Chapter 1

Introduction

1.1 General

An *autonomous* robot is one that decides of it's own accord how to relate it's sensor data to motor commands in order to successfully attain a predetermined set of goals. Such a robot is said to be *effective* if it is actually able to achieve its goals and *adaptive* if it is able to improve it's performance at achieving these goals over time. An autonomous robot is *robust* if it is able to demonstrate graceful degradation [Maes94].

Autonomous robots that are effective, adaptive, and robust are certainly worth investigating. Such robots show promise for taking on the burden of humanity's mundane tasks (such as vacuuming around the house [Ulrich97]) and could prove to be indispensable when it comes to cleaning up the world's nuclear and chemical hotspots (see "Send in the Robots", *New Scientist*, September 1995). However, as described in [Levy92], their most noble use might be in the realm of efficient and cost effective exploration and/or exploitation of our cosmic neighbours. For this application, autonymity would address the usual problem of the time delay imposed on a remote control circuit. And, as the combined costs incurred in sending a robot anywhere extra-terrestrial are extremely high, robust and adaptive behaviour is *essential*. It is difficult to shrug off failure when it is measured in *billions* of dollars.

One of the key problems to be solved in autonomous robotics research is to come up with

a control architecture for robots that will result in autonomous, yet effective, robust, and adaptive behaviour [Maes94]. Unfortunately, there are few formal methodologies (i.e. “Robot Shaping” [Dorigo94], [Perkins96]) for solving this problem and even fewer general purpose platforms upon which to test potential solutions. This work addresses the methodological and platform deficiencies in the course of developing an autonomous robot which performs a light seeking task.

1.2 Integrated Tools for Experimentation

There is a growing consensus in the field that, to be considered useful for real robotic applications, a learning technique must be tested on a real robot [Dorigo96][Mondada94]. However, working with real robots can be costly in terms of time and necessary resources and may detract unnecessarily from the development of the learning technique or algorithm. For example, before beginning their research, many authors had to develop their own robot platforms (eg. “Autonomouse” [Dorigo95], “Obelix”[Mahadevan92], “Carbot”[Meeden93], “Marvin” [Fagg96] to name a few) along with both the software and hardware infrastructure required to run and maintain them. Some systems even require their own custom programming languages (eg. [Mataric94]). Software tools required for the production and analysis of results have to be developed or customized. In many cases, research is performed using available commercial robot platforms. Unfortunately, researchers may be unfamiliar with the programming interface or the robot may not meet desirable physical requirements. The resulting proliferation of differing robotics platforms and programming languages makes code portability, algorithm diffusion, and results comparison difficult.

Chapter 1 Introduction

Two tools have come to fruition in recent years and their integration may provide a useful development environment for investigations in autonomous robot control. The first is the Khepera mobile robot. It provides a miniature, expandable, fully integrated robot platform on which experiments may be carried out safely and in a small area. A serial link is provided to run control algorithms on a workstation or PC during initial testing and debugging after which the algorithm can be downloaded to the robot and run on its own processor.

The second tool is Matlab - a powerful high level programming environment for linear algebra, matrix operations, and high quality graphical presentation. Matlab provides the researcher with a programming environment that is easy to learn and manipulate and the programming principles are much the same as Fortran or C. More importantly, the core functionality of Matlab can be extended by application specific toolboxes (eg Control, Neural Networks) which provide factory-made plug-and-play functionality for the creation of autonomous robotic control algorithms.

An integrated environment made up of the Khepera and Matlab offers a reduction in the time and resource investment required to investigate control algorithms for autonomous robots in a number of ways. Both are Commercial Off-The-Shelf (COTS) tools which has positive implications for software and hardware support as well as algorithm diffusion. Since the control algorithm may reside on a PC or workstation, all of the resources of the Matlab programming environment can be used to help understand and debug the control process. Finally, significant proficiency at standard programming languages should not be a prerequisite to investigate autonomous control algorithms.

1.3 Structured Methodologies for Control System Design

When a control system is required to interact with an environment in order to achieve a particular task or goal, it (or the designer) is faced with the problem of determining what action is to be taken at each state in the environment in order to achieve the goal. One means of solving this problem is provided by traditional engineering methodologies. That is, the designer will analyse a model of the task and come up with the necessary sequences of actions that the system should perform. Of course, allowances must be given for noise and for discrepancies between the modelled world and the actual environment. Allowances must also be made to provide for sensor or actuator calibration and degradation over time or for permanent changes in the environment if the control system is to be labelled robust and adaptive. This traditional approach can work well where the interaction between the robot and the external environment is well known and can be described fairly simply.

However, the interaction between a sophisticated robot and an unknown, uncertain or varying environment can be complex and it has been suggested that programming the control algorithms of such robots by traditional engineering methodologies is difficult ([Dorigo96], [Perkins96], [Maes94]). One way to shift the burden from the human engineer to the robot itself is for the robot to *learn* the required task on it's own. Machine learning techniques have emerged as an attempt to provide this capability, but it is not clear how they should be integrated with more traditional design methodologies. For instance, there are many examples in the literature of control systems that use learning and evolutionary techniques to successfully solve

complex real world problems ([Digney93][Fagg93] to name a few). Such solutions tend to be problem oriented and reflect no common principled approach to the design of their control systems.

Recently, Colombetti, Dorigo, and Borghi [Columbetti96] have proposed a structured methodology for the design of autonomous robot control systems which incorporates a machine learning component. This methodology, termed the *Behaviour Analysis and Training* (BAT) methodology, is a first attempt and has not been completely defined.

The BAT methodology attempts to formalize and provide structure to an older string of thought originated by Brooks [Brooks85] which has been termed *behaviour-based control* [Maes93]. Behaviour-based control is rooted in the intuitive premise that decomposing a complex task into a number of simpler sub-tasks and implementing each as a separate module is likely to make the engineering of a robot easier and more structured. The positive aspects of task decomposition have been observed in such studies as [Jacobs93], [Boyan91], [Anderson94], and [Houk92].

The original control system described in [Brooks85] incorporated modules that were directly programmed. As mentioned earlier, a relatively new vein of thought has focused on replacing the directly programmed module with one which employs machine learning techniques to realize the required functionality. Encouraging results have come to pass from this approach (eg. [Digney93],[Perkins96],[Mahadevan91]). The application of machine learning techniques to a task which is decomposed into simpler sub-tasks is a central theme of the Behaviour Analysis and Training methodology.

The authors of the BAT methodology provide four examples of the design and implementation of robotics control applications, but no further examples of it's use have been found in the literature. The Behaviour Analysis and Training methodology has been selected as the design process for this research since it provides a structured approach to the design of a robot employing a learning component.

1.4 Reactive Versus Dynamic Behaviour

The design of a controller following the premise of behaviour-based control would proceed as follows: individually build and train several modules, each corresponding to a different competency necessary for the overall task (a module for goal seeking, a module for obstacle avoidance, etc). All of the modules would operate in parallel and an arbitration scheme (eg, suppression and inhibition wires or messages among the modules, or a controller module that is trained to perform arbitration) would implement the desired module priority.

A possible source of trouble lies in the design of the individual competence modules. Learning systems that immediately map their sensor inputs onto motor outputs are *reactive* [Arkin, 1995] and, as such, have a limited repertoire of behaviours. That is, the robot action is a direct result of a specific sensory input pattern and each such input will always lead to the same output action being chosen (in the same manner that the output of a combinational logic circuit maps to its inputs). Consider the example posed by [Sharkey96]: when a robot approaches a symmetrical corner, the control structure will not be able to tell in which way the robot should be steered to free itself. When turning left, the leftmost sensor reveals little space between it and the wall, resulting in a turn to the right. Of course, the same is discovered on the right and the

sequence repeats until the robot is firmly jammed in the corner. This is known as a *deadlock*, where the robot continues to activate the same actions even though they have proven not to result in any change of state.

The problem cited above is a manifestation of the lack memory on the part of the competence module. One way of avoiding this problem is to provide the module with the capability of more *dynamic* behaviour [Dorigo 93] by providing some form of memory or temporal processing capability. When considering the use of memory in learning systems, difficult design questions are raised: *what* is to be remembered, *how* it is to be remembered, and for *how long*. Such questions will have an obvious impact on the selection of the learning architecture and ability to adapt to new environments.

1.5 Simple Recurrent Networks

One type of architecture that addresses these questions is the Elman-type Simple Recurrent Neural Network ([Elman90],[Elman 93]), or SRN. An SRN is a *feedforward neural network* where the previous states of the hidden layer are made available as an additional bank of input units called the *context* layer. Feedforward neural networks and SRN's are more fully described in Chapter 2.

With the hidden output layer fed back, it is possible for an SRN to develop a short-term memory of its own encodings of previous states. It can learn to respond to short action *sequences*. More importantly, time is represented implicitly by the effect it has on processing as opposed to making it another dimension to the input vector. This elegantly answers *what* is to be stored and for *how long*. Finally, neural networks belong to the cadre of connectionist

architectures which, by their very nature, answer the question of *how* the information is to be stored. That is, the system constructs its own internal representations built directly from the sensor readings to achieve the desired control behaviour.

Some research has been conducted using recurrent neural networks as part of successful control algorithms (eg. [Floreano96][Tani96]). In particular, Meeden has investigated the use of Elman type SRN's in the context of *immediate reinforcement learning* [Meeden94]. Immediate reinforcement learning is the machine learning paradigm assumed by the BAT methodology which suggests that Elman type SRN's may be useful building blocks for constructing dynamic competence modules.

1.6 Speeding Up Learning With SRN's

Meeden [Meeden94] uses *Complementary Reinforcement Back-Propagation* (CRBP) to provide the necessary Elman network weight and bias updates. In this work, Elman-type Simple Recurrent Networks were used in the design of each competence (or *behaviour*) module and CRBP was also used to train these networks. Meeden's implementation of CRBP uses a uniform distribution of random numbers as the basis of the algorithm's exploration function. In this work, it was determined that the target behaviour of each module could be reached faster by basing the exploration function on a Gaussian distribution of random numbers.

1.7 Thesis Objectives

The objectives of this thesis may now be summarized:

- To develop an interface between the Khepera mobile robot and PC-based Matlab which will allow the implementation of robot control algorithms in real time on the Khepera

from within the Matlab environment.

- To use a Behaviour Analysis and Training Methodology to design a robot control architecture whose task is to seek and approach a light source while avoiding obstacles.
- To evaluate the usefulness of the BAT methodology when combined with the Matlab-Khepera interface and applied to the implementation of the light seeking and approaching task.

1.8 Thesis Outline

Chapter 2 provides background into reinforcement learning, recurrent neural networks, and the Complementary Reinforcement Backpropagation algorithm. It also gives an overview of the stages of the BAT methodology. Chapters 3 through 7 describe the design stages of the BAT methodology as they pertain to the development of the controller in this work. In Chapter 8, the relevant aspects of the work are discussed, conclusions are drawn, and suggested areas for future research are identified. The Matlab-Khepera interface development and evaluation work is described throughout Chapters 3 to 7.