions by testing network controllers for both a real and simulated robot in a very simple environment, using as experimental variables both type of sensory data, type of training subtasks, and amount of memory. To train the network controllers, we use a reinforcement learning algorithm which converts abstract measures of goodness (reward and punishment) into specific teacher signals.

The environment, called the "playpen", is a rectangular box (2×4 feet) with a light in one corner. The reinforcement training problems we investigate involve coordination of motor activity with the information being supplied by the sensors. The navigation problem, which we refer to as *avoid and move*, reinforces the robot for moving in the playpen while avoiding the walls. The more difficult problem, *light as food*, adds a goal state in order to simulate hunger, and reinforces the robot for periodically seeking and avoiding the light while still avoiding the walls and moving.

By holding a problem constant and varying sensory ability we can determine to what extent the addition of more sensors helps the robot succeed at its problem. Similarly, we can evaluate the utility of contextual memory and different training subtasks such as auto-association and prediction.

3. Carbot—an autonomous robot

3.1 The autonomous vehicle

Our robot, called carbot, is a modified toy car (6×9 inches) controlled by a programmable mini-board (designed by [Martin, 1992]). Carbot was inexpensive to build, primarily because it makes use of primitive sensors—no lasers, video, or sonar. It has two servo-motors; one controls forward and backward motion and the other steering. The robot has two types of physical sensors: digital touch sensors on the front and back bumpers, and analog light sensors on stalks near the back. The light sensors are directed 30 degrees to each side of carbot.

3.2 The control networks

Carbot is controlled by a remote connectionist network that communicates with the mini-board. The network gathers input data from the sensors and determines how to set the motors for the next time step. Figure 1 shows the standard network used in our experiments.

There are four discrete sets of input units; three are for sensors and one is for context memory. The first set represents the previous state of the two motors—two units per motor. The first motor unit represents the spin direction of the rear motor (this determines direction of motion—forward or backward). The second unit designates the state of the motor as on or off. The third unit represents the spin direction of the front motor (this determines the direction of turning—left or right). Note that in order to turn carbot must have both motors running—the back motor provides movement while the front motor steers. The fourth unit designates the state of the front motor as on or off.

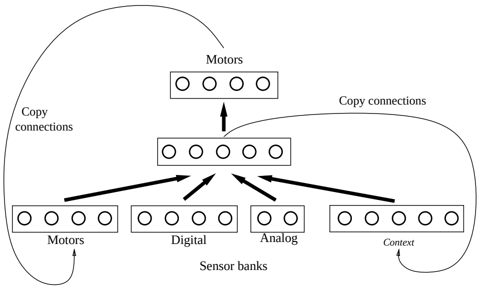

Fig. 1. The standard control network. Bold arrows indicate that the units are fully connected. Note that some experiments were conducted without any context units and/or without any sensor units.

The next set of units in the input layer represent the state of the digital touch sensors. There are four digital sensors—three in front and one in back. Two of the front sensors are out to either side so that carbot can sense side collisions when moving forward. The next two units represent the state of the analog light sensors, whose values can range from 0.0 to 1.0 (due to ambient light, their values rarely fall below 0.25). One is for the right sensor, the other for the left.

The context units are present in most, but not all of the experiments. Activations from the hidden layer on the previous time step are copied into the context units directly. The simple recurrence of the context units allows the network to have a limited short-term memory of its past states [Elman, 1990]. We refer to the simple recurrent networks as SRNs and the feed-forward networks (without context memory) as FFNs.

The four output units of the standard network determine what the activity of the motors will be for the next time step. We show a recurrent connection between the output motor units and the input motor units because the robot has no physical sensors to tell what its motors are doing. Since the network controls the motors, the information from the last time step can be easily copied down to the input layer and considered an additional type of sensor.

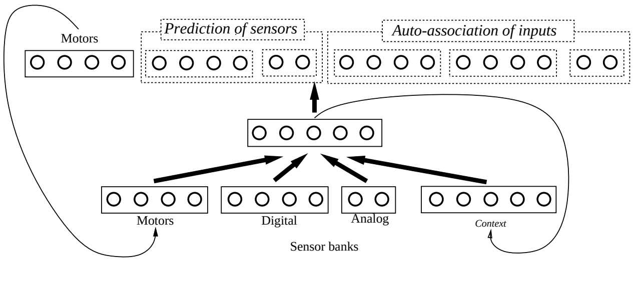

Fig. 2. To add prediction to the network, six additional output units representing the predicted sensor information were required. To add auto-association, ten more output units representing the current state of the motors and the sensors were required. The most complex network we investigated (the one shown) includes both prediction and auto-association.

Part of our research involved determining the effect of adding specific training subtasks to the controller network such as auto-association and prediction. Figure 2 shows how these subtasks are included. In general, auto-association forces the network to pay more attention to its inputs (since it must learn to duplicate the input activations on the output layer). Prediction may help the network build a more complex model of its environment so that it can, for instance, avoid punishment by predicting that it may hit a wall during the next time step. Another portion of our research involved determining the effect of varying the size of the contextual memory.

3.3 Reinforcement training

Because autonomous agent problems are typically defined in terms of abstract goals rather than specific input to output pairs, some type of reinforcement procedure is required for learning. For example, in the *avoid and move* problem, suppose that carbot has just bumped into a wall triggering its front sensors. Any action that moves it away from the wall and clears its sensors should be rewarded, while any action that persists in bumping into the wall should be punished. There is not necessarily one "right" action for a given situation, and even if there were, it might not be known *a priori*.

In all of our experiments, the control networks were trained with a modified version of the complementary reinforcement back-propagation (CRBP) learning algorithm [Ackley and Littman, 1990]. Back-propagation learning requires precise error measures for each output produced by a network. CRBP provides these exact error measures from the abstract reward and punishment signals as follows.

A forward propagation of the input values produces a real-valued search vector S. Each of these activations is interpreted as the probability that an associated random bit takes on the value 1. From these probabilities a binary output vector O is stochastically produced. If O is rewarded, then learning should push the network towards this vector, so the error measure $(O - S)$ is back-propagated. If O is punished, then learning should push the network away from this vector, but the appropriate direction is not clear. CRBP chooses to push the network directly toward the complement of O, using the error measure $((1-O)-S)$. In this way rewarded outputs will be more likely to occur again and punished outputs will tend to produce the complement output vector in similar situations.

CRBP was designed for static problems. Because carbot's problems are temporally extended, we needed to modify the algorithm. To the best of our knowledge, this work is the first application of CRBP to a continuous problem. The original version of CRBP uses a learning rate ten times greater for reward than for punishment to reflect the informativeness of the associated errors. But in carbot's domain, the system gets rewarded for simply moving in the environment, so the reward to punishment ratio is much larger than for static problems, making such a large disparity in learning rates infeasible.

To side-step this problem, we empirically determined that a reward learning rate only three times the punishment learning rate worked well for our dynamic domain. In the experiments reported below, we used a reward learning rate of 0.3, a punishment learning rate of 0.1, and no momentum for training the motor outputs. Any additional outputs for which an explicit target was available (*e.g.*, auto-association and prediction), were trained with a fixed learning rate of 0.5.

3.4 Evaluating global behavior

After running many preliminary tests with carbot, we found one global metric that provide a good measure of overall behavior—the percent time punished. We used this measure for the graphs shown in this paper as well as for the statistical analyses. All of the statistical analyses reported below make use of analysis of variance (ANOVA) testing. Post hoc significance comparisons were done using Scheffé tests.

It is important to keep in mind the temporal nature of our experimental problems. Analyzing one particular time-slice of behavior is not useful. Instead, behavior over some large range of time-slices must be considered. We found that most *avoid and move* networks tended to converge (*i.e.,* learn a successful strategy) within the first 3000 cycles of a run. We chose to consider these first cycles the "training phase". We used the next 2000 cycles of a run (during which learning continued) as the "performance phase". We gathered data during the performance phase in order to evaluate carbot's behavior.

3.5 Training in the real world

Since the robot had to physically move in the world, bumping into things, it took a long time to train and test a particular controller network (approximately two and a half hours for a 5,000 cycle run). To get around the time problem we decided to simulate the behavior of the robot in software. We are aware that simulators are often too clean and not very much like the real world, but since we actually have a real robot we were able to test the correspondence of our simulator with the real-world behavior of the robot. We even did a series of "brain-transplant" experiments to verify that the sorts of networks that worked well in the simulator would work well in the robot.

4. The carbot simulator

The simulator is a C program using the same controller networks as the robot. We empirically determined the average turning radius and distance traveled by carbot in the playpen. We used these averages in the simulator, adding small amounts of random noise to its heading, position, and analog sensor readings on every time step. To test the accuracy of the simulator we transplanted trained networks from the simulator to the robot and from the robot to the simulator, and then compared the behavior in terms of percent time punished.

4.1 Transplanting controller networks

Transplant tests provided a surprising result. The robot's behavior on any trained network (trained either in the simulator, or in the robot) is always superior to the simulator's behavior. We suspect that this is because the robot's movements are not noisy in the same way as the simulator's. The robot's movement is only occasionally noisy while the simulator systematically adds noise on every time step. Interestingly, this added noise seems to be beneficial to training. Just as it is useful for runners to train with weights on their legs and then run without them during competition, it seems useful to have more noise during simulation training than is actually present during robot testing. Further investigation into the benefits of a noisy simulator is needed. We would like to compare controllers trained in the simulator without noise to controllers trained with noise.

The simulated robot's behavior is close enough to the actual behavior of carbot to warrant use of the simulator for research. Our methodology is to use the simulator to test hypotheses and develop useful architectures that we then apply back to the robot. This saves many hours since a typical 5,000 cycle run on the simulator takes less than a minute (a speedup factor of 150).

5. Experiments in the simulator

We ran several groups of experiments with the simulator. For each experimental variable described below, 20 trials were run. For each trial, the network controller being investigated was initialized with a random set of weights and run for 5000 cycles. Performance was evaluated during the last 2000 cycles of each trial producing an average percent punishment. These averages were then averaged over all 20 trials. We then applied ANOVA to these composite averages in order to determine the significance of the variable.

There was a large amount of variation across the 20 trials for a particular variable (the standard deviations ranged from 4 to 13). We believe that the primary source of this variation was the random initial weights. Some sets of random weights tended to be quite bad while others were quite good. Unfortunately the large variations in the results may have tended to obscure significant effects in the experiments.

5.1 The *avoid and move* problem

Recall that for this problem the robot was rewarded for avoiding walls while constantly moving. There are many possible successful strategies for this simple navigation problem. The easiest solution is to continually turn in one direction forming a circular path which avoids all the walls. The limited size of the playpen made this solution impossible since carbot's turning radius was larger than the smallest dimension (although this was a frequent result in larger environments not reported on here). Another simple solution is to oscillate one step forward then one step backward, this proved to be the most frequent solution in the experiments reported here. Since carbot's only constraints were to keep moving and avoid walls, there was no impetus to explore the environment or to develop more complex patterns of behavior.

5.1.1 Baselines

To determine the baseline behavior of the simulator we did not use a controller network, but instead set the motors randomly. Since there was no network to be trained, there was no learning. This resulted in an average punishment of 67.29%.

To determine the baseline behavior of the network architectures, we ran two sets of tests, one with FFNs and the other with SRNs. Both types of networks had access to the sensor data, but did not learn. We wanted to see how well network controllers with random weights would perform. The average punishment rates were 75.47% for the FFN model and 72.60% for the SRN model. The differences between either of the non-learning network baselines vs. the simulator baseline are both significant ($p < 0.01$). This is interesting because it shows that untrained network controllers perform worse than random. Fortunately learning alleviates this.

5.1.2 Varying sensory data

To discover the effect that the addition of sensor data has on controlling ability, we tested 16 types of networks. Starting with either a FFN or an SRN (with 5 context units) we systematically added all combinations of sensors: analog (A), digital (D), and motor (M).

Figure 3 depicts the results of these variations. It shows average convergence histories of the 16 types of networks, one curve for the 20 trials of each type. This kind of graph gives a sense of the learning as it progresses in time. Data points for the graphs were calculated by finding the percentage of punishments in 500 cycle bins.

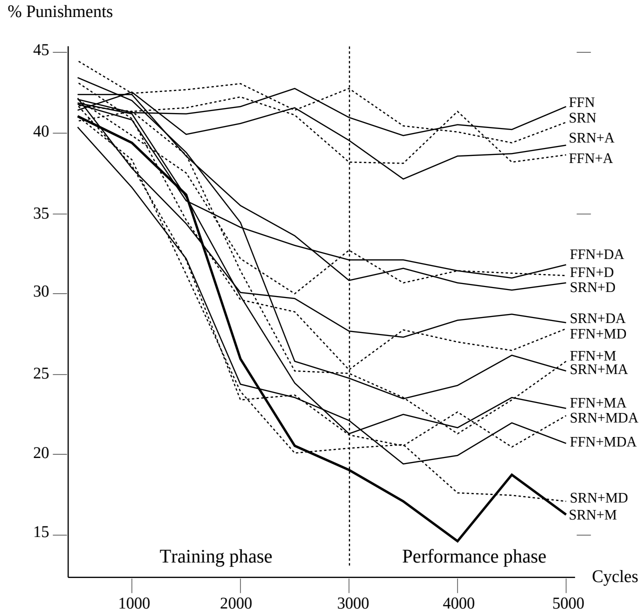

Fig. 3. Convergence histories for the 16 types of networks used to investigate the utility of sensory input. Each curve represents the average performance of 20 networks. To the right of each curve is a label denoting the category of the network. The labels make use of the following convention: digital sensors (D), analog sensors (A), motor sensors (M).

Percentages of punishment in each bin were plotted and connected to form a curve. As noted above, we divided each run into two phases, a training phase and a performance phase. These phases are marked on the graph. The statistical analysis was run on the averages over the performance phases.

As seen in Figure 3, the sensory input networks can be divided into three groups. Starting at the top of the graph, the first group is made up of the first four types of network (which all failed to converge). These are networks of both types (SRN and FFN) with either no inputs or analog inputs alone. The next group is made up of networks without motor sensors but including digital, analog, or both. The most successful group includes all the networks with motor sensors. Of all the 16 types of networks, the best is an SRN with motor sensors alone.

Through statistical analysis we found that there is a significant difference between the SRN and FFN models for the sensory input tests ($p < 0.05$). On average the SRNs were punished 27.31% of the time while the FFNs were punished 29.53% of the time. However, there is no significant interaction between the architecture types and the inputs. This is also apparent from the convergence graph.

The analysis confirmed that networks with access to motor sensors far outperformed networks without motor sensors. In pairwise comparisons, networks with motor inputs alone were significantly better than those with no input, digital input alone, or analog input alone ($p < 0.01$). Motors alone were also better than digital and analog together ($p < 0.05$). Although there was no significant difference between digital alone and analog alone, combined they are better than analog alone ($p < 0.01$).

It is interesting that motors, digitals, and analogs together are not significantly better than motors alone. Intuitively we expected that the more perceptual input available, the better the controller would be. This may be true for increasingly complex problems, but clearly for this problem, just having access to the previous motor settings alone is very informative. Recall that the motor sensor values are provided through a recurrence from the output layer to the input layer (see Figure 1). This simple, one-step memory is probably what makes the motor sensors so much more useful than the other sensors.

5.1.3 Varying training subtasks

To discover the effect that adding training subtasks has on controlling ability, we tested four types of networks. Each was an SRN provided with all the sensory data (motors, digitals, and analogs). Figure 4 shows the convergence histories of the four architecture networks. The best two networks (at the bottom of the graph) both have prediction units.

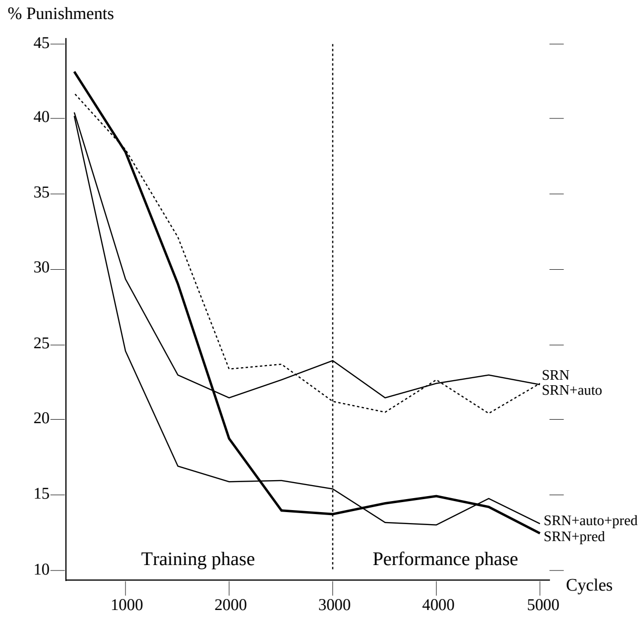

Fig. 4. Convergence histories of the four types of networks used to determine the utility of training subtasks. Some networks have prediction units (pred), others have auto-association (auto).

Although there appears to be a substantial advantage for networks with prediction, no significance is found when all four networks are compared (the standard deviations are greater than 10). However, if the comparison is changed from a four-level comparison to a two-level comparison (between networks with prediction vs. networks without prediction), the result is significant ($p < 0.01$) with the average time punished being 13.75% vs. 21.88%.

Forcing the network, through auto-association, to pay more attention to its perceptual input is not enough to improve performance. Yet having to predict the subsequent input is extremely useful for learning navigation control. Learning to mimic the input is a static problem, while predicting the next input is a temporal problem. Thus it appears that the network does not use its context memory effectively unless its training subtask explicitly depends on temporal information.

5.1.4 Varying contextual memory

To discover the effect that the size of the contextual memory has on controlling ability, we tested seven types of networks, varying memory size from 0 to 50 units. Again, each of the networks had access to all of the available perceptual data, including motors.

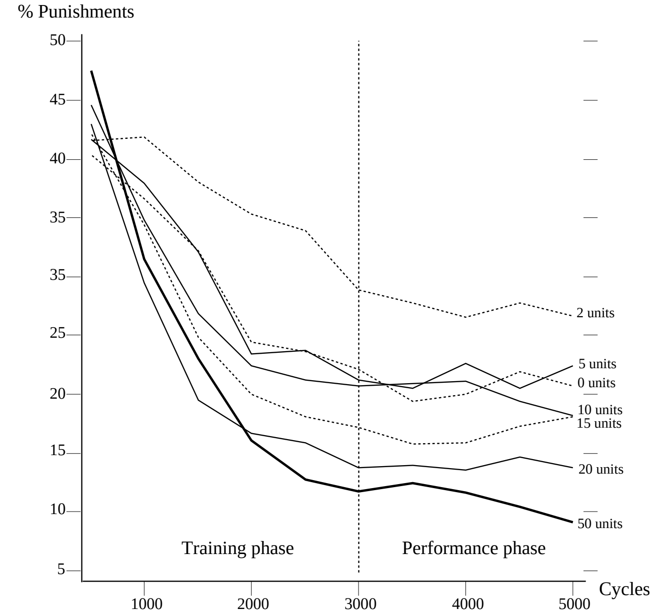

Fig. 5. Convergence histories of the seven types of networks used to determine the utility of contextual memory.

Figure 5 shows the convergence histories of the seven types of network tested for memory utility. The graph shows that above a certain memory size (somewhere between 5 and 10 units) more memory seems useful. However, the only significant pairwise results were size 20 vs. size 2 ($p < 0.05$) and size 50 vs. size 2 ($p < 0.01$). There may be a point at which too large a memory becomes disadvantageous, but we did not have the resources to explore this question.

In these experiments, pairwise comparisons between SRNs with up to 50 units of context memory are not significantly better than FFNs with no context memory at all (shown as 0 units on the graph). This result is somewhat surprising. Although the *avoid and move* problem does not require temporal information, we expected that access to temporal information would be a significant benefit in learning control. This result can be explained by the findings from the input experiments. We noted earlier that the motor inputs are really a form of recurrence from the output layer to the input layer. They provide information about the previous time step and are therefore temporal in nature. This means that the FFN model is not strictly feed-forward.

In summary, the *avoid and move* experiments revealed that, contrary to our intuitions, more is not always better (at least for this simple navigation problem). For the sensory input, the past motor states were the most informative, followed by the digital touch sensors, and finally the analog light sensors. Using all three types of input was not better than using the motor sensors alone. With respect to training subtasks, prediction was a significant benefit, but auto-association did not seem to be useful. Finally, for the contextual memory size, more actually was better, although there may be some limit to this improvement. In the next set of experiments we apply these design insights to a more difficult problem.

5.2 The *light as food* problem

Since we believe that abstract reasoning abilities, such as planning, arise developmentally from concrete activity [Chapman and Agre, 1987], a connectionist, autonomous agent controller should also be able to exhibit plan-like behavior if given a more complex problem.

For this problem, a goal unit was added to the input layer. A positive value for the goal indicated that carbot should seek out the light (placed in one corner of the playpen) until a maximum light reading was obtained. Once this happened, the goal unit switched to a negative value, indicating that carbot should avoid the light until a minimum light reading was obtained. Successful avoidance switched the goal back to seek-mode again.

When in seek-mode, carbot was rewarded if the sum of its light sensor readings increased relative to the previous time step. When in avoid-mode, carbot was rewarded if the sum of its light sensor readings decreased relative to the previous time step. Carbot was concurrently trained on the *avoid and move* problem.

For these experiments we focused on one type of SRN model. The controller networks had a context memory of size 20, used both prediction and auto-association, and were provided with all the available sensory inputs. Again 20 trials were run.

Intuitively the *light as food* problem seems much more difficult than the *avoid and move* problem. Baseline experiments proved this to be true. Without learning, the controller networks for *light as food* were punished on average 90.42% of the time, while the *avoid and move* SRNs were punished 75.47%, which is a significant difference ($p < 0.01$).

The difficulty of this problem is also reflected in the amount of training required. After 100,000 cycles, the most successful network controller was still receiving punishment 46.11% of the time while the average performance was 50.71%. After another 100,000 cycles (for a total of 200,000), the best network improved to 17.78% performance while the average was 23.37%.

Throughout the training phases, there was a general trend observed in the distribution of the punishments for all the networks. To illustrate, we will describe this trend using one network. In the initial training phase, 36% of the punishment resulted from sensor hits, 12% from not moving, 43% from seeking the light incorrectly, and 9% from avoiding the light incorrectly. Note that it is easier to avoid the light than to seek it, since there are many positions in the environment which satisfy the minimum light requirement, but few that satisfy the maximum requirement. By the end of the second training phase, the distribution of punishments was substantially different, 55% from sensor hits, 2% from not moving, 24% from seeking the light incorrectly, and 19% from avoiding the light incorrectly. The controller has become almost as successful at seeking the light as avoiding the light.

6. Emergent planning

The same basic strategies were adopted by all 20 controller networks trained on the *light as food* problem. When in avoid-mode, first move away from the light, then orient carbot away from the light. When in seek-mode, the opposite strategy was used, first orient carbot toward the light, and next move to the light. See Figure 6 for examples of these strategies from an actual run of one network.

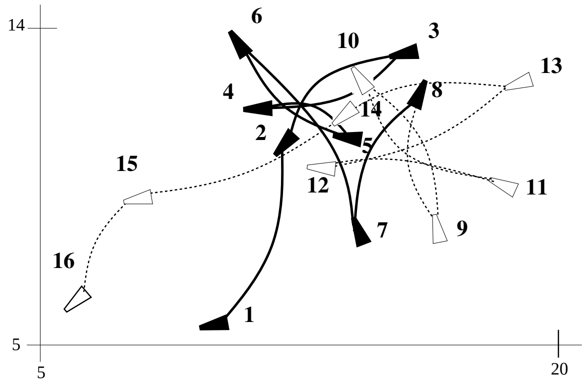

Fig. 6. Path of a simulated carbot through the playpen (units are inches). The light is located at the origin. The direction of the arrows indicate carbot's current heading. The numbers on the path refer to steps in a sequence of motion. 1-8 occurred during avoid-mode, 9-16 occurred during seek-mode. Note that it has satisfied its goals at steps 8 and 16.

To enact these strategies, when in avoid-mode, first carbot moved backward away from the light (steps 1-3). Then it alternated between moving forward turning left and backward turning right, until it was facing away from the light (steps 4-8). When in seek-mode, it alternated between moving backward turning right and forward turning left, until it was facing the light (steps 9-13). Then it moved forward towards the light (steps 14-16).

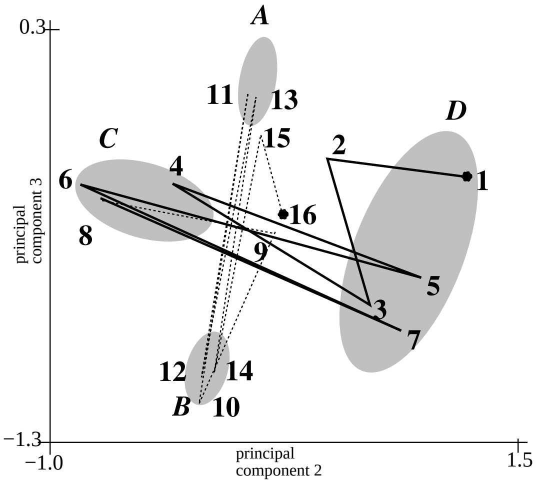

Fig. 7. Path of the network's internal states corresponding to the actions in Figure 6. The shaded regions highlight the most visited areas. When carbot is orienting itself toward the light in seek-mode, the network alternates between regions A and B. When carbot is orienting itself away from the light in avoid-mode, the network alternates between C and D.

To examine how the network has implemented these strategies we ran principal components analysis on the hidden layer activations from 5,000 cycles of the trained network. Whereas Figure 6 shows the path of carbot in the playpen, Figure 7 shows the path of the network's internal states at each of the same points in time. The two modes of carbot's behavior, seeking and avoiding the light, are distinct in the network's internal transitions as well. In addition, the two phases of each mode, orient toward then move forward or move backward then orient away, are also evident. Another interesting aspect of Figure 7 is that regions A and B are much more compact than C and D which reflects the fact that seeking the light (A and B) is a more constrained task than avoiding the light (C and D).

In summary, the behavior produced by the network controllers is plan-like in a number of ways. First, the controllers learned to associate abstract goals with sequences of primitive actions which occurred over time. Second, the behavior can be easily described hierarchically in abstract terms (see Figure 8). Finally, the controller can flexibly react to the environmental conditions while still maintaining its overall strategy. For example, if a wall is encountered during the move-toward-the-light phase of the seek strategy, then carbot suspends its current task of heading to the light by backing away from the wall, and then returns to moving forward towards the light.

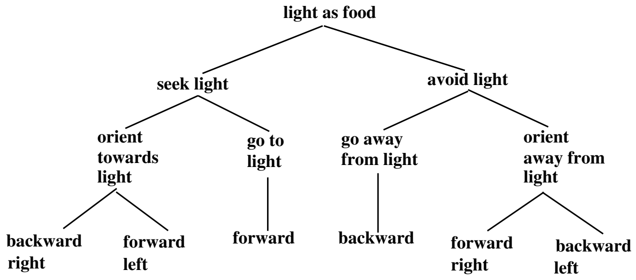

Fig. 8. A hierarchical view of carbot's behavior.

In other respects this behavior is not very plan-like, at least as planning is traditionally conceived. The controllers do not consistently anticipate and avoid punishment, nor attempt to minimize their resources to produce optimal behavior, and the number and complexity of strategies they exhibit is still minimal. In spite of these deficiencies, we believe that the multi-step, almost procedural behavior that emerged from the *light as food* problem is interestingly plan-like. Our results lend empirical weight to the argument that complex behavior can result from the low-level interaction of an autonomous agent with its environment.

7. Acknowledgements

We would like to thank Sven Anderson, Amy Barley, Laura Blankenship, Dave Chalmers, Mike Gasser, Jim Marshall, Devin McAuley, Jonathon Mills, John Nienart, Cathy Rogers, and Andy Strauss for their support during this research. Thanks also to Yoshiro Miyata and Andreas Stolcke for their cluster and principal component analysis program.

References

[Ackley and Littman, 1990] Ackley, D. H. and Littman, M. L. (1990). Generalization and scaling in reinforcement learning. In Touretsky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 550–557. Morgan Kaufmann, San Mateo, CA.

[Chapman and Agre, 1987] Chapman, D. and Agre, P. E. (1987). Abstract reasoning as emergent from concrete activity. In Georgeff, M. P. and Lansky, A. L., editors, *Reasoning about actions and plans: Proceedings of the 1986*

Workshop, pages 411–424. Morgan Kaufmann, Los Altos, CA.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179–212.

[Maes, 1990] Maes, P. (1990). Guest editorial: Designing autonomous agents. *Robotics and Autonomous Systems*, (6):1–2.

[Martin, 1992] Martin, F. (1992). Mini board 2.0 technical reference. MIT Media Lab, Cambridge MA 02139.

[Russell and Wefald, 1991] Russell, S. and Wefald, E. (1991). *Do the Right Thing: Studies in Limited Rationality.* MIT Press, Cambridge, MA.