Chapter 8

Discussion and Conclusions

8.1 Discussion

8.1.1 Effectiveness of Khepera Toolbox: Matlab-Khepera Interface

The group of functions and programs prepared as part of this project for the Matlab-Khepera interface (known as the *Khepera Toolbox*) were particularly effective for implementing control algorithms from within Matlab in real time. The development of the light seeking/obstacle avoidance algorithms served to verify this claim and to help define the functional requirements of the toolbox as it was being constructed.

The implementation of the neural networks in this work was matrix intensive. Due to the fact that the basic element in the Matlab environment is the matrix, the control algorithms were relatively easy to program within the Matlab environment (easier, for us, than C or Fortran). Additionally, Matlab matrix functions have been optimized for speed and the higher level of expression made the programs appear simpler.

The availability of the *Neural Networks Toolbox*, as an extension of Matlab, helped reduce design and implementation time and complexity. Several functions from this toolbox were used with minor modifications to allow single entry processing rather than batch entry processing. On the whole, the modifications were very easy and straight-forward. For more traditional design approaches, Matlab can be extended by the addition of the *Control Toolbox*.

The most important aspect of the Khepera Toolbox from our point of view is that the control algorithms developed in this work were implemented on the Khepera robot in *real time*. The robot was not required to move incrementally and then stop while the computer updated the network. This was achieved on a serial link running at 9600 bps. For comparison purposes, Meeden's Carbot [Meeden94] took 2.5 hours to complete 3000 iterations of learning. The Khepera took no more than 20 minutes to complete 5000 iterations of learning.

8.1.2 Effectiveness of Khepera Toolbox: Khepera Simulator

The run-time performance of the Khepera Simulator (part of the Khepera Toolbox) was somewhat disappointing. When all of the required functionality had been integrated, the simulated robot running on a Pentium 100 CPU moved no faster than a real Khepera robot. It had been hoped that the simulator would be able to provide a speed increase of several orders of magnitude since the dynamics of the real robot could be implemented in software and then executed as speeds which are not safe or feasible for the real robot. Unfortunately, the computational overhead of providing and updating a display of the simulated robot combined with the complexity of the sensor response computations lead to a significant reduction in simulator speed.

This does not, however, detract from the usefulness of the simulation environment. From a global perspective, robot control algorithms can still be evaluated safely in the simulator where bugs can be ironed out prior to implementation on the real robot. This can be done on more than one computer at a time, thereby allowing parallel implementations to be performed.

The software written to allow the simulator sensor responses to be calibrated to match a real Khepera was found to be particularly effective and useful. This was made evident by the almost perfect performance shown by robot control algorithms which were trained initially in the simulation environment and then transplanted into the real Khepera in a real environment. Of course, this performance had a lot to do with the input representation function chosen. It allowed the sensor responses to be loosely approximated as opposed to modelled perfectly. However, if the sensor data was being used in its raw form, a sufficiently accurate modelling could still be achieved by spending more time iterating the sensor calibration procedures.

8.1.3 Integration of the BAT Methodology with the Khepera Toolbox

The use of the BAT methodology in conjunction with the fixed sensorimotor apparatus of the Khepera base unit resulted in the creation of a control architecture that was generally successful in achieving its goals. However, the fixed nature of the sensors did limit controller development in several respects. These are discussed individually in the following two sub-sections.

8.1.3.1 Complexity of Reinforcement Programs

The fixed sensorimotor apparatus of the Khepera base unit made difficult the shaping of the individual behaviour modules (as prescribed in the BAT methodology).

Both the reinforcement program and the learning system (the recurrent neural network) for each behaviour module were required to share the same sensors to perform their respective tasks. As noted, the Khepera base unit was not equipped with any other external sensing devices. Identification of the reinforcement program's sensors with the learning system's sensors has been done before [Mahadevan92] but several shortcomings have been identified [Dorigo93]. The most important of these shortcomings is that the shaping policy is bound to the low-level details of the robot's physical structure and this will, in general, force the reinforcement program to be as complex as a program directly implementing the target behaviour. The need to create a complex reinforcement program limits the effectiveness of learning as an alternative to robot programming.

The listings of the reinforcement programs for light seeking and obstacle avoidance are complex, validating the shortcoming identified above. In the case of obstacle avoidance, there is little that can be done in terms of simplification. When an obstacle is detected, and an action is performed, all of the data from all of the sensors is required by the reinforcement program to determine the merit of the action in avoiding the wall. If the Khepera could be fitted with more complex sensors which could give an indication of relative proximity (in relation to the last position of the Khepera) while taking into account the angle of approach, then the reinforcement program could be simplified greatly.

It was suggested that the reinforcement program for light seeking could have been simplified by using the light gradient between the front two sensors or the front 45 degree sensors. However, in this work, the robot can only perform four discrete actions (right/left turn, forwards/backwards), and each action has only one speed. It is anticipated that the reinforcement program using a light gradient approach would have difficulty determining when it was appropriate to reward the action of moving forward. The left and right light levels would have to be equal (or within some small tolerance of each other) in order to move forward. Since both sensors would rarely be equal, the tolerance level would require empirical determination and may vary with differing light sources. Additionally, as pointed out earlier, the required reinforcement program would have trouble in observing and reinforcing the *correct* behaviour given that the front 45 degree sensor was biased incorrectly. A neutral, more complex sensor could simplify the reinforcement program for light seeking considerably.

Having just identified that the reinforcement programs are complex, it was determined that the reinforcement program for the obstacle avoidance was *not* complex enough to allow the Khepera to extract itself from corners whose angles subtended less than 90 degrees.

The manner in which rewards were determined for the obstacle avoidance module lead the avoidance behaviour to become caught in a deadlock situation when the robot approached a corner in which the angle was less than 90 degrees.

Recall that recurrent neural networks were introduced in this thesis as the principal learning architecture of each behaviour module since their temporal processing capability allows them to learn sequences of actions. This capability was to be exploited to allow the Khepera to avoid being stuck in deadlock situations by learning the sequence of actions that would allow it to extract itself. However, by implementing the CRBP algorithm, each action in the sequence must be rewarded in order to learn the sequence since, if an action is incorrect, the opposite action is used as the training vector.

As an example, consider corners in the environment which were 90 degrees or more. The reinforcement program was successfully able to use the available proximity sensors to determine if each action lead to a better state of affairs. This allowed an *extraction sequence* of actions to be learned. As the robot rotated its way out of the corner, the reinforcement program was able to determine that each individual rotation action lead to a better state of affairs than the last state, and hence each action in the sequence was rewarded. Note that the training environment for the obstacle avoidance behaviour did not contain any corners in which the angle was less than 90 degrees.

For corners that were less than 90 degrees, the robot would get stuck. The reason for the failure was determined by observing the behaviour of the reinforcement program during the extraction attempt. When the robot executes a turn to the right to extract itself from the corner, the reinforcement program determined if the turn resulted in the sum of the (preprocessed) sensor values on the right side ended up being less than the sum of the values on the left (as indicated in Listing 5-1). That is, it attempts to determine if the current state is better than the last state. Due to the thresholding of the **preproo.m** function, the right sum *is* less than the left for a corner that is equal to or greater than 90 degrees and the network is rewarded. However, for corners that are less than 90 degrees, this is not the case (due to the shape of the walls and distribution of the sensors) and a punishment is received causing the Khepera to choose the *opposite* action. This results in a turn back towards the corner. The outcome is that the Khepera ends up demonstrating the “stupid reactive behaviour” that should be avoided.

The temporal processing capabilities of the recurrent neural network could be put to greater use in extracting the Khepera from all corners (regardless of angle) if additional complexity was incorporated in the reinforcement program for obstacle avoidance. However, creating a reinforcement program that is more complex than directly programming the behaviour was not the thrust of this work. The provision of additional or different sensors for determining the Khepera's position and angle in relation to the walls in the environment could greatly reduce the coding requirements in the reinforcement program.

In summary, a key precept of the BAT methodology is that sensorimotor interface of the robot cannot be fully identified until the Specification stage is complete and that sensors can be designed and implemented to suit the requirements of the behaviours to be learned. This work validates that precept. The Khepera base unit was equipped with a fixed set of sensors. As demonstrated in the previous paragraphs, this had repercussions for the complexity of software design and resulted in making the learning of a behaviour generally as difficult as directly programming the behaviour. Custom designed sensors must be added to the Khepera base unit is required in order to simplify the learning of behaviours.

8.1.3.2 **Limitation on Intelligent Control**

The Khepera base unit's lack of long range sensors limited the application of the BAT methodology in creating "intelligent", hierarchical controllers in this work.

In the chapter on Specification, a simple switch architecture was used to arbitrate between the two behaviour modules. This was not the original intent. Initially, an hierarchical controller architecture was chosen as shown in Figure 8-1. In this architecture, the third behaviour module is a co-ordination module that was to be taught how to arbitrate between the two lower behaviours.

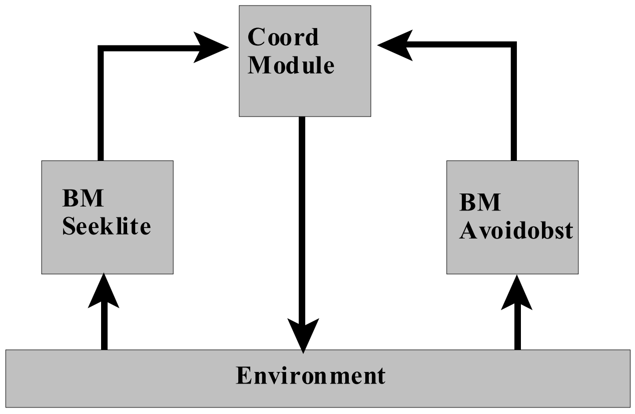

Figure 8-1
Original Controller Architecture

It was not possible to construct a controller for the Khepera base unit (as shown in Figure 8-1) which was any more “intelligent” than using a simple switch architecture in place of the coordination behaviour module. The reason was that the base unit's proximity sensors were too near-sighted to provide an effective evaluation at every iteration of the control algorithm. That is, a reinforcement program could not be developed for the coordination module using the available sensors which would able to determine the long term effect of the current action and would thus not be capable of providing the appropriate reward value.

For example, if the robot was to encounter the situation shown in Figure 8-2A, it would be intuitive for an observer looking from above to identify the correct direction in which to turn. Indeed, if the reinforcement program of the coordination module had access to the same view of the environment as the external observer, it could easily be written in such a way that it would issue a punishment if the robot turned right and a reward if it turned left.

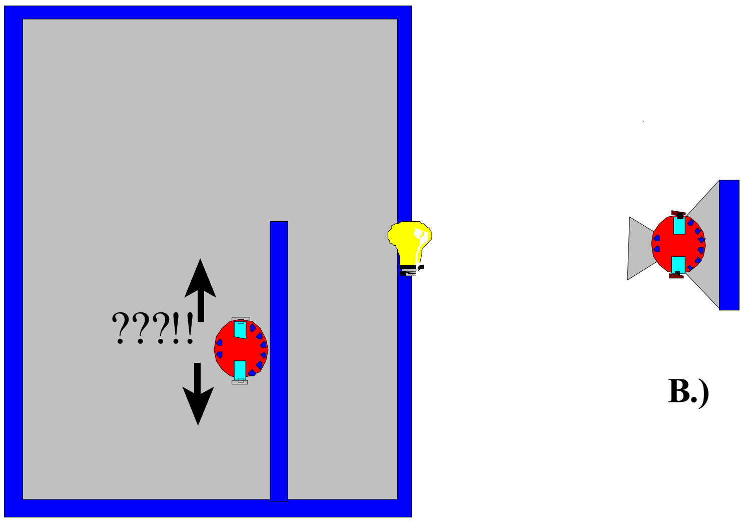

Figure 8-2
The Near-Sighted Problem

However, the coordination module reinforcement program (using the sensors available to the Khepera base unit) would only be able to “see” the world as shown in Figure 8-2B. Effectively, a reinforcement program cannot be written using this information which would be able to tell which way to turn with respect to the goal and would therefore be paralysed when trying to determine a reinforcement value to assign. Recall that in Chapter 2, it was established that the most critical component in the learning system is the “trainer” (or the reinforcement program). Teaching the coordination module the “right thing” presupposes the ability to observe the robot's behaviour in relation to the goal and this was not possible with the sensors available.

The Khepera is equipped with an odometric capability when it is used in the position mode. This capability was considered for use by the reinforcement program of the coordination module to provide the necessary long range sensing. However, it was required that the reinforcement program be equipped with a map of the environment and that the Khepera be started from a known position and angle at each trial. Additionally, any slippage due to motion or collision would cause the Khepera's position on the internal map held by the reinforcement program to diverge from the actual position. Also, it was not clear how two-dimensional position references could be massaged into intelligent input for the neural network of the co-ordination module. Given these uncertainties, this option was not pursued.

8.1.4 Exploration vs Exploitation in CRBP

The exploration function in the original implementation of CRBP did not take full advantage of network confidence in determining when to explore the environment and when to exploit the knowledge already possessed by the network. In that implementation, a uniform distribution of random numbers was used to enable random bit changes in the network action output vectors thereby causing random exploratory actions to occur.

The uniform distribution essentially related exploration and exploitation in an inverse manner. That is, as the network outputs moved further away from the mean of 0.5 (indicating increasing confidence in the choice of action), the amount of random exploration decreased in a linear fashion. With a uniform distribution of random numbers, the amount of random exploration occurring in regions of high network confidence is significant. For examples, a network output of 0.8 has a twenty percent chance of being randomly mutated to a zero and similarly an output of 0.2 can be mutated to a one.

It was postulated that a better approach would be to more tightly bound the exploration capability of the network. Specifically, if an output is near zero or one, then it is most likely that very low values should become zero and high values should become one rather than being switched to the opposite extremes. However, when an output is close to 0.5, there is little to suggest a preference for selecting a one or a zero. In this case, a random choice is as good as any and should allow enough "mutations" to occur such that the network eventually finds a suitable value.

We could have used a rectangular distribution of random numbers which would permit random mutation between, say, 0.4 and 0.6, but we felt that a more gradual slope would be better. We used a Gaussian distribution for the random numbers because it could be implemented easily within Matlab. The Gaussian distribution, whose variance contained the random numbers between 0.4 and 0.6, gave good results and allowed faster learning thereby confirming our premise. However, further study could be devoted to determining an optimum distribution when using random exploration as the search technique.

8.2 Conclusions

In this thesis, an interface was successfully developed between the Matlab programming environment and the Khepera mobile robot. This interface includes: a.) a set of core functions which allow Khepera commands to be issued from the Matlab prompt, b.) a simulation environment for the Khepera, and c.) a set of graphical user interfaces which allow various useful functions to be performed on both a real and simulated Khepera robot.

The Behaviour Analysis and Training Methodology was used as a basis in developing a control algorithm for a mobile robot which successfully approached a light source while avoiding obstacles. The *Khepera Toolbox* described above was used to implement this control algorithm on both a real and simulated Khepera.

The final goal of this work was to evaluate the utility of the using the BAT methodology in conjunction with the *Khepera Toolbox* (and Khepera robot) in the design and implementation of the light seeking and approaching control algorithm.

The BAT methodology was a very practical means of providing structure to the control system design. While most stages in the methodology were well defined, there was some difficulty during the *Behaviour Assessment* stage in determining an appropriate and comprehensive quantitative measure of performance. Additionally, two minor sections were added to the major stages of the methodology in order to give consideration to relevant aspects of the controller design. In particular, a section was added to the *Application Description* stage in order to describe software tools and hardware setup. A section was also added to the *Specification* stage in order to give a detailed treatment of the software controller structure and design.

The toolbox was found to be a very effective agent for implementing robotic control algorithms in real time from within the Matlab environment. The *Khepera Simulator*, as part of the toolbox, was found to be particularly effective in providing the behaviour modules with a high level of “initial knowledge.” This was clearly observed by the low punishment index values incurred when the simulator trained modules were transplanted into the real Khepera.

The sensor apparatus of the Khepera base unit proved to limit the application of the BAT methodology in two important ways. First, the fixed nature of the sensors on board the Khepera base unit made the shaping of the individual behaviour modules difficult and expensive in terms of code length. It also limited our ability to exploit the memory structure of recurrent neural networks to avoid "stupid reactive behaviour" when the robot was faced with tight corners.

Second, the short range of the proximity sensors on the Khepera base unit limited the application of the BAT methodology in creating more "intelligent" controllers. The use of only the light and proximity sensors of the Khepera could not provide the reinforcement program with the information required to shape the higher level controllers intelligently.

These limitations imply that the BAT methodology and Khepera Toolbox can be more tightly integrated as a design tool for autonomous robot development if both the Toolbox and Khepera robot can be expanded to include custom designed and longer range sensors.

As a final point, it was determined that the speed of learning with the CRBP algorithm could be increased by choosing a Gaussian distribution of random numbers when stochastically determining the action vector output of the network. Other distributions may prove to be superior.

8.3 Future Work

The next logical step for this research is to implement a reliable longer range sensor capability for the Khepera base unit. This could be provided by the acquisition of the *Khepera General I/O Turret* which is an optional additional turret for the Khepera base unit. It provides the user with 8 digital inputs, 3 analog inputs, and a number of digital and analog outputs. The inputs can be used to receive information from a longer range sensor of custom design.

Also, there is a host of items requiring further research and work involving the *Khepera Toolbox*. These are enumerated below:

a. The Toolbox application and simulator require porting to the Matlab 5 and Windows 95 environment. Functionality upgrades and beta testing results should be incorporated and a second version released.

b. Speed optimization of the simulator proximity and light sensor calculations **(upsv.m** and **ulsv.m**) could be achieved by conversion to MEX functions. Currently, the simulation environment is only about as fast as the operation of a real Khepera. Such optimization would allow the simulator to operate many times faster than a real Khepera.

c. The MEX core communications functions, which allow serial commands to be issued from within Matlab, currently operate at 9.6 kbps. This is the fastest speed allowed by the current PC implementation using BIOS INT14 calls. The Khepera serial channel can accommodate speeds up to 19.6 kbps so an increase to the speed of communications should be investigated and implemented.

d. MEX functions for use with Khepera optional turrets are required for use from within the Matlab Environment.

e. An extension of the Matlab/Khepera toolbox is required so that the Matlab C compiler and a third party 68020 cross compiler can be used to allow verified algorithms to be downloaded into the robot's resident RAM. This will allow the Khepera to operate completely independently (ie. without any serial cable or radio link).

f. Finally, other types of learning systems (Adaptive Heuristic Critic, Q-Learning, W-Learning) should be tried using the Khepera Toolbox to evaluate it's utility with them. Although the BAT methodology does not currently allow for learning paradigms other than immediate reinforcement learning, it would be interesting to determine if other learning paradigms could be accommodated.