Chapter 7

Behaviour Assessment

7.1 Introduction

This chapter is concerned with the assessment of the individual performance of each behaviour module as well as the overall performance of the controller with respect to the requirements on the target behaviour which were specified in the *Application Description* stage. This chapter begins with the definitions of the metrics that will be used to assess the learning process and the final behaviour of the integrated controller architecture. The learning performance of the obstacle avoidance module for both a Gaussian and uniformly distributed random exploration rule is assessed for both a real Khepera and a simulated Khepera. This is followed by a similar evaluation for the light seeking module (using the Gaussian random exploration rule only). Finally, the global behaviour of the integrated controller architecture with trained obstacle avoidance and light seeking modules is evaluated.

7.2 Behaviour Metrics

The BAT methodology proposes that the performance of the robot be evaluated quantitatively in terms of two types of performance index. A *local index* measures the effectiveness of the learning process and allows for its assessment. A *global index* measures the correspondence between the robot's *trained* behaviour with the *target* behaviour outlined in the chapter on Behaviour Analysis.

The local performance index proposed in the BAT methodology was defined as

$$L(t)=\frac{R(t)}{t}$$

where *R(t)* is the number of moves that have been positively reinforced from the beginning of the experiment and *t* is the total number of moves since the start. The proposed global performance index was defined as

$$G(t)=\frac{t}{A(t)}$$

where *A(t)* is the number of achievements from the beginning of the experiment (ie., the number of times the robot has reached the goal). If *A(t)* is a fixed prior to the beginning of the assessment, then *G(t)* is computed as the total number of individual moves or cycles required to achieve them.

The local performance index proposed in the BAT methodology was not deemed suitable for application in this work because it tracks the *cumulative* performance of the network in terms of *reward*. In the reinforcement programs for both light seeking and obstacle avoidance, the network outputs are compared to the target vector (as described in section 6.2.2) and if they are sufficiently close to each other, no reward is given and no network changes are made. If we assume that the robot has learned several actions so well that it receives no reward for them, and then we examine a plot of the local performance for some time period, we might see the graph level off in several places (remember the index plots *cumulative* reward). The difficulty is explaining *why* the graph levels off since there are two possibilities: either the robot has learned so well that rewards are not given or the robot is receiving punishments during those time frames.

In this work, the local performance metric used to measure the effectiveness of the learning process was the *percentage of punishments per epoch index*. This index (known from this point on as the *punishment index*) is very similar to that used in [meeden94] to evaluate and compare network performance versus network architecture. It is computed here by totalling the number of moves that were punished in each *n* move epoch and dividing by *n*/100 to obtain a percentage. Using punishments as a local performance index solves the problem described in the previous paragraph. When the system is not receiving a punishment, we may assume that it is either being rewarded or that the system has learned to the point that rewards are no longer required. Both instances are on the same end of the spectrum, thereby eliminating any ambiguity.

It is important to note, however, that the punishment index can be misleading since the lack of punishment *does not in itself indicate success*. For example, a robot that learns to turn in circles in order to avoid bumping into walls will likely boast a very low punishment index. Looking only at the graph of its punishment performance, one could conclude that the robot has learned to navigate its environment very well. Arguably, it does - but the strategy chosen makes it next to useless.

In order to ensure that punishment index data truly reflects the performance of a behaviour module, it is essential that the module be required to explore all relevant aspects of its environment. This can only be done by monitoring the robot during its learning phase and "encouraging" it to abandon less than optimal strategies. For example, an obstacle avoidance module controlling a robot in a perfectly square playpen may only ever execute right hand turns (as it follows the outer wall through all four corners). The playpen should either be reconfigured to present a more diverse mixture of avoidance options or the experimenter should periodically reposition the robot so that it has equal opportunities for executing avoidance behaviours other than right hand turns.

The proposed global performance index was also deemed unsuitable for the present work. It places emphasis on the optimality of the *paths* chosen to achieve a goal by assessing the number of achievements versus the number of robot moves, *t*, since the start of the observation. In other words, if two controllers that have learned how to accomplish the goal are compared using the proposed global criteria, the one which completes the task using the fewest number of moves will receive a higher score.

From a system resource conservation standpoint, it is desirable to have controllers which can find the optimal path to a goal state. However, in this work, we are interested only in assessing whether the controller behaviour is "good enough" to do the intended job rather than "how good" it is in comparison to itself or another controller. Therefore, the assessment of the global performance of the controller in this work is a relaxed version of the proposed index. It was evaluated by the taking the ratio of the number of times the controller is successful, $n_{success}$, versus the total number of trials, *n*, for any particular fixed environment, *i*, as shown below.

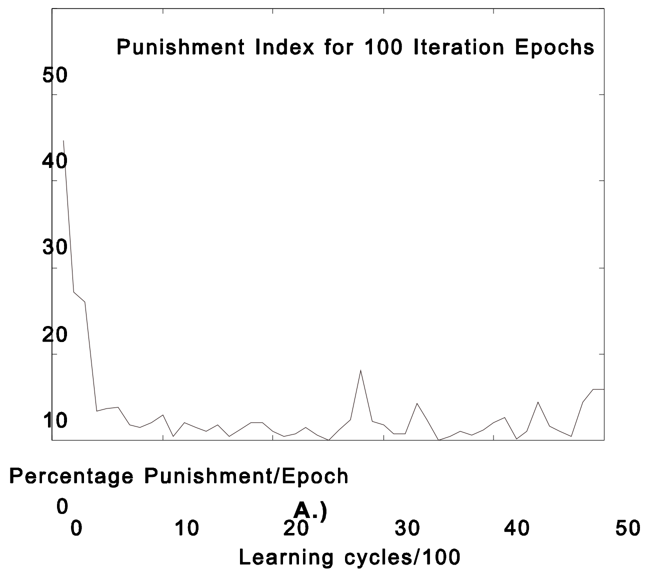
$$G(t) = \frac{n^{i}_{success}}{n^{i}}$$

Success is defined as the robot having reached the light object as specified in the *Application Description* stage. No time limit on this success was specified because it may take the controller longer to navigate through a complex environment than one that is less complex.

7.3 Obstacle Avoidance Module Performance

7.3.1 Uniform versus Gaussian Random Exploration Rule

Figure 7-1A shows a graph of the *punishment index* for the obstacle avoidance module trained with an exploration rule based on uniformly distributed random numbers. Figure 7-1B shows the same type of graph for the module trained using a Gaussian random number distribution. Both graphs show performance data averaged over five training sessions. Data points in these graphs were computed by averaging the *punishment index* data points of each individual graph. Epochs were 100 cycles long. For Figure 7-1B, the mean of the Gaussian distribution was fixed at 0.5 and its variance was fixed by *randscale* at 40.



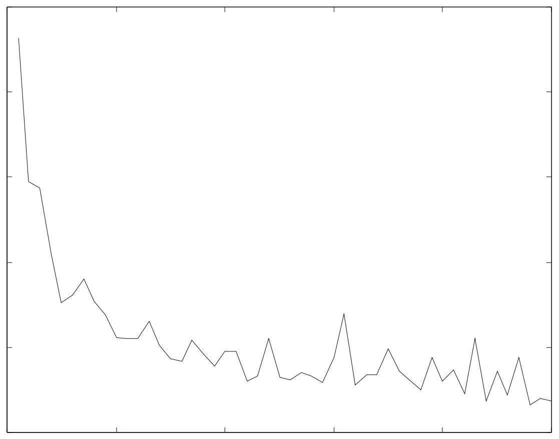


Figure 7-1
Obstacle Avoidance Punishment Index Data for Real Khepera (Averaged)
A.) Gaussian Distribution B.) Uniform Distribution

The general trend in learning with the Gaussian exploration rule shows a marked decline in the number of punishments per epoch very early on in the training. The punishment performance of the uniform distribution exploration rule shows a decreasing trend but this is not as dramatic as that of Figure 7-1A.

During the training of an obstacle avoidance module (learning with a Gausian random exploration rule), a change to the environment was made to observe the effect it would have. The new situation was introduced by moving a wall as indicated in Figure 7-2. This created a tighter dead end so that when the robot executed a turn to extract itself from the corner, more of its proximity sensors would be activated than normal which would therefore present previously unvisited states at the inputs to the network.

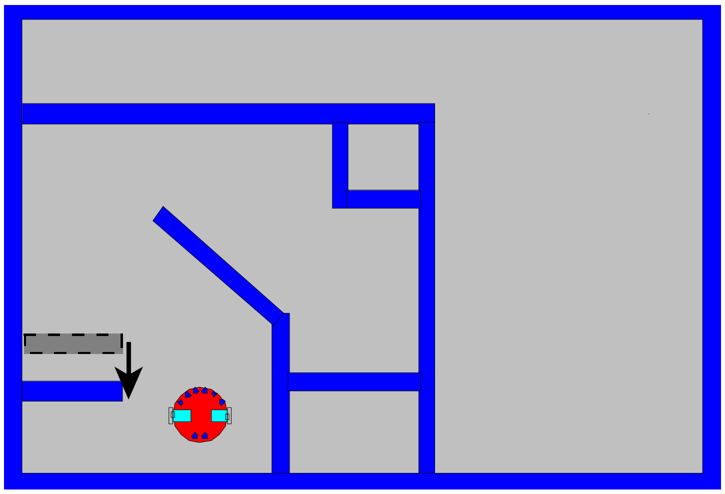


Figure 7-2
Creating a "New Situation"

Figure 7-3 shows the effect of the change in terms of punishment incurred. The module has had a chance to learn its environment over 2500 cycles. When the new situation is encountered, a brief period of "new learning" occurs in which a large amount of punishment is received.

The reason for this large amount of punishment is intuitive and relates to the network *confidence* level. The network outputs are pushed incrementally either towards the target vector (provided by the environment through CRBP) or towards 0.5. As a network gains experience, and the outputs get closer to the target vector, then we may say that the network becomes more *confident* in its action selection at each state in its environment. When new states are encountered in the environment, the network outputs produce a generalized response (since the network is a function approximator). If this response is incorrect, then the outputs of a network using the *uniform distribution of random numbers* as an exploration rule generally do not have to be moved very far before the correct action is chosen and reinforced. This is due to the fact that the movement towards the target vector is slower since a considerable amount of exploration is still being performed (as evidenced by Figure 7-1B). For an exploration rule based on Gaussian distributed random numbers, correct network outputs are reinforced at *every* step outside of the zone of exploration (defined by the variance of the Gausian distribution), quickly driving them to near saturation. The actions chosen in previously unvisited states or states that have changed due to environment dynamics will generally have to be punished a large number of times to drive the network outputs back into the exploration zone, where more punishment will be incurred during the exploration process which is required to find a correct solution. This is reflected as the spike in Figure 7.3.

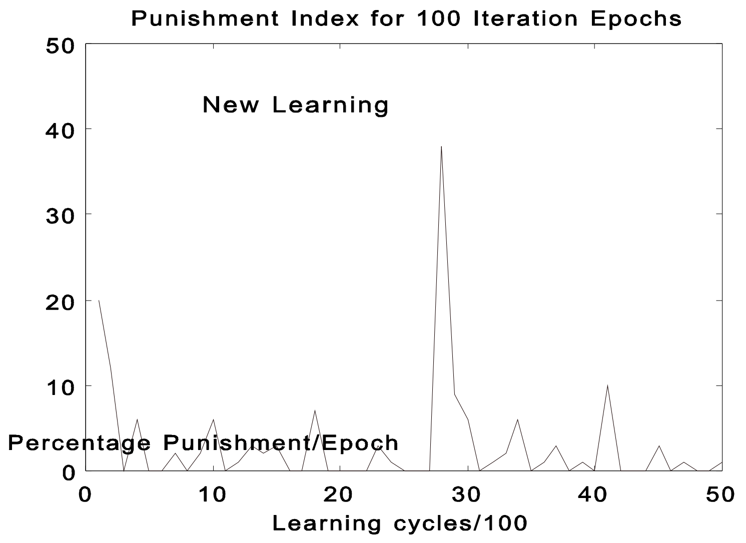


Figure 7-3
Individual Training Run of Gaussian Random Exploration Rule
During Obstacle Avoidance Learning

7.3.2 Threshold Exploration

A threshold function is one in which any network output greater than or equal to 0.5 is forced to 1 and any output below 0.5 is forced to zero. It is the limiting case of the Gaussian exploration rule. That is, it is the case where the Gaussian distribution has an infinitely small variance from the mean of 0.5.

In the previous section, the Gausian distribution of random numbers for the Gaussian exploration rule was confined to the relatively small region between 0.4 and 0.6. It was desired to see if the limiting case (ie the threshold function) could perform any better as an exploration rule than the Gaussian distribution.

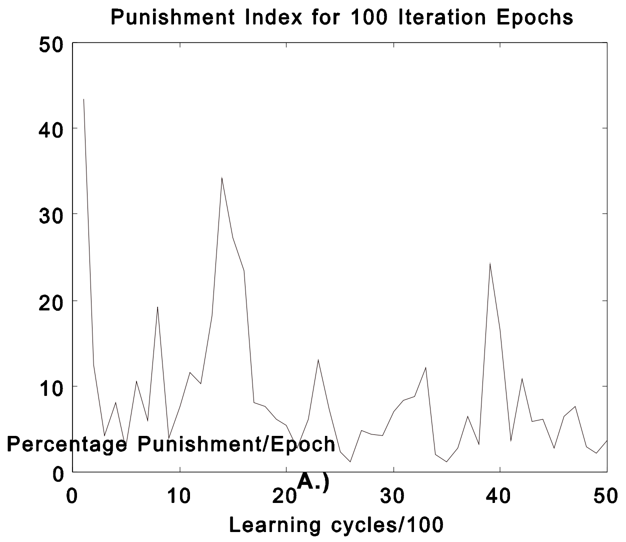
The only exploration capability provided by the threshold function is supplied by minor differences in the adjustment of the weights and biases during the back-propagation of error and this turned out to be too little to enable efficient learning. Although some learning took place, the modules tested spent a large amount of their training time oscillating between undesirable states. For example, if the network outputs were [0.51 0.55], they would be thresholded to [1 1]. If the action selected by this vector was incorrect, the network would be trained on [0 0] (due to CRBP) so that on the next cycle the network outputs might be [.47 .49]. If the action selected by this vector was also incorrect, then the network outputs would be pushed back towards [1 1]. This oscillation would continue until one of the network outputs was not pushed completely over the threshold of 0.5 - for example [.51 .48]. This would be thresholded to [1 0] which corresponds to action not tried since the network was oscillating between [1 1] and [0 0]. If the action [1 0] was punished, then the network would get pushed towards [0 1] (again due to CRBP choosing the opposite action). This, of course, would likely be rewarded since there are only four possible actions in each state in this work and three had already been tried.

7.3.3 Simulator Trained and Transplanted Modules

The behaviour module for obstacle avoidance was modified as outlined in the *Khepera Toolbox User's Manual* for use in simulation. This involved attaching the extension `sim_` to each command in the module that issued an instruction to the real Khepera. It also involved making a call to **kheprom.m** once every iteration to update the simulated Khepera. The code for the modified behaviour module is provided in Attachment 1.

Once again, both the uniform and Gaussian random exploration rules were used during training (5000 cycles). Figure 7-4 shows graphs of the *punishment index* for the obstacle avoidance module trained with both of the respective exploration rules in simulation. Both graphs display punishment performance data averaged over five training sessions. Data points in these graphs were computed by averaging the *punishment index* data points of each individual graph. Epochs were 100 cycles long. For Figure 7-4B, the mean of the Gaussian distribution was fixed at 0.5 and its variance was fixed by *randscale* at 40.

The general trend in both Figure 7-4A and 7-4B is a decline in the number of punishments per epoch as the number of cycles progress. However, the simulated modules trained with the Gaussian exploration rule show a much faster decline to a minimum than the modules trained using the uniform exploration rule, as was expected. Note the large peaks in Figure 7-4A which correspond to areas where individual modules making up the average have engaged in learning a new aspect of the simulated environment.



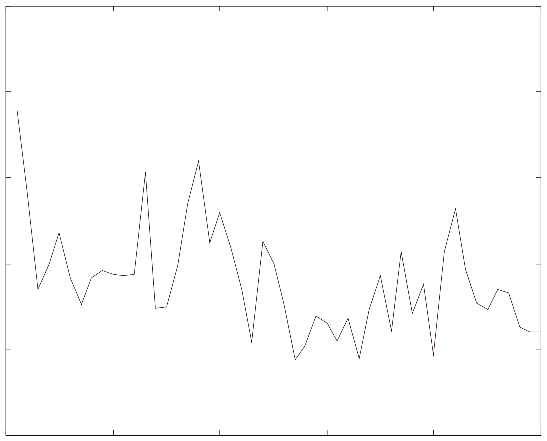


Figure 7-4
Obstacle Avoidance Punishment Index Data for Simulated Khepera (Averaged)
A.) Gaussian Random Distribution B.) Uniform Random Distribution

The simulator trained weights and biases of *each* avoidance module were transplanted individually into the real Khepera and 2000 more training iterations were performed. Since they have already been somewhat trained, each transplanted module can be said to have some "initial knowledge" about its environment.

The punishment performance of the Gaussian and uniform random exploration rule (averaged over five training runs each) for the transplanted modules is displayed in Figure 7-5. Both graphs lack the initial drop from very high punishments per epoch due to the "initial knowledge" successfully implanted by the simulator. It is clear from the downward trend in the curves of both Figure 7-5A and 7-5B that the incremental learning required to adapt to the real environment is taking place.

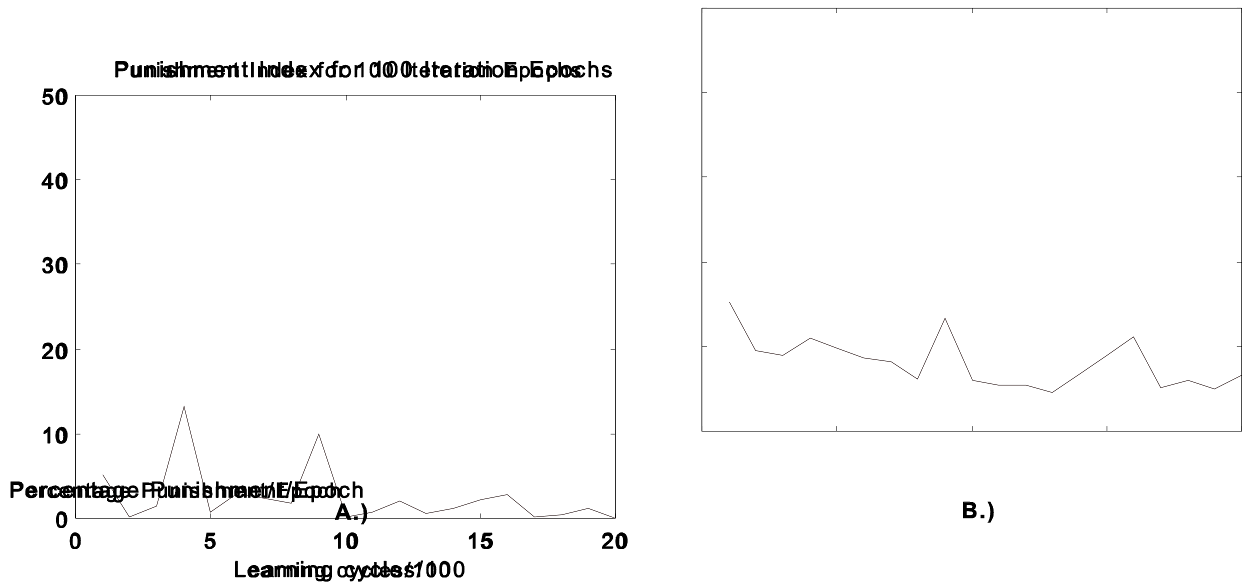


Figure 7-5
Obstacle Avoidance Punishment Index Data (Averaged) for
Real Khepera With Initial Knowledge
A.) Gaussian Random Distribution B.) Uniform Random Distribution

All of the modules with initial knowledge which were placed in the real environment performed very well from the outset and some performed better than others (from a punishment index and observed performance perspective). This diversity can be attributed to the training in the simulation environment. All of the sensors responses in the simulator are computed equally. However, it was determined that the left 45 degree sensor on the real Khepera is biased incorrectly resulting in lower readings for proximity sensing and higher readings for ambient light. Also, the simulation environment is rigid. There is no guarantee that the simulated Khepera will explore the entire space provided and there is no means of pausing the operation to move the robot to another location within the environment. In fact, two simulation runs had to be discarded because the Khepera got “stuck” immediately in a small corner area where it performed only right hand turns during the whole training session. Finally, the diversity of training examples may have been affected by the low percentage of noise selected for the trials. It was demonstrated in [Meeden93] that higher levels of noise seemed to played a key role in the training of a simulated connectionist architecture. As a result of these factors, when the initially trained module is transplanted, there are situations which it does not recognize and a certain amount of punishment will be incurred in exploring those situations. Of course, as mentioned previously, when exploration is required, the punishment index is likely to be higher for the Gaussian random exploration rule than for the uniform random exploration rule since it must “unlearn” until its network outputs are back in the exploration zone. This is evident as the two peaks in Figure7-5A which were not completely averaged out.

7.4 Light Seeking Module Performance

7.4.1 Real World Trained Module

The average punishment index for five training runs of the light seeking module (using a real Khepera) with a Gaussian distributed random exploration rule is shown in Figure 7-6. Only the Gaussian exploration rule was used during the training of the light seeking modules.

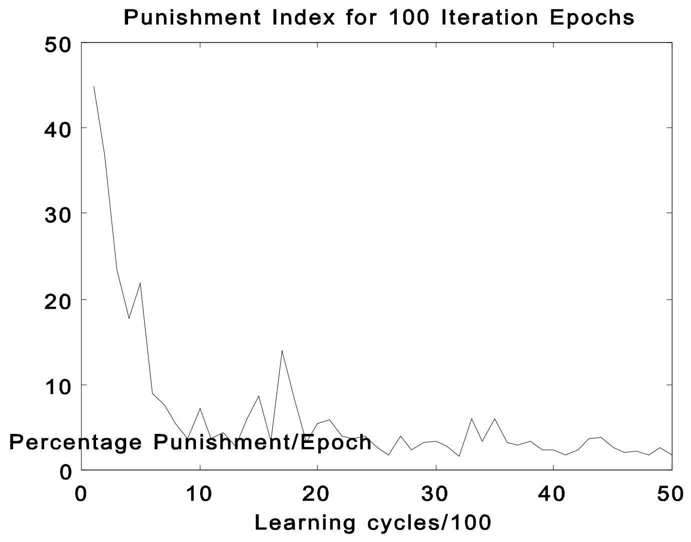


Figure 7-6
Light Seeking Punishment Index Data (Averaged)
for Real Khepera and Gaussian Random Exploration Rule

Although the punishment index would indicate a quick proficiency at seeking and approaching the light, the metric does not show whether the proficiency being developed is actually correct. In this case, it was almost - but not quite- correct. As mentioned earlier, the left 45 degree sensor on the Khepera was biased incorrectly. The result was that the left 10 degree and left 90 degree sensors often read values that were smaller (the light reading decreases with increasing intensity) than the left 45 degree sensor even when the light source was directly 45 degrees off to the left. This was not interpreted well by the input representation function. Recall that this function scans the inputs for the minimum value and sets that sensor position to 10 while setting all others to zero. The network is then taught to locate the 10 on one of the centre sensors and move forward. With the sensor biased incorrectly, the robot would follow a trajectory as shown in Figure 7-7. That is, it would not directly approach a light if it appeared off to the left.

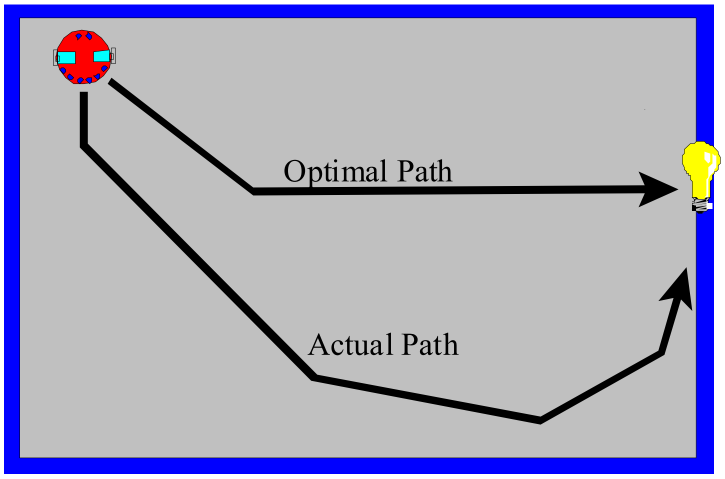


Figure 7-7
Approach Path Due to Bad Sensor

The behaviour of Figure 7-7 was due to the fact that both the network and the reinforcement program were using the same set of sensors to observe the world around the robot. Observing the behaviour of the robot and generating intelligent reinforcement data is a nontrivial function which is made much more difficult when the sensors are not working as required. This is an obvious disadvantage and [Dorigo93] suggests that additional sensors be used by the reinforcement program for assessing the behaviour. However, the Khepera base unit is not equipped with any other means of sensing the outside world. It has also been suggested that the reinforcement program could use the two front sensors and generate reinforcement signals from the light gradient, but this does not eliminate the problem of a poorly biased sensor which could just as easily have been one of the front sensors.

Manually adjusting the sensor bias was not an option (the Khepera is tightly integrated and circuit diagrams are not supplied). Providing a mathematical bias within the input representation function that would work properly at all distances from the lamp was also challenging. Therefore, the behaviour – although clearly not optimal – was deemed acceptable since it did eventually get the robot to the light source.

7.4.2 Simulator Trained and Transplanted Module

The behaviour module for light seeking was modified as outlined in the *Khepera Toolbox User's Manual* for use in simulation. The code for the modified behaviour module is given in Attachment 1.

Figure 7-8 shows graphs of the *punishment index* for the simulator trained lite seeking module trained using the Gaussian random exploration rule. The graph shows data averaged over five training sessions. Data points in these graphs were computed by averaging the *punishment index* data points of each individual graph. Epochs were 100 cycles long. The mean of the Gaussian distribution was fixed at 0.5 and its variance was fixed by *randscale* at 40.

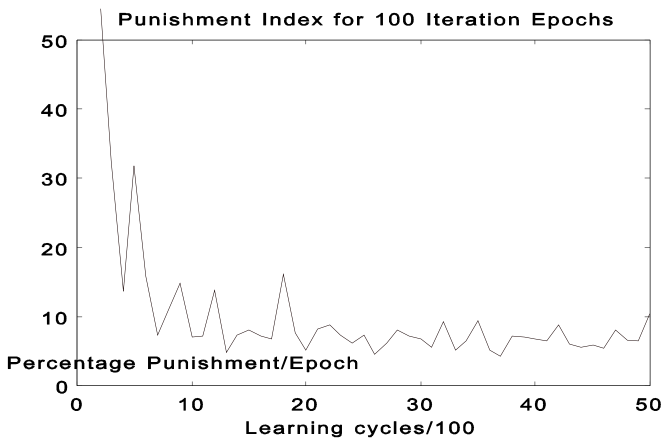


Figure 7-8
Light Seeking Punishment Index Data (Averaged)
for Simulated Khepera and Gaussian Random Exploration Rule

The simulator trained weights and biases of each light seeking module were transplanted individually into the real Khepera and 2000 more training iterations were performed. The performance (averaged over five training runs each) for the transplanted modules is shown in Figure 7-9.

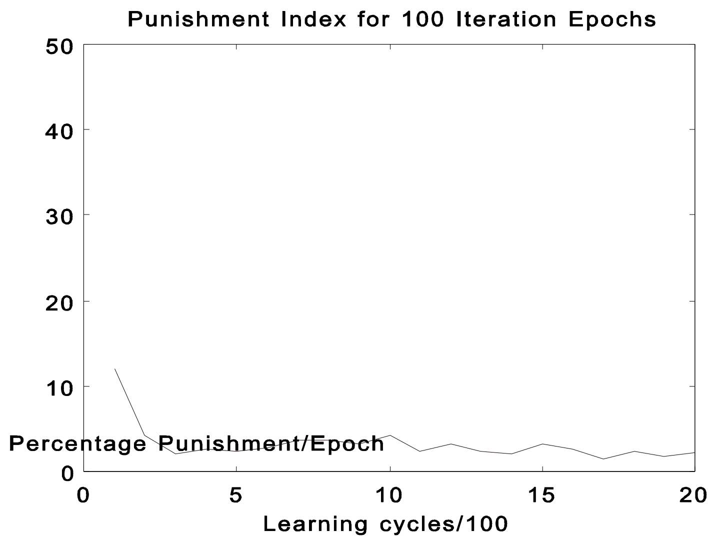


Figure 7-9
Light Seeking Punishment Index Data (Averaged) for
Real Khepera With Initial Knowledge

7.5 Global Behaviour Analysis

7.5.1 Initial Test Environment

Figure 7-10 shows the environment in which the controller was initially evaluated. It consisted of the entire arena, with walls as shown. The apparatus shown in front of the light source is an optical infra-red (IR) filter used to prevent the robot from becoming confused by IR emitted from the bulb. Since the lamp is driven by a DC source and the input representation functions perform low level IR filtering, the optical filter was not strictly needed unless the robot was required to approach to within several centimetres of the bulb. In this case, the robot did not approach to within this distance because of the height of the bulb above the playpen.

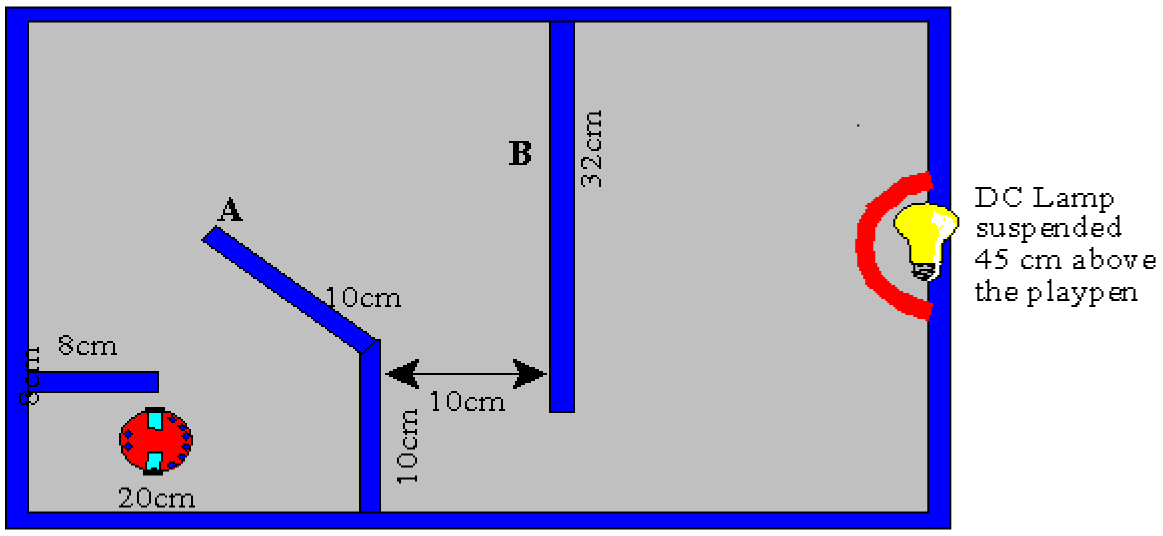


Figure 7-10
Initial Evaluation Environment

7.5.2 Global Performance in the Initial Environment

The weights and biases of one trained obstacle avoidance module and one trained light seeking module were selected at random from the modules that were trained in the real environment[1]. These were incorporated into the controller architecture (**phoebus.m**) and 10 trials were conducted in the initial environment. Another 10 trials were conducted with weights and biases from obstacle avoidance and light seeking modules trained exclusively in the simulator.

The global performance of the controller architecture for the both the real and simulator trained behaviour modules is shown in Table 7-1 under Environment Index 1. The modules trained in the real environment had a 100% success rate in reaching the light source but no trial resulted in an “optimal” solution (the most direct path to the light source). In many of the cases, the robot turned left at the wall marked **B** and followed it back to the corner before turning around. In one case, the robot turned left at **B** and followed the outer wall all the way back to the start position. In other cases, the robot doubled back into the starting corner after reaching the end of the wall marked **A**.

The modules trained exclusively in the simulator and integrated into **phoebus.m** also had a 100% success rate in the environment of Figure 7-10 even though the modules had never been

[1]It did not matter in this case whether the modules were trained with the uniform or Gaussian random exploration rule as long as they exhibited the desired behaviour by the end of the initial training session. The reason is that the behaviour module network outputs are thresholded at 0.5 by **dataround.m** in **phoebus.m**. Therefore, it does not matter how “confident” the module is in choosing an action, as long as the action chosen is correct.

exposed to real sensor data. This was particularly encouraging since it actively demonstrated that the simulation sensor responses were calibrated reasonably well to the actual environment.

The relatively quick negotiation of the environment by the robot in reaching the light source was not expected. The environment of Figure 7-10 was chosen to see *if* the robot could actually make it to the lamp because it was assumed that the angled wall (marked A in Figure 7-10) would cause the controller some difficulty. This assumption was based on the fact that the obstacle avoidance module would turn the robot away whenever a wall was present and, once the wall was no longer visible, the light module would take over and drive the robot back at the wall. It was not known if such combined behaviour would be able to get the robot beyond the angled wall or if the robot would end up in an "eternal loop" as shown in Figure 7-11.

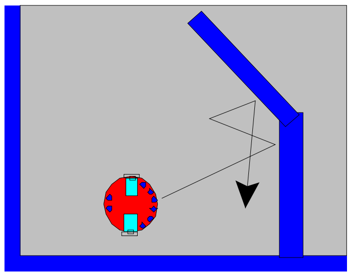


Figure 7-11
The "Eternal Loop"

The looping behaviour did not happen and the robot did not encounter any problems with the angled wall (or any of the other walls) because the obstacle avoidance module was implicitly trained to perform wall following by its reinforcement program. That is, the reinforcement program did not cause the network to react to obstacles that were detected on either left or right 90 degree sensors. This lack of reaction was exploited by the integration mechanism to cause a wall following behaviour to emerge; any time the integrator switch saw that an obstacle was detected on *any* sensor, it gave control to the obstacle avoidance mechanism. If the detection was on the left or right 90 degree sensors, then the obstacle avoidance module would continue to issue "move forward" commands. This effectively caused the robot to follow linear contours.

7.5.3 Extended Environments

Several extended test environments were produced to evaluate the *robustness* of the controller architecture. There is an obvious limit as to what can be expected from the controller given that there is no real path planning or world modelling capability. However, it was desired to see what effect various types of obstacles would have on the performance of the controller architecture in moving towards a light source.

The controller was evaluated (using two randomly selected modules from obstacle avoidance and light seeking, both real and simulated) in the five environments shown in Figure 7-12. Environment 2 was introduced to see what size of hole the robot could negotiate. Two testing sessions were performed with the hole size at 7.5 cm and 6.5 cm respectively (the robot itself is 5.5 cm in diameter). Environment 3 was introduced to see if the robot would follow a hallway of a depth larger than one cm. Two testing sessions were performed with the width of the hall at 7.5 cm and 6.5 cm respectively and a depth of 10 cm. Environment 4 was introduced to see if the robot would get stuck in a false opening. Finally environments 5 and 6 were introduced as possible failure scenarios.

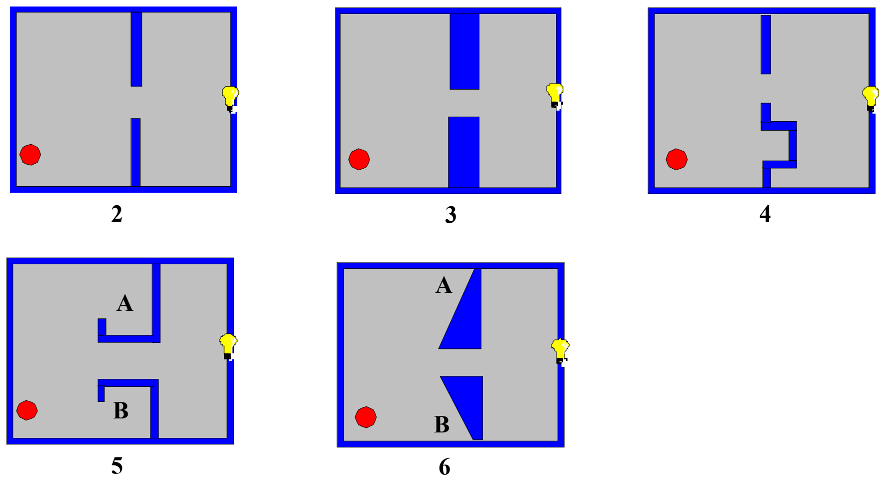


Figure 7-12
Extended Environments for Global Performance Evaluation

7.5.4 Global Performance in Extended Environments

The global performance indices (for modules trained both in the real environment and in simulation) are given in Table 7-1. All indices were computed on 10 trials. Due to the symmetry of the extended environments, even trials were initiated with the Khepera in the left bottom corner while odd trials were initiated with the Khepera in the left top corner.

The controller with the modules trained in the real environment had no trouble getting to the light in Environment 2 through an opening of 7.5 cm or the smaller opening of 6.5 cm. However, when starting in the top left corner, the robot had occasion to miss the door (ie it would make contact with the lower portion of the dividing wall first) due to the indirect light approaching behaviour caused by the bad sensor.

The controller using modules trained in the simulator had some difficulty when its approach began from the top left hand corner of Environment 2. The robot would not fully turn the corner when going into the space between the walls. It would move partially into the doorway and oscillate back and forth, hitting the far edge of the opposite wall. This failure was caused primarily by the left 45 degree sensor bias problem. By symmetry, it is reasonable to assume that the robot would have performed correctly from both approaches had there not been a poorly biased sensor on the left side.

The controller using modules trained in the real environment successfully navigated Environment 3 when the hallway width was 7.5 cm. However, when it was reduced to 6.5 cm, the robot showed rather poor global performance. In order to successfully get through constricted hallway, the robot was required to enter in an exactly straight trajectory or else it would spend its time oscillating in place. A straight line entry occurred only once. If the turn granularity was finer, this problem could be overcome.

The controller modules trained in the simulation environment successfully navigated both hallway widths without problem.

Environment 4 did not present a problem. The controllers using modules trained in both the real environment and in simulation wall-followed their way out of the false opening.

The controllers using both real and simulator trained modules did not do well in Environments 5 or 6. Two successes in Environment 5 were due to the indirect light approaching behaviour; the robot moved directly to the opening in the wall which would not have occurred if the light sensor was biased correctly. The remainder of the trials got stuck circling in the spaces marked A and B in Figure 7-12(5). There were no successes in Environment 6. All trials ended up oscillating in the corners marked A and B in Figure 7-12(6). This, however, was expected since the training environment for obstacle avoidance did not contain any corners whose angles were less than 90 degrees, so the module chosen for integration into overall controller was not expected to know how to deal with the corners marked A and B of Environment 6 in Figure 7-12.

Summary of Global Performance Results			
Environment Index	G(t) Real	G(t) Simulated	Notes
1	1.0	1.0	Initial Environment
2	1.0	0.8	Opening size 7.5cm

Summary of Global Performance Results			
2	1.0	0.5	Opening size 6.5cm
3	1.0	1.0	Opening size 7.5cm
3	0.1	0.8	Opening size 6.5cm
4	1.0	1.0	
5	0.2	0.0	
6	0.0	0.0	

Table 7-1
Global Performance Results G(t) for Controllers
Trained in Real and Simulated Environments.
G(t) is the number of successful trials divided by the
number of toal trials. 1.0 corresponds to 100% sucess.