Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems

David McMinn

A thesis submitted in partial fulfilment of the requirement for the degree of Doctor of Philosophy awarded by The Robert Gordon University

This research programme was carried out in collaboration with Eident Ltd.

December 2001

ABSTRACT

The research presented in this thesis examines the area of control systems for robots or animats (animal-like robots). Existing systems have problems in that they require a great deal of manual design or are limited to performing jobs of a single type. For these reasons, a better solution is desired.

The system studied here is an Artificial Nervous System (ANS) which is biologically inspired; it is arranged as a hierarchy of layers containing modules operating in parallel. The ANS model has been developed to be flexible, scalable, extensible and modular. The ANS can be implemented using any suitable technology, for many different environments.

The implementation focused on the two lowest layers (the reflex and action layers) of the ANS, which are concerned with control and rhythmic movement. Both layers were realised as Artificial Neural Networks (ANN) which were created using Evolutionary Algorithms (EAs). The task of the reflex layer was to control the position of an actuator (such as linear actuators or D.C. motors). The action layer performed the task of Central Pattern Generators (CPG), which produce rhythmic patterns of activity. In particular, different biped and quadruped gait patterns were created. An original neural model was specifically developed for assisting in the creation of these time-based patterns.

It is shown in the thesis that Artificial Reflexes and CPGs can be configured successfully using this technique. The Artificial Reflexes were better at generalising across different actuators, without changes, than traditional controllers. Gaits such as pace, trot, gallop and pronk were successfully created using the CPGs. Experiments were conducted to determine whether modularity in the networks had an impact. It has been demonstrated that the degree of modularization in the network influences its evolvability, with more modular networks evolving more efficiently.

Acknowledgements

I would like to thank the following people:

- Grant Maxwell (my Director of Studies) for his guidance and supervision
- Chris MacLeod for critically evaluating my work
- My friends and family for their support and encouragement

Contents

Table of Figures

Chapter 1. Introduction

1.1 Introduction to chapter

This chapter introduces the problem under consideration, background, objectives and thesis structure. The contents of the following chapters will be outlined at the end of this chapter.

The research presented in this thesis describes the investigation of a new biologically inspired model for an Artificial Nervous System (ANS). Evolutionary Artificial Neural Networks are used to implement the model. These three elements (nervous systems, neural networks and evolution) are based on computer simulations of their biological equivalents.

The main target of this project is to create a flexible, modular, general purpose control system for robots and other electronic devices, based on a framework suggested as an idea for further study in a PhD thesis by C. MacLeod [MacLeod 1999]. The idea is to create a feasible engineering solution, which incorporates the best ideas from nature and electronics. Given that all animals have common structures to their nervous system (albeit some are more advanced) and have evolved over millions of years to help the animal survive, it would be a logical starting place for ideas. However, to take full advantage of an Artificial Nervous System one would want to make use of the speed of electronic devices. Therefore, this approach combines the best of both worlds: the elegance of nature and the efficiency of modern technology.

The Artificial Nervous System is used to collect different Artificial Neural Networks together to provide a complete system for controlling a robot or other electronic device. Artificial Neural Networks are an attempt at creating Artificial Intelligence by modelling the structure of the brain. Artificial Evolution is an optimisation technique (based on Darwinian Evolution) which can be used to create and optimise Artificial Neural Networks (as well as having many other applications). A brief background to these elements is given in sections 1.2 to 1.5.

Important related aspects such as neuron functionality (how the operation of neurons affects the task of the neural network) are considered. The scope of the overall ANS is large and covers several research projects and therefore only the lower levels of the ANS are considered here.

1.2 Artificial Neural Networks

Biological nervous systems are made up from networks of interconnected processing units, called neurons. Each neuron is a single living biological cell and can only process a small amount of information using electro-chemical signals.

Most neurons have evolved to become specialised and take part in performing specific tasks. However, there are also a great number of “general purpose” neurons present in biological systems. Although the processing capability of a single neuron is minute, the fact that there are many interconnected neurons (in the order of 10^{11} in the human brain) in these networks means they can perform more complex tasks.

This inspired researchers to try to create artificial intelligence with the same techniques: using a number of simple processing units interconnected in a network to perform more complex tasks. The name for this approach is Artificial Neural Networks or ANNs, implying that the neurons in the network are not biological.

1.3 Artificial Evolutionary Techniques

Biological evolution was introduced as an explanation for the survival of different species of animals. Animals which are well adapted to the problem of surviving in the natural world lived on, while other animals, which did not perform well, died. In this sense, biological evolution is efficient at finding the most optimal solutions for surviving.

The biological brain has evolved as the control system for these animals. This has encouraged researchers to use Artificial Evolution to create Artificial Neural Networks in their attempts at creating Artificial Intelligence.

Artificial Evolutionary techniques are similar to biological evolution in the sense that they are a method of optimisation (and are not limited to neural networks). In the work presented in this thesis, evolutionary techniques are used to create and optimise the ANNs.

1.4 Nervous Systems

Biological nervous systems are the collections of specialised neural networks in an animal that give it intelligence and the ability to control its body. Biological nervous systems have evolved over millions of years and have generally increased in complexity and size in higher animals.

Only simple invertebrate and lower vertebrate nervous systems are fully understood due to their highly structured organisation and low level of complexity. More complex nervous systems are also structured and it is possible to identify the structures common across most nervous systems, although the added complexity can alter the functionality of components.

An Artificial Nervous System (ANS) can be described as a controller for a non-biological system. It performs the same task as the nervous system in an animal: to control and regulate the body and allow the animal to interact with and survive in its environment. A formal plan for an ANS, based on ideas from Biological Nervous Systems, is proposed in chapter 2 of this thesis and serves as a framework for the investigations undertaken in this project. The work presented in this thesis is an extension of the work in [MacLeod 1999] which originally proposed the framework of the ANS under investigation.

1.5 Limitations of Current Implementations

Artificial neural networks work like their biological counterparts; they use a number of simple interconnected processing units to accomplish a more complex task. However, this is only possible when the ANN is relatively small (compared to biological networks) and the task is not too complex. For example, it would be impractical to create a single ANN to fulfil the functionality of the entire brain of an animal and create all its different behaviours. The network would be too large and would contain many unused or unnecessary connections. Training or evolving an ANN of that size would be an incredibly time consuming task.

Artificial Nervous Systems (in the sense used in this thesis) are currently used for the control of mobile robots. Usually these are not (or are only partially) made using artificial neural networks, with the majority of the system being programmed manually. This is because each part of the system takes on a fairly complex or large task. Most artificial nervous systems are frequently tailored to a single specific task and are reliant on a specific robot configuration.

1.6 Objectives

Due to the limitations mentioned in section 1.5, there is a need for a flexible and extendible ANS. Decomposing the problem down into more specialised parts and structuring in a suitable manner allows ANNs to be created for each part of the ANS.

The specific objectives of the project were:

1. Literature survey of previous work in the area
2. Study of biological nervous systems and refinement of the ANS model
3. Implementation of a basic single reflex system
4. Comparison against other published work
5. Implementation of a multiple reflex system
6. Construction of an evolutionary co-ordinating network
7. Consideration of the neural functionality aspect of the network
8. System testing and comparison with published results

1.7 Unique Aspects

The ANS framework, which is under investigation during the course of this research, is unique. The study and implementation of the parts of the ANS during this research is original as this is the first work on this ANS model. The refinements made to the ANS model are unique to this thesis.

Original neural models have been developed for use in Central Pattern Generator (CPG) neural networks. This allows the CPG networks to create all their patterns of activity internally, rather than rely on a specifically sequenced input stimulus.

Although the use of Evolutionary Algorithms to create ANNs is not new, it is the combination of all these parts, to produce a viable engineering solution for controlling a robot that is unique.

1.8 Thesis Structure

The biological background to this project is explained in depth in chapter 2. The artificial equivalents are also explained and the framework of the ANS is presented.

Chapter 3 looks at how the ANS, under investigation in this thesis, compares with other related work about robot control systems. The methods used in finding this literature are also covered.

Biological reflexes are detailed in chapter 4. The workings of muscles and the neural networks which control them, along with how this gives rise to movement, is explained in this chapter.

Following that are the artificial reflexes created as part of this research. The theory behind the artificial reflex is presented, along with a comparison to biological reflexes. The creation and operation of the artificial reflexes is considered with a comparison to other control methods. This is presented in chapter 5.

Chapter 6 describes the biological and theoretical operation of actions. Central Pattern Generators (CPGs) and multiple reflex systems are also described.

The construction and operational results of artificial actions, which are compared to biological CPGs and other similar artificial methods of controlling actions are detailed in chapter 7.

Chapter 8 discusses the ideas and research presented in the previous chapters of the thesis, and suggests alternative methods, improvements and explanations for specific features of the research.

Finally, the original contributions, conclusions of the work presented in this thesis and ideas for further work are given in chapter 9.

Chapter 2. Background

2.1. Introduction to chapter

This chapter introduces the biological background to the project and describes the artificial equivalents.

Nervous systems, neural networks and evolution are all considered in the biological background sections.

Finally, the Artificial Nervous System (ANS) framework (which is the main idea behind this research) is explained along with Artificial Neural Networks (ANNs) and Artificial Evolution.

2.2. Biological Nervous Systems

The Biological Nervous System (BNS) is the control system of an animal. It works to keep the animal alive by interacting with its environment and regulating the internal systems of the body. The increasing complexity of the creature’s body and environment have meant that the nervous system needs to be more complex to keep it alive.

The lowest invertebrates have nervous systems which are distributed across the body of the animal. This type of nervous system is called a “nerve net” and usually displays only simple reactive behaviours.

As the nervous systems of these creatures advanced during evolution, the nervous system became less distributed. An early example of this is the Hydra. Although the nervous system is still a nerve net, a large number of the neurons can be found around the mouth of the animal and on the tentacles [Buchsbaum 1968].

This process is called cephalization and can be seen more clearly in higher invertebrates and vertebrates. The body of the animal evolved to include special sensory organs, which are located at one end of the animal (usually the head). All the neurons involved in controlling and processing information from these are local to the organs. Since most of these organs are physically close to each other, the groups of neurons associated with them are also close and ultimately gathered in one area (the brain). The spinal cord carries information to and from these higher centres of the nervous system and performs local neural processing for the body parts not controlled by the higher centres.

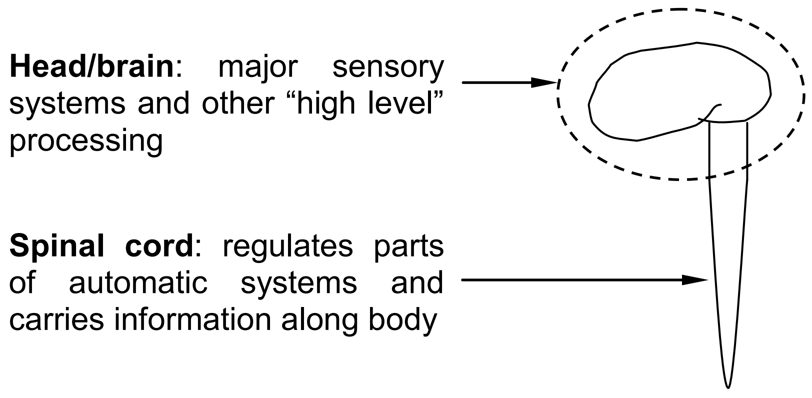

Head/brain: major sensory systems and other "high level" processing
Spinal cord: regulates parts of automatic systems and carries information along body

Figure 2.1: Distribution of nervous system connections

Vertebrate nervous systems can be considered as two parts: the Central Nervous System (CNS) and the Peripheral Nervous System (PNS). The PNS consists of the afferent and efferent nerves connecting the CNS to the body of the animal. The CNS comprises of the brain and spinal cord. All vertebrates have a common structural organisation of the CNS [Sarnat 1981]. More advanced nervous systems contain additional structures, rather than replace existing structures. Due to this reason, it is possible to find the same structures and functionality in parts of the nervous system of most vertebrates.

Reflexes are present in all vertebrates, and are the simplest form of movement control. These are also present in lower invertebrates, such as the Hydra. In Hydra, the nerve net is distributed across the whole body, with neurons acting as both sensors and actuator stimulators. When a stimulus is presented to the neurons, they cause the body to contract at that location. At the same time, the neuron propagates the stimulus to the other neurons it connects to, although with diminishing levels. Reflexes in vertebrates perform a similar local processing of stimuli, although without a nerve-net the processing takes place between an afferent (sensory) and efferent (motor) neuron pair. Reflexes will be discussed in more depth in Chapter 4.

Central Pattern Generators (CPGs), which create automatic rhythmic movements, have been shown to exist in the spinal cord of most vertebrates, such as, fish, cats and humans. These rhythmic movements produce actions such as walking, swimming, scratching and eating. Chapter 6 deals with CPGs in detail. Experiments on the brains of vertebrates have shown that stimulation of similar structures in different species have similar effects.

2.3. Biological Neural Networks (BNNs)

BNNs are essentially biological circuits used for processing data. The data, which is encoded as electrical impulses when travelling through the BNNs, is information about the animal's environment, internal status, muscle control and other regulatory signals, memories, thoughts and any other information which helps the animal stay alive.

Inputs to BNNs can be from neurons in other BNNs or from sensors on the body of the host animal. Similarly the outputs can connect to other BNNs or the actuators (muscles) of the body.

Networks of interconnected neurons are the lower level structure of BNNs. Although individual neurons only have a limited processing ability (as will be shown later), greater numbers of neurons (connected and operating in parallel) mean the BNN can perform more complex tasks.

2.3.1. Structure of Biological Neurons

Although there are many different types of neurons across all species of animals (and many in just one type of animal), all neurons share a similar structure. This is shown in Figure 2.2.

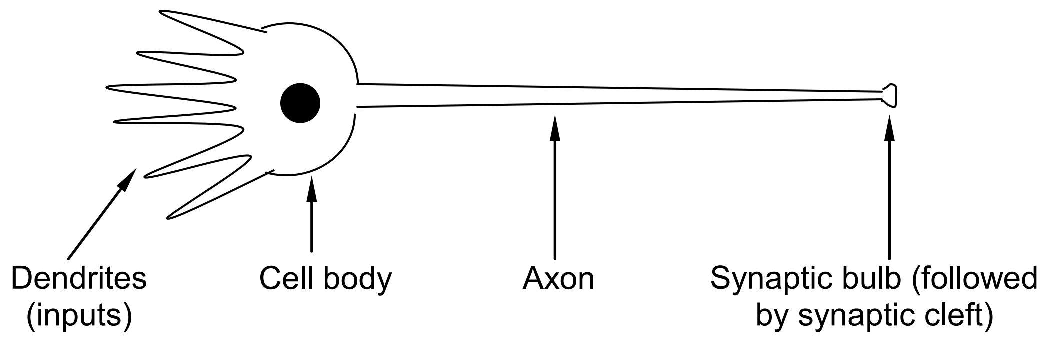


Figure 2.2: General structure of a neuron

The dendrites are the inputs to the neuron; they receive nerve pulses from the axons of other neurons. In afferent neurons, these connect from the sensors on the body. The cell body is the major part of the neuron as this is where the neural processing takes place. When the inputs to the neuron are sufficiently strong, it produces a pulse which travels down the axon and ends in the synaptic bulb. The synaptic bulb is responsible for transferring the output of one neuron to the input of the next neuron, where there is a small gap called the synaptic cleft. Efferent neurons (also called motor neurons) connect to muscles instead of other neurons. Interneurons are ones which have their inputs and outputs connected only to neurons (and not sensors or muscles).

2.3.2. Operation of Biological Neurons

The cell body contains a fluid (the intracellular fluid) composed of positively charged potassium ions (K+) and negatively charged chloride ions (Cl-) and proteins. The extra-cellular fluid (outside the cell body) is rich in positively charged sodium ions (Na+) and Cl- ions. It is the balance between the extra-cellular and intracellular fluids that creates the membrane potential (voltage difference across the membrane of the cell) and nerve pulses. The membrane of the cell body has a different permeability to each of these ions.

At rest (when there are no inputs applied to the neuron) the concentration gradient (the difference between the concentration of ions inside and outside the cell) forces K+ ions out of the cell body. This causes the membrane potential to decrease which opposes the outflow of K+ ions. Na+ ions also flow into the cell at the same time although, due to a lower permeability of the membrane, at a much lower rate. This process continues until equilibrium is reached, when the membrane potential is about -70mV, as the majority of positive ions are now outside the cell. The balance is restored by the sodium-potassium pump which exchanges intracellular Na+ ions with extra-cellular K+ ions.

Inputs on the dendrites, caused by the neurotransmitters from connecting neurons, increase the permeability of the membrane to the Na+ ions by opening certain channels. These channels are called transmitter gated Na+ channels, since they are activated by neurotransmitters and allow increased flow of Na+ ions. As the channels open, the membrane potential of the cell increases. When the voltage is high enough, voltage gated Na+ channels open (as these are controlled by the membrane potential). These allow a much higher flow of Na+ ions, and cause a large, quick increase in the membrane potential. This increase is the cell depolarising and reaches a peak of about +35mV.

As more Na+ ions flow into the cell and the membrane potential increases, increasing quantities of K+ ions are forced out and at the peak membrane potential, voltage gated K+ channels open, causing a massive outflow. The membrane potential decreases rapidly as the cell repolarises, until it is at a level lower than the membrane potential at rest when the K+ channels close. This process of depolarisation then repolarisation produces an action potential (nerve pulse) which is transmitted along the axon to the synaptic bulb.

At this lower membrane potential the cell is said to be hyperpolarised. An increased amount of inputs (compared to the rest state) is required to cause another action potential. The membrane potential gradually decays back to the resting potential of about -70mV. This entire process is shown in Figure 2.3.

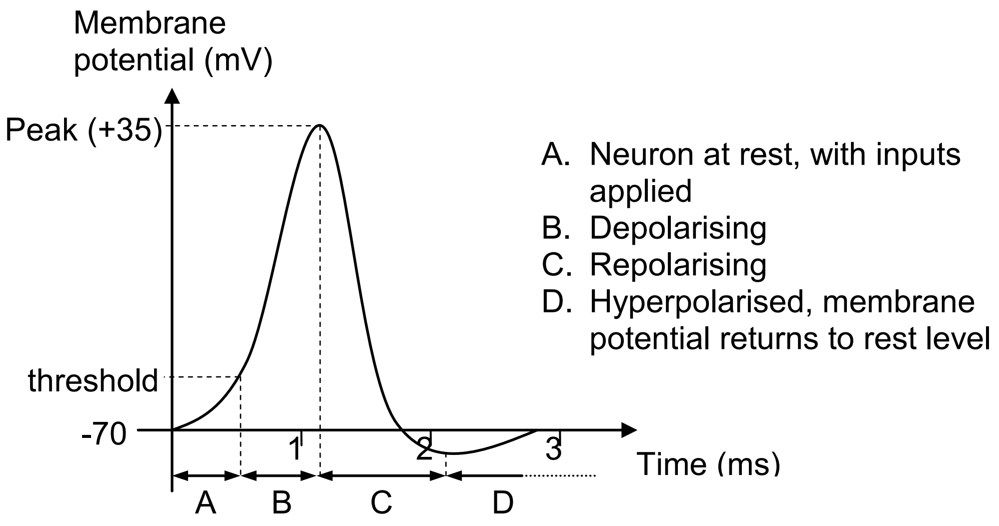


Figure 2.3: Timing of an action potential

A single action potential can take as little as 2 to 3 milliseconds and can travel along the axon at speeds up to $120ms^{-1}$.

When the action potential reaches the synaptic bulb, it will be transmitted to the following neurons either electrically or chemically. Electrical transmission is not as common as chemical, and its role in nervous systems is not yet fully understood

[Levitan 1997].

When action potentials are transmitted electrically, the synaptic cleft is much smaller than in chemical transmission (2nm as opposed to 20nm). The action potential passes directly across this gap using the same method of ion flow as the cell body. This allows for much quicker conduction of the signal and an enhanced level of stimulus. Electrical synapses also allow data to be transmitted bidirectionally.

During chemical transmission, the action potential causes neurotransmitters to be released from the synaptic bulb. These cross the synaptic cleft and temporarily bind to the dendrites of the next neuron, causing an input. Different neurotransmitters have different effects on the inputs, and some may even affect the more general area around the neuron. Not all neurotransmitters are known and, of the ones that are, it is unknown what all of the effects are [Ganong 1995].

Certain neurotransmitters are known to excite or inhibit the stimulus of a post-synaptic neuron, and others can cause fast or slow acting responses, by altering the working of channels on the cell body.

2.4. Biological Evolution

Biological evolution is more complicated than the artificial evolution used in optimisation problems (which will be described later).

2.4.1. Genetic basis of evolution

Cells exist in every living organism. Within the nucleus of every cell, the chromosomes of the organism (which are present in pairs) can be found. Different organisms contain different numbers of chromosomes, although it is not necessarily the most advanced organisms that have the most. These chromosomes are strings of genes, each gene storing a particular characteristic about the organism, and always being in the same place in the chromosomes. The reason the chromosomes are paired is that two parents each contribute half of the genetic material. Every gene is either dominant or recessive, determining which parent the characteristic for that gene comes from.

The collection of genes in an organism is called the genotype and the external characteristics (what the genotype translates to) are the phenotype. Genes can be further broken down into chemical components, deoxyribonucleic acid (DNA) and ribonucleic acid (RNA).

2.4.2. Processes of Evolution

The genes are altered by the processes of mutation and recombination. The changes in the genes are passed through generations under Mendelian inheritance [Jenkins 1998]. Mutation is any change in the chemical structure of a gene and may or may not cause major changes in the organism's chemistry.

Recombination occurs when new organisms are created by sexual reproduction. The two parent sex cells (the gametes) contain exactly half the number of chromosomes that make up the full organism. As the cells divide in a parent to produce gametes (meiosis), the paired chromosomes can swap genes. Two offspring from the same two parents are unlikely to be the same due to the random nature of this process. When the gametes of two parents combine, they form a single cell, with the full complement of chromosomes, out of which a new organism grows.

These are the methods for changing the genetic material in organisms, but there must also be reasons why different organisms evolved. The main reasons are listed below (although there are also many other factors which affect something as complex as evolution).

Darwin proposed the idea of Natural Selection [Darwin 1859]. Organisms which are best suited to surviving in a changing environment will survive in larger numbers and therefore are more likely to pass on their genes to the next generation. The best suited organisms are said to be "fitter" at the task of surviving in their environment, and so Natural Selection is sometimes called "survival of the fittest". This is the major idea on which artificial evolution is based, although this is probably not the only mechanism for biological evolution.

"Punctuated equilibria" [Eldredge 1972] suggested that species did not continually and gradually evolve as Darwin thought. Instead, there were long periods of equilibrium between the species (stasis) where little or nothing changed, followed by sudden punctuation (change) where species diverged (speciation), probably due to rapid changes in environment. Stanley later elaborated on this and called it “species selection” [Stanley 1975]. This can be seen as the different survival of species, which can be traced back to a common ancestor, as a result of different rates of speciation and extinction.

Mass extinction has been a major cause of change in life. The most severe mass extinction happened at the end of the Permian era (with an estimated ninety five percent of life wiped out). Another mass extinction caused all the dinosaurs to die at the end of the Cretaceous period. Popular explanations for these extinctions include meteorite impacts and volcanic activity.

2.5. Artificial Nervous System Framework

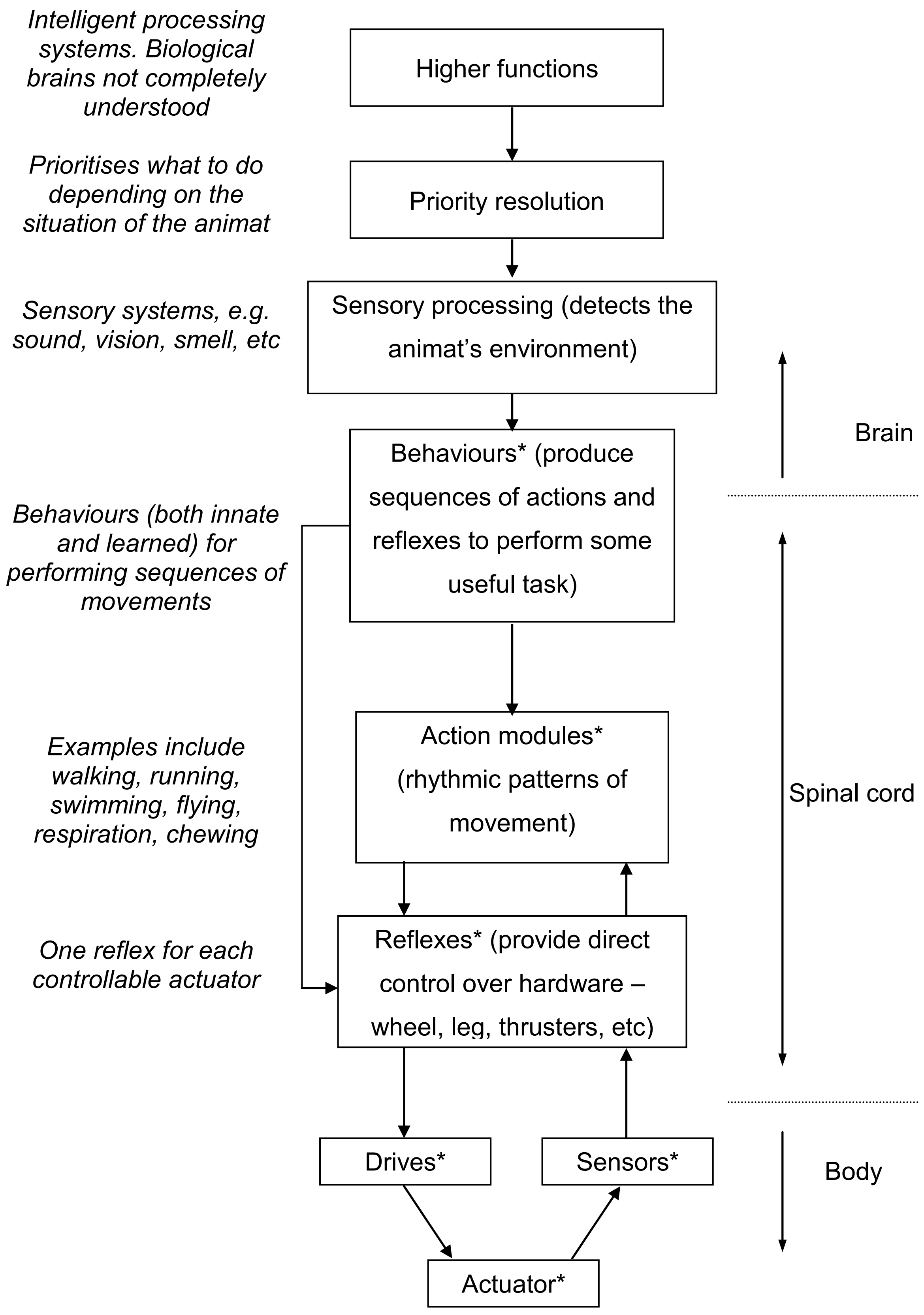
2.5.1. Structure of ANS

A biologically inspired framework for an ANS was originally proposed in [MacLeod 1998]. This is hierarchical in structure, as can be seen in Figure 2.4 (which shows the ANS with refinements added during this research). Those layers marked with an asterisk can have multiple modules in parallel.

The ANS is structured similarly to the biological nervous systems of vertebrates. Similar functional structures can be found in the same location in the ANS as in biological nervous systems (e.g. reflexes are the lowest level in the ANS, which corresponds to the simplest functions developed in biological nervous systems).

Layers are controlled by the modules in the one above and some receive information from the layers below. This type of information flow is also present in biological nervous systems [Kandel 1991], although in complex nervous systems (such as in humans), there are many more connections between the layers, and connections which skip layers.

The lower layers which are present in biological nervous systems are operational as soon as the animal is born. The corresponding layers in the ANS can be fixed, although, as they receive connections from the layers above, their operation can be modified. Layers higher up in the structure become more like the higher centres in the biological brain, and thus are required to be more plastic and capable of learning.



Intelligent processing systems. Biological brains not completely understood
Higher functions
Prioritises what to do depending on the situation of the animat
Priority resolution
Sensory systems, e.g. sound, vision, smell, etc
Sensory processing (detects the animat's environment)
Brain
Behaviours (both innate and learned) for performing sequences of movements
Behaviours* (produce sequences of actions and reflexes to perform some useful task)
Examples include walking, running, swimming, flying, respiration, chewing
Action modules* (rhythmic patterns of movement)
Spinal cord
One reflex for each controllable actuator
Reflexes* (provide direct control over hardware – wheel, leg, thrusters, etc)
Drives*
Sensors*
Body
Actuator*

Figure 2.4: Artificial Nervous System structure

This ANS can be applied to the control of animal-like robots, known as animats. However, the general structure can be applied to other similar problems.

2.5.2. Layers of the ANS

The reflex layer is the lowest layer in the ANS. This corresponds to the reflexes in biological animals, which provide the simplest and most automated forms of movement. The reflexes are used to directly control the hardware of the animat. Multiple reflexes can be applied in parallel, each operating independently, and each controlling a separate sensor-actuator pair. Chapter 5 deals with the reflex layer of the ANS.

The action layer corresponds to the CPGs which are present in the spinal cords of vertebrates. These produce rhythmic, semi-autonomous actions by controlling the reflexes. Examples of these would be walking, running or swimming. These action networks may be hierarchies within themselves; one main CPG to coordinate e.g. limbs, with sub-CPGs controlling the different joints of each limb. The action layer is explained in chapter 7.

Some biological animals have innate behaviours, such as eating, mating and predator avoidance. These would be represented in the behaviour layer, and activate the required sequence of actions (or a mixture of actions and reflexes).

Details about the animat's current environment would be acquired in the Sensory Processing layer, using for example, vision or sound. Depending on the situation of the animat, the required behaviour will be triggered. For example, if the animat senses a predator in the vicinity, the behaviour for avoidance would be initiated, which in turn activates the "turn" action followed by the "run" action.

The priority resolution layer gives precedence to important tasks depending on the situation and internal status of the animat. This is connected to the higher layers of the ANS. At this point, the biological nervous systems are more complex and not fully understood. However, the ANS at this point would perform tasks such as skilled movement, task planning, as well as other intelligent processes.

A breakdown of the ANS model showing more specific connectivity between the layers is shown in Figure 2.5.

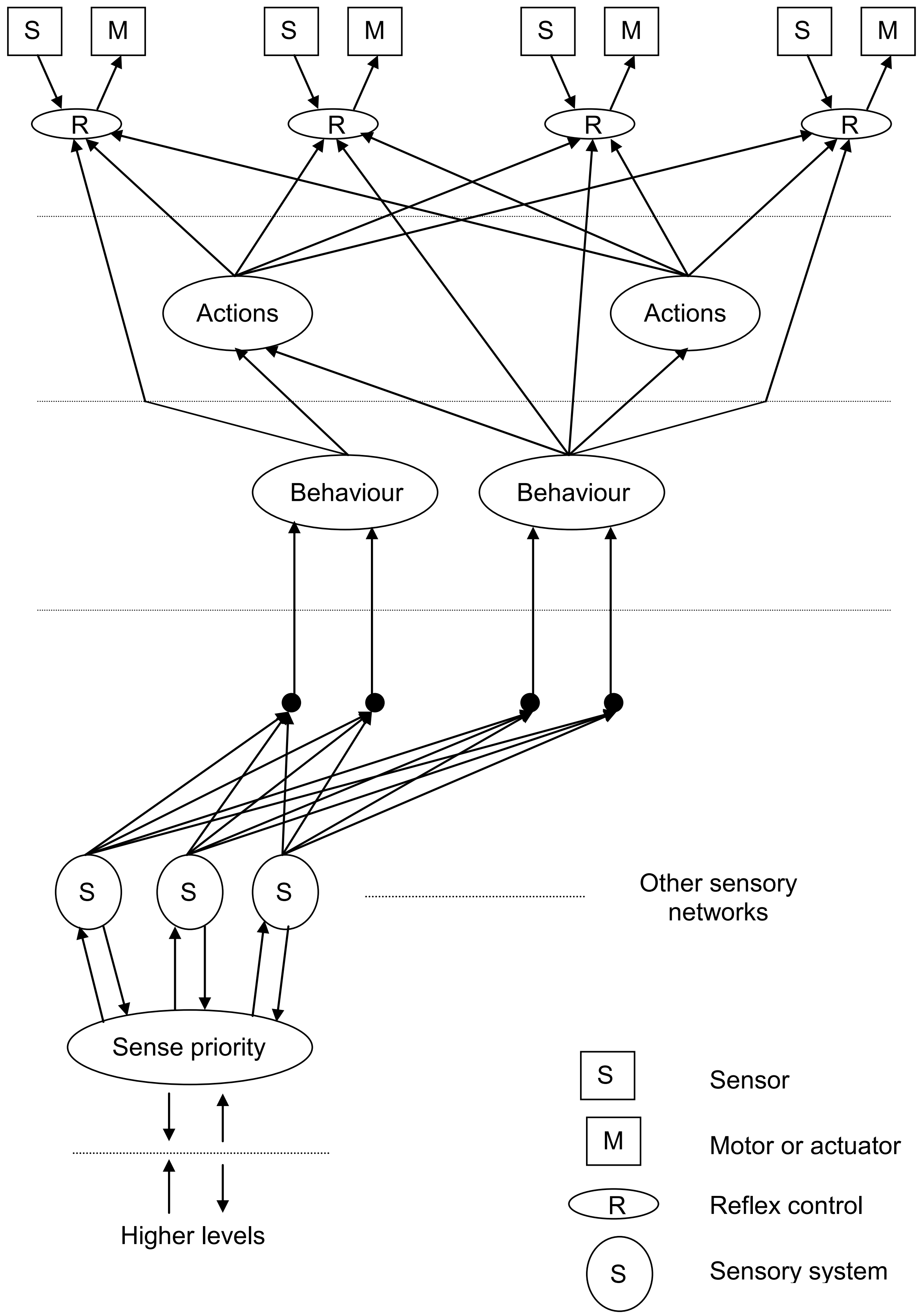


Figure 2.5: Detailed connectivity between the layers of the ANS model

2.5.3. Advantages of the ANS

The main advantage of this system is that it is flexible. New tasks can be performed simply by adding new modules to a particular layer. Modules can be kept relatively simple, as they will use the functions provided by a lower level to achieve their task. The model is also flexible in that it may be implemented using any convenient technology. The use of artificial intelligence techniques (such as artificial neural networks) allows the system as a whole to be general and not dependent on the particular implementation of a piece of hardware within the animat. However, if the hardware did require a change in the lower layers, it would only be those lower layers that would require modification; the upper layers could be kept as they are. This desirable property means the model is applicable to many robot hardware configurations, such as walking, swimming or wheeled robots.

The model is also flexible enough so that it can be applied to many different kinds of animats. For example, if the system was used in ROVs (Remotely Operated Vehicles) then the upper layers of intelligence could be replaced by a human operator. Similarly, if the system was to be used to control an autonomous vehicle, then the lower layers could be used (to control the hardware of the robot) by a programmed task system. It could also be integrated with other intelligence techniques such as Expert Systems or Fuzzy Logic.

2.6. Artificial Neural Networks

2.6.1. Overview

The subject of Artificial Neural Networks (ANNs) covers a large area, and as such, the reader is advised to look at a standard textbook, such as [Wasserman 1989a], for further explanations.

ANNs are a method which attempts to create intelligent computing devices using the same principles that biological nervous systems are based on. Simple processing units are used to perform more complex tasks when many are connected in parallel.

ANNs have been successful in applications where the task is constrained in terms of the possible inputs and required outputs. ANNs are usually designed for a specific purpose, such as to be used in character recognition or image reconstruction, in which the range of inputs and outputs can be pre-determined. However, as the task grows in size and becomes less constrained, the efficiency of ANNs reduces. This is because there is no single network type or training algorithm to suit all tasks.

2.6.2. How an ANN works

The generalised view of most types of artificial neuron is shown in Figure 2.6. This is the "McCulloch-Pitts" neuron [McCulloch 1943].

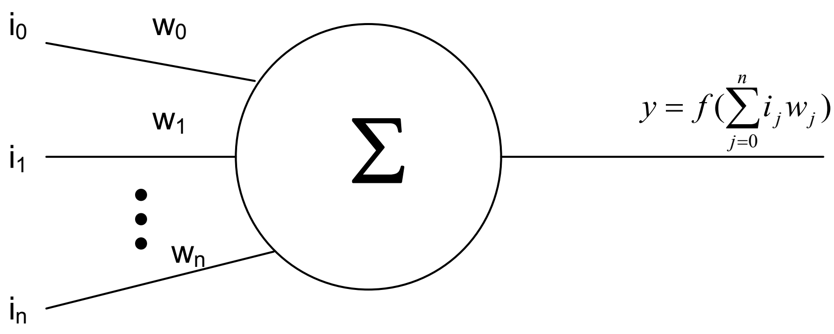


Figure 2.6: Generalised view of an artificial neuron

The weights of the connections represent the strength of the neurotransmitter in biological neurons. The total input to the neuron is calculated as the weighted sum of all inputs. The output is some function of the input, known as the threshold function. This was defined as McCulloch and Pitts to be a binary threshold, although most commonly this is a member of the sigmoid family of mathematical functions.

ANNs contain a number of these artificial neurons connected together. A feedforward (or associative) network is one in which all the neurons connect between adjacent layers and there are no connections going back towards the inputs or within layers. A feedback (or auto-associative) network is one in which there are connections from neurons to neurons in previous layers. Generally it is better to construct networks with more than 1 layer, as having limited numbers of layers can limit the abilities of ANNs, as shown in [Minsky 1969]. However, it is generally accepted that a three layer network is the maximum required.

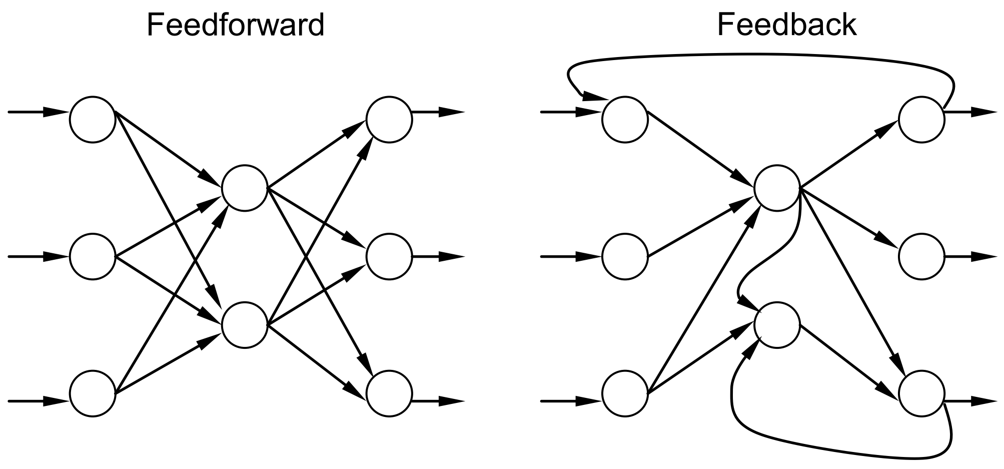


Figure 2.7: Feedforward and Feedback networks

2.6.3. Training the ANN

ANNs require training before they are capable of performing the desired task. There are two main categories of training; supervised and unsupervised. Supervised training requires that the inputs and desired outputs for those inputs (the targets) are known before training. Unsupervised learning does not require specific targets: the network will organise itself to produce the best outputs for any inputs.

The most common method of supervised training is Backpropagation (BP) [Werbos 1974]. This is similar to Perceptron (feedforward network with one layer) learning [Rosenblatt 1962], but allows feedforward networks of any depth to be trained by propagating the error (between the targets and the outputs) back through the network and changing the weights accordingly.

Self Organising Maps (SOMs) are based on Kohonen neurons [Wasserman 1989b] and competitive learning. The Kohonen neurons in a layer behave in a “winner-takes-all” manner: When the outputs of the neurons in the layer are calculated, the neuron with the highest output has its output set to 1 and all the other neurons have their outputs set to 0. The neuron with the highest output then has its weights modified. This weight modification is based on Hebbian learning [Hebb 1949].

Other training algorithms generally apply to a specific type of network; for example, Hopfield networks [Hopfield 1982] are a type of feedback network with binary threshold neurons and a specific training method. The weights are calculated in a one-shot training run using an equation. Hopfield networks are useful in reconstructing a specific output from a corrupted input, and also in optimisation problems.

Statistical methods [Wasserman 1989c] are another useful method of training. These can be applied to any network configuration, and can avoid problems including lack of a suitable specific training algorithm and local minima. Random change is applied to a weight in the network and if the performance of the network gets better, the change is kept. If the performance worsens, then another random number is calculated and compared against the probability of the change being kept. This allows the performance of the network to occasionally worsen in favour of finding the optimum solution.

2.6.4. Advanced neuron models

Researchers working in fields that have closer ties to biology (such as biologically inspired robotics, psychology and biologists themselves) normally require artificial neurons which model the biological neuron more closely.

A common neuron model used in generating movement using ANNs in robotics is the Leaky Integrator model [Arbib 1989]. This simulates the dynamics of the biological neuron to produce time domain patterns of neural activity. The membrane potential of the neuron is given by

$$t_c \frac{d(m(t))}{dt} = -m(t) + \sum_{j=0}^{n} i_j w_j \qquad \text{Equation 2.1}$$

where tc is the time constant of the neuron, m(t) is the membrane potential, i_j is the input to neuron i from neuron j, and w_j is the weight connecting the neurons.

A more complex model is the Hodgkin-Huxley model [Hodgkin 1952]. The membrane of the cell and dendrites are both modelled as electrical circuits, which can also be represented as a series of differential equations. Whilst this does give quite accurate representation of neurons and action potentials, it is also the most computationally intensive and does not take into account the chemical transmission between neurons. This type of neuron model (and others like it) is classed as Compartmental Models, indicating that the neuron is modelled as the current flow through a series of compartments. However, this model is more commonly used to model real biological systems rather than in ANN simulations.

2.7. Artificial Evolution

Artificial Evolution is an optimisation technique based on biological evolution and the idea of natural selection. It can also be used for tasks such as training or creating ANNs.

2.7.1. Basics

There are three main methods of artificial evolution: Evolutionary Programming [Baeck 1995], Evolutionary Strategies (ES's) [Schwefel 1995] and Genetic Algorithms (GA's) [Holland 1975]. All three operate similarly, but have notable differences in their algorithms which will be described later.

In each method there exists a large number of chromosomes (the population), each chromosome being an encoding of a possible solution to the optimisation problem. Chromosomes contain a string of genes (known as the genotype) which are usually either single bits (binary digits) or real numbers, depending on the algorithm.

The general algorithm for performing artificial evolution is:

1. Randomly create initial population
2. Evaluate fitness of each chromosome
3. Select n chromosomes to become parents for next generation
4. Perform genetic operations on the parents to produce offspring to fill the next generation
5. If termination criteria not met, return to step 2

Termination of evolution can be determined by either the chromosomes reaching a satisfactory fitness level or passing a certain number of generations.

2.7.2. Fitness Evaluation

The fitness score of a chromosome is a measure of how good a solution it encodes. This is evaluated using an Objective Function (sometimes called the

Fitness Function). The measure of fitness is dependant on the problem which is being optimised, and choosing an appropriate Objective Function is critical to how the final solutions perform.

Before the fitness score of a chromosome is evaluated, it must usually be decoded. This decoded solution is known as the phenotype.

2.7.3. Selection of parents

To produce the subsequent generation of chromosomes, certain chromosomes are selected from the current generation to act as parents. These parent chromosomes have evolutionary operators (described in the next section) applied to produce children to populate the next generation.

The selection of parents is performed so that the average fitness score of the population increases with each generation. It can be performed in many ways ranging from the simple selection of the n fittest chromosomes, to tournament methods where parents are repeatedly chosen from the fittest chromosome out of n random chromosomes.

The more advanced selection algorithms, while all trying to improve the average fitness score, attempt to keep as much genetic variation in the population as possible. This prevents the population from going “stale” (all chromosomes being very similar) and provides more varied solutions to the problem. A stale population’s fitness will not increase as quickly, or will not improve, with further generations.

Selection can also be applied between parents and children, to identify which should pass through to the next generation. Methods where parents are present in the next generation are known as elitist.

2.7.4. Evolutionary Operations

The core operations are mutation and recombination (or crossover), which act in a functionally similar manner to the biological operations of the same names.

Mutation involves modifying genes, within the chromosome, chosen at random. This will either be inverting a bit or changing the real number by a random amount, depending on whether the genes are single bits or real numbers. This is used to add more variation to the population, enabling a global search of the solution space.

Crossover is the process of taking two (or more) parents and splicing their genes together. A position within the strings is chosen at random. One child is created from the first portion of the first parent and the second portion of the second parent. The other child is created from the first set of genes from the second parent and the second part of the first parent. This idea can be extended to use more parents and more crossover points to produce more children. Another possibility is to combine the equivalent genes from a number of parents (by taking the average) to produce one child.

Other, more complex, operators exist aimed at improving the efficiency of the algorithm at finding the most optimal solution.

2.7.5. Differences between algorithms

There are notable differences between the three main methods of artificial evolution. Table 2.1 shows these differences, according to the formal description of the methods.

Table 2.1: Differences between artificial evolutionary methods

Method	**Selection**	**Operators**
Genetic Algorithms	Probabilistic	Mainly crossover
Evolutionary Programming	Probabilistic	Only mutation
Evolutionary Strategies	Deterministic	Crossover and mutation

The selection method in a GA is probabilistic because the chromosomes are selected using their fitness scores as the probability that any one will be selected as a parent. In an EP the selection method is tournament based, with a chromosome competing against a number of other randomly selected chromosomes, with final selection being performed on the number of “tournament wins” each individual has. The deterministic nature of an ES arises from the fact that the best μ chromosomes from the population (chosen directly on having the best fitness scores) are selected as parents.

In addition to this, Evolutionary Strategies take one of two forms. The so called (μ, λ) version populates the next generation with the μ best chromosomes out of λ children. The alternative is to use the elitist ($\mu + \lambda$) version which populates the next generation with μ chromosomes from the best of μ parents and λ children from the current generation.

2.7.6. Artificial Evolution applied to ANNs

Artificial Evolution can be used in either a supporting role or a constructing role with ANNs. In a supporting role it can be used to pre-process ANN input data [Kelly 1991], select training parameters or methods [Schaffer 1990] or analyse the ANNs produced [Eberhart 1991]. In a constructing role, Artificial Evolution can be used to optimise either or both the weights (i.e. train the network) and topology of the networks.

While using evolution to train a network may not be as fast as using gradient methods (such as BP), it can escape local minima and converge to the optimum solution (given enough time).

It is possible to use simple genotype encodings when using evolution with neural networks. One method, used while training, involves each weight in the network being a string of genes. Each gene would be a single binary digit which gets combined to encode each weight as a binary value [Haupt 1998a]. Another encoding scheme would be to have each gene as simply a single real-valued number [Haupt 1998b]. In both cases, the gene(s) which represent all the weights in the network would be concatenated to become the chromosome (called direct encoding) [Montana 1989].

A similar strategy can be used for the topology. Each of the possible connections in the network could be a gene, each gene being a single bit. If the bit was ‘1’, it would represent a connection between two neurons and a ‘0’ would mean an absent connection [Miller 1989]. An example of this encoding is shown in Figure 2.8. It can also be extended to the previous case where the genes are the real valued weights in the network.

Problems can arise with these simple encodings when the network size increases. This would also increase the size of the chromosome, which leads to increased computation times and a larger evolutionary space to search. More elegant (indirect) encodings are required when this occurs.

[Yao 1999] provides descriptions and examples of all these topics.

From To	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	1	1	0	0	0
3	1	0	0	0	0
4	0	0	1	1	0

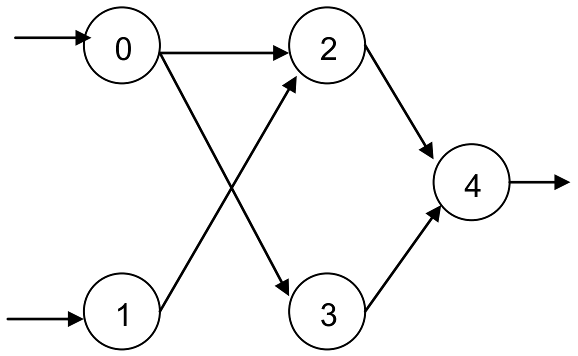


Figure 2.8: Example connection matrix and decoded ANN

2.8. Summary

This chapter has given a description of biological nervous systems, which included their overall structure and function. The operation of BNNs and the individual biological neurons has been shown. Evolution of biological entities has been explained in terms of function and the most common methods.

The artificial equivalents of the above biological items has been described. The ANS framework which is the basis of this research, and has a novel design, has been proposed. Background to the components which will be used to create the ANS (namely ANNs and Artificial Evolution) has been presented. Detailed accounts of the creation of the ANS at each layer will be given later, in the relevant chapters.

The next chapter will provide a review of the current literature in areas related to the Artificial Nervous System Framework described in this chapter. Work at all levels will be shown, from other ANS to single ANNs and neuron models.

Chapter 3. Literature Review

3.1. Introduction

Chapter 2 described the main ideas behind the research in the thesis. The purpose of this chapter is to review work by other researchers, which is in the same area (or is relevant) to the research presented in this thesis.

The sections of this chapter will cover a brief review of the history of robots, research into robot control systems, central pattern generator and reflexes. Finally, a summary will put the research presented here into context with the reviewed work.

3.2. Historical overview

The word robot was first used by the Czech author Capek in the play "Rossum's Universal Robot", in 1922. The actual word, robota, means "one in subserviant labour", which has been the main use for robots in real life (to perform some useful task under the control of humans). Soon afterwards (in 1939) the science fiction author Asimov wrote a series of short stories in which he introduced his famous three laws of robotics [Todd 1986].

The British Automation and Robot Association describes an industrial robot as:

> "a re-programmable device designed to both manipulate and transport parts, tools, or specialised manufacturing implements through variable programmed motions for the performance of specific manufacturing tasks."

The 1950's saw the start of robots being built commercially, with early robots starting to be used in the 1960's for industrial applications, especially manufacturing lines. These tended to be manipulators rather than mobile robots [Zeldman 1984]. Recently, robots have been used in many more fields, including entertainment, domestic use, industry, research, exploration, safety and anywhere else that it would be advantageous to have automation.

The deployment of robots is useful when tasks need to be carried out automatically, whether that be faster, more efficiently or for longer periods of time than if humans were performing the operation. Robots become more useful when they are versatile. For example, a robotic manipulator which has some degrees of freedom is more generally useful than an arm which has a fixed mechanical movement. Control systems are required to make these robots behave properly.

3.3. Robot Control Systems

3.3.1. Subsumption Architecture (SA)

SA was first introduced by Brooks, in [Brooks 1986] and is a popular system for controlling mobile robots, along with the principle behind it (behaviour based control). The system consists of a hierarchical layered structure, each layer defining a separate behaviour. The intelligence of the system can be seen to increase with higher layers, where behaviours become more specialised. For example, the lowest layer (called layer 0 in SA) might give the robot the behaviour of wandering whilst avoiding collisions. Layer 1 will be more specialised and may add the behaviour of moving to a goal position. These two layers would therefore create a SA architecture which is capable of moving to a goal position while avoiding collisions. However, there could be a possible conflict between moving to a goal position and avoiding obstacles where it would be undesirable for the upper layer to subsume the lower.

SA works because higher layers subsume the lower ones, a higher layer suppressing the behaviour of a lower. Alternatively, the inputs to modules in the lower layer can be substituted with inputs from a module in a higher layer. SA allows the robot to be functionally useful from the beginning, since each layer provides a fully working behaviour.

Subsumption architectures are usually shown as in Figure 3.1. This gives a false impression of the simplicity of SA. Layers are usually constructed from a series of modules, designed to implement the behaviour of the layer. The suppressive connections between layers are taken from internal connections in the higher layers to internal connections in the lower layer. Whilst this makes the connectivity between layers more complex, it simplifies the overall design as the same modules need not be recreated in every layer.

The original proposal for SA [Brooks 1986] only used suppression with higher layers always taking priority. Early success was found with the work of Connell which used SA to control a robot which moved around offices picking up and disposing of cans [Connell 1988]. A modified version of SA was presented in [Rosenblatt 1989] which demonstrated the advantages (more modularity, simpler design, finer control of tasks) of decomposing the behaviours into smaller decision making units.

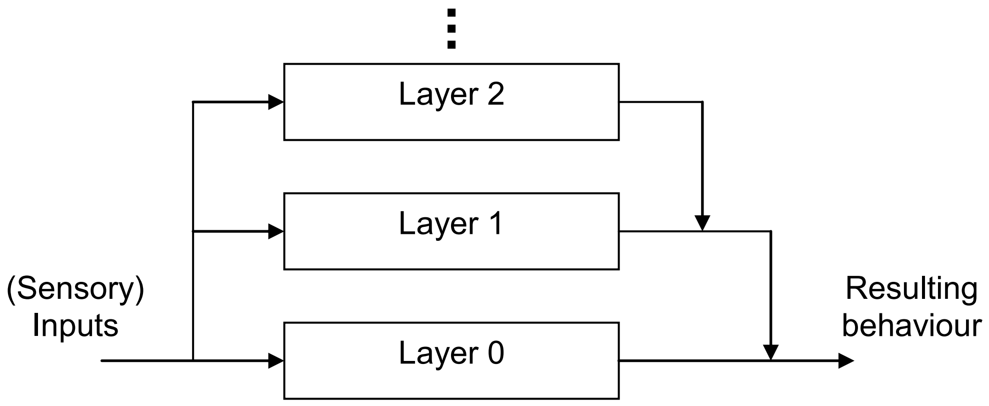


Figure 3.1: A typical subsumption architecture

More recently, SA has been used as the basis for modified robotic control systems, due to the limitations presented in [Hartley 1991].

An extension to the behaviour selection mechanism was described in [Maes 1992], allowing different behaviours to be selectively activated or suppressed. Further extensions of the SA approach are demonstrated in [Mataric 1992] where a map representation is integrated into the SA behaviours (traditionally, planning and reactive elements of a control system are separate).

In [Izumi 1997] the behaviours are all implemented as fuzzy rules (learned using a GA) and individual behaviours are selected by sign and saturation functions. Further work [Izumi 2000] demonstrates a hierarchy of different behaviour sets with the lower layers pre-defined and the higher layers learned.

The work explained in [Nakashima 1998] concerns a “dynamic subsumption architecture”. It is intended to be an enhancement to the original SA system, by being more modular, allowing behaviours to be learned, and subsumption can be between any modules in the system.

3.3.2. Hugo de Garis

Hugo de Garis has proposed a system for evolving an artificial nervous system (ANS) using Genetic Programming (GP) in [de Garis 1991a]. This system is based on GenNets: neural networks which perform a specific time dependant task. These GenNets are evolved using a GA; the designer specifies the parameters for the network, evolutionary system and the desired low level behaviour. Once there are enough GenNets to cover all the desired behaviours, they are combined using GP to form the ANS. Hierarchical nervous systems with more levels can be evolved using this approach, by repeating the process of combining multiple structures, through the use of GP, to produce another layer.

De Garis illustrates this idea by evolving a GenNet to generate a walking gait for a pair of simulated stick legs. This process has been used, by de Garis, to construct an ANS for an artificial lizard called “LIZZY”. LIZZY is capable of performing a number of different behaviours, as described in [de Garis 1991b]. These include (but are not limited to) approaching prey and mates and fleeing from predators.

The present work of de Garis centres around the CAM Brain project [de Garis 1994]. The project aims to grow artificial brains using EAs and Cellular Automata (CA) using custom built hardware to perform the evolution. The chromosomes of the EA encode a growth direction for each cell in either a 2D or 3D space. The CA space is seeded with neurons, from which the axon and dendrite connections are grown (following the paths specified by the directions in each cell the connections pass through). Synapses are formed when axons and dendrites cross. After the growth phase is complete, the ANN module is ready for fitness evaluation or use. Signals in the form of known single bit spike trains are used as the inputs and outputs for the network, from which the fitness can be evaluated [de Garis 1995]. More recent work [de Garis 2000] describes how the ANN modules can be connected together by a designer to form an artificial brain.

3.3.3. Other Robot Control Systems

Many hierarchical control systems will have three main components: a controller, a sequencer and a planner. The job of the controller is to control the hardware of the robot. The sequencer creates a series of movements for the controller to perform. The planner is responsible for generating a global sequence of movements to achieve its goal. The general flow of information and control within the hierarchy is from the planner to the sequencer to the controller. These are known as Three Layer Architectures (TLA) [Gat 1998]. Although the names of the three layers are sometimes different, the function of each remains the same.

The breakdown of the three layers follows the principle of increasing precision decreasing intelligence (IPDI) [Saridis 1992]. The top layer is the most intelligent (planner) but least precise (generating broad movement specifications). The precision increases and intelligence decreases through the layers until the controller which only performs automatic actions as given by the layers above it, but is the most precise (as it directly controls the hardware of the robot). Examples of this type of systems include [Bartolini 1995], [Sousa 1996], [Maurer 1997] and [Herbert 1998].

Rather than use a fixed control structure, Digney presents a system in which the hierarchical control structure is generated as the robot interacts with its environment [Digney 1997]. Specific sensory conditions for the robot are called features and the control strategy for a particular feature is a combination of primitive actions and other control strategies. The hierarchical structure emerges from the possibility of having nested and recursive control strategies.

Dellaert and Beer have created a system in which both the nervous system and body of an autonomous agent are evolved simultaneously [Dellaert 1996]. Based on the development of biological systems, the base genetic material is evolved and then develops into the mature organism. The genetic material is specified by a grid of cells, each having a specific type (such as sensor, axon or actuator) and properties. The cells are created using a model of cell division. When cell division ceases, a neural network is grown on top of the genetic substrate, to connect the sensors and actuators.

Initially, a hand designed chromosome, which encoded an agent for avoidance, was tested and found to perform appropriately in a simulated environment. However, when attempting to evolve an agent from scratch, the sheer complexity of the system proved excessively prohibitive, resulting in an inability to evolve an agent for the same avoidance behaviour. However, it was possible to improve the agent's performance through incremental evolution, seeded with the hand designed agent. By simplifying the model of evolution and development, agents were evolved which performed the avoidance task successfully. Further evolution of the agent also allowed it to follow a curved line.

An ANS model was proposed in [Jacob 1994]. The structure of this ANS was a fixed set of input and outputs (as required by the task to be solved) between which there is a pool of cortex modules. The cortex modules can be interconnected in any way, and some of them pass their outputs to the output neurons. Each cortex module can either be a neuron or a sub-network (chosen from a pre-defined library of subnetworks which have been previously designed by hand or evolved). The evolution of the ANS is performed in three stages; net connectivity, neuron functionality and weight settings. The former two stages are performed using GP while the latter is achieved with a GA. The approach is demonstrated by evolving ANSs to balance a ball on a see-saw and control a two dimensional stick walker.

3.4. Central Pattern Generators

3.4.1. Modelling Existing Biological CPGs

CPGs have been extensively studied in many animals, although they have only been fully understood in simple invertebrates and lower vertebrates [Shik 1976][Grillner 1981] due to the increasing complexity of vertebrate nervous systems. These studies of biological CPGs have been used to produce computer simulated animals, permitting further investigation and understanding of CPGs.

A commonly simulated CPG is that which controls swimming in the lamprey (a primitive, small, eel-like fish). The spinal cord of the lamprey has been studied in great detail, therefore making it a good candidate for this type of research. Buchanan and Grillner proposed a model of this CPG [Buchanan 1987]. Further investigations have tried to create highly realistic simulations of the lamprey by incorporating as much experimentally observed data as can possibly be understood [Grillner 1990], or by creating complex models of the CPG components (neurons and synapses) [Ekeberg 1991]. These realistic simulations have proven to be successful in explaining the observed experimental data (as shown in the 1991 Ekeberg paper and [Wadden 1993]).

Researchers have also investigated simpler solutions. Buchanan (in [Buchanan 1992]) has taken the model of the lamprey swimming CPG and simplified the neuronal model, although retaining the original connectivity. Whilst the neuron model used was simpler, it matched some of the important features of the more complex models, and therefore allowed a single segment to produce realistic oscillations. The connectivity between successive oscillator segments was investigated and a set of connections were found which enabled the CPG to create the oscillation patterns as found in the observed experimental data. The conclusion of this work was that connectivity in a CPG is important and sufficient for oscillations, rather than requiring a complex, realistic model. Ekeberg (in [Ekeberg 1993]) took this one stage further and added a mechanical simulation of a lamprey (the body and its environment) to a simplified model of the CPG. The CPG could be tested not only by comparing the patterns of oscillation to those produced by the biological CPG, but also to see if the artificial lamprey could swim. The simulated lamprey produced nearly all the swimming patterns of a real lamprey.

Ijspeert et al, in [Ijspeert 1997], have also used the model of the lamprey swimming CPG in computer simulations. In this work, GAs are used to evolve connection weights for a lamprey CPG for swimming. The network structure and neuron model are fixed to that of the biological model. The evolved controllers are compared to the original model, to verify that GAs are a suitable method for creating CPGs.

Further work by Ijspeert ([Ijspeert 1998]) has involved the creation of CPGs for multiple behaviours in a simulated salamander. The initial CPG is taken from the previous work on the lamprey, and the ability to produce walking patterns is added to it, allowing the salamander to both swim and walk.

3.4.2. General CPGs

The Cerebellar Model Articulation Controller (CMAC) architecture [Albus 1975] is one in which the learning processes and functionality of the cerebellum in animals is replicated. Since the cerebellum is used in the control of movements, it is logical to use it to control the positions of parts of an animat, whether this be the rhythmically cycling positions of legs during locomotion, or the precision of grasping movements. Benbrahim used adaptive CMAC networks as CPGs in a bipedal walking robot [Benbrahim 1997]. The CPG takes time, step period and step length as inputs and produces angular positions for the six joints on the legs as outputs. These positions are combined with outputs from other CMAC networks, which monitor robot status such as posture and body height, to control the actual joint positions. Some of the networks were pre-trained, while the main CPG network was trained using a Reinforcement Learning (RL) scheme, on a simulation of the biped. Once trained successfully, the networks were transferred to a mechanical robot, on which it also managed to produce correct walking patterns.

Cerebellar Adaptive Modular Behaviour (CAMB) was developed by Smith, and presented in [Smith 1996], as a method for producing a behavioural control system capable of learning. It is a hybrid of SA (described above) and the approach in [Beer 1991]. The important differences are that it adapts to the environment by learning and utilises a competitive behaviour selection. However, the behaviours and capacity for learning are fixed by the designer. Cerebellar Model Articulation Controller (CMAC, described above) neural networks are used in the CAMB model to learn the appropriate behaviour in a certain situation. In this work by Smith, a mobile robot is trained to explore its environment and to reach a goal position, while avoiding obstacles.

Research into using CPGs for legged locomotion has taken place over the last ten years. One of the first papers was [Taga 1991] which demonstrated walking in a simulated biped in two dimensions. This was later extended in [Miyakoshi 1998] to allow the biped to walk in three dimensions. Other researchers concentrated on producing CPGs which would give gait patterns for quadrupeds ([Akiyama 1995] and [Kimura 1998]). These were used to control real robot quadrupeds which are capable of various gaits over uneven terrain (including slopes, bumps and obstacles). The main feature of all these papers is that they use neural oscillators (each unit in the CPG is an oscillator with its behaviour described by differential equations) rather than neural networks. General frameworks for creating CPGs out of oscillators, and the different gait patterns they produce, are described in [Golubitsky 1998] and [Stewart 1999].

A similar approach was proposed in [Berkemeier 1995] where the legs of a quadruped were designed to include springs and dampers, thus causing them to act as oscillators.

A slightly different system was used in [Venkataraman 1997]. It still relied on oscillators, but instead of one per leg, there was one master oscillator which sent its output to a delay unit for each leg. The oscillator pattern of activity was simply delayed and then sent to the leg. This simple approach was shown to be capable of producing various gait patterns for bipeds, quadrupeds and hexapods.

Staged evolution (evaluating a different fitness function in sequential evolutionary runs) using a GA was used in [Lewis 1992] to evolve neural networks (consisting of leaky integrator neurons) to control the walking patterns of a hexapod robot. The staged evolution was used to first evolve an oscillator, then evolving a larger network of those oscillators to produce the gaits (successfully performing tripod and wave gaits).

The evolutionary creation of CPGs for walking robots has also been investigated in [Cao 1999] and [Reeve 1999]. Cao's work uses GAs to evolve neural networks for a specific bipedal gait pattern. One neuron in the network controls the joint angle it is associated with. A solution is found when the leg patterns match a set of designed rules. The work presented by Reeve studies a framework that can be used to create CPGs to produce dynamically stable walking patterns for robots, in three dimensions. That work includes investigations into the most appropriate neuron models, chromosome encoding and fitness functions for a general method of evolving controllers, for a variety of different robots. The main findings of Reeve's work were that a simple encoding (allowing fully connected networks), high order neurons and simple fitness functions (specifying only the behaviour required, and not making assumptions about the functionality of any part of the system) are best for evolving these CPGs. Additional work was carried out by Reeve to evolve simple higher level controllers for these CPGs, namely for controlling turning and slowing the speed of walking.

A different approach to constructing a CPG, although still using general ANN structures, was carried out by Prentice et al [Prentice 1995][Prentice 1998]. The

CPG was split into two ANNs for different functions: the first for creating a timing signal and the second for shaping the timing signal into muscular activity patterns.

The timing ANN was supplied with a single tonic input and was trained using Back-Propagation (which modified the connection weights and biases) to produce sine and cosine waveforms (with a period equal to the stepping period) on the two outputs. The network comprises mainly feedforward connections, although there were fixed weight recurrent connections from the two output neurons to two state neurons, and from the two state neurons to themselves. These were located in the input layer, although they received no external inputs. All neurons had an hyperbolic tangent activation function, except for the input and state neurons which were linear. The shaping ANN was simply a feedforward network, again trained using the Back-Propagation algorithm. Two inputs to the network were the outputs from the timing ANN. The outputs from the shaping network were eight activity patterns for different muscle groups. After training, the network as able to produce outputs which captured the basic features and were of the correct general magnitude and timing as the activation patterns, although there were noticeable errors in absolute magnitude.

3.5. Reflexes

3.5.1. Simple Reflexive Systems

Due to the unique nature of the ANS model investigated in this research, reflexes as defined here are not used in other systems. Reflex-like movements are used in mobile robotics for obstacle avoidance (since this is a major requirement in all mobile robots). Systems which include these movements can also be classed as reactive, as they react immediately to their environment (through a reflex movement). Functionally based on pain withdrawal reflexes, these simple systems will cause the robot to move away from the source of a collision (for a brief overview see [Arkin 1995]). Extensions to the operation of these reflexes include trying to prevent a collision from occurring in the first instance, such as in [Wikman 1996]. This is a different definition of reflex than that used in the ANS presented here. These are at a higher functional level (as pain withdrawal reflexes can become quite complex, e.g. the crossed-extensor reflex) and could be considered as simple actions.

Reflexes have also been used in the walking systems of robotic insects (usually hexapods). The purpose of the reflexes used are normally to move a leg between the anterior and posterior stepping positions. The reflexes in each leg are then linked in a manner to produce co-ordinated walking gaits. For a comparison of various reflex stepping mechanisms, see [Ferrell 1995]. Additional work has been carried out which integrates other reflexes to enhance the insect's walking abilities. Additional reflexes, such as elevating the legs over obstacles and finding secure footholds, help improve locomotion over realistic terrain [Espenchied 1996]. These types of reflexes would operate at a similar level to that of reflexes in the ANS in this research.

3.5.2. Reflexes Used for Learning

In the architecture described in [Millan 1996] for navigating a robot to a (sensed) goal position, there exists a basic set of low level reflexes for generating movement. The sensory situation is mapped to an action, which moves the robot in a certain direction. If the current situation cannot be generalised into a learned movement, the reflexes are used, both to move the robot and as a training example. Initially there is no knowledge in the system, and so the robot ends up simply following walls and avoiding obstacles. However, as the robot learns (using Reinforcement Learning) it will move directly towards the goal position. The robot is still capable of avoiding obstacles, does not get stuck in dead ends and can even find the goal if moved. However, there are a lot of factors that need to be manually tuned and the performance of the system could be worse in large or more complex environments.

The system of Scutt and Damper (described in [Scutt 1997] and [Damper 1998]) contains a hard-wired CPG to produce an alternating left-right motion to move the robot forwards. However, the main focus of these papers are the reflexes and the method in which they are used during learning. The reflexes were functionally equivalent to pain withdrawal reflexes in biology: a noxious stimulus is presented (in this case a signal is activated when the robot collides with an object) and the robot withdraws from it. Learning in the system is due to habituation and sensitisation, which cause the robot to react with decreased or enhanced magnitude to the same stimulus in the future. Linked with this is the learning that takes place due to conditioning. Synapses from the collision sensors modulate the activity of synapses carrying information from light sensors. As the robot operates in its environment, it learns to associate falling light levels with an imminent collision and therefore learns to avoid collisions by avoiding low levels of light.

A similar system for conditioning collisions and reflex avoidance has been presented in [Gaudiano 1996].

3.5.3. Reflex models

Just as with CPGs, there are reflexes which are modelled on those found in biology (and not only in a functionally similar manner, like the pain withdrawal reflexes mentioned above).

The work of van Heijst [van Heijst 1998] proposed enhancements to that of Bullock [Bullock 1993] for a model of spinal circuitry controlling muscle activation around a joint (the FLETE model). The aim of van Heijst is to find a general method for creating controllers for technology, which are based on the stretch reflex in biology. The major neuron groups required for controlling the muscles around a joint are modelled using leaky integrators, and the connection patterns between groups are pre-specified. The enhancement over the previous FLETE model is that the motorneurons (the neurons which activate the muscles) behave according to the size principle. The size principle states that smaller muscle fibres are activated at lower thresholds than large muscle fibres. All the extra control required (additional neuron groups) for this mode of operation is included in the model. The precise weights for the connections are trained using a modified Hebbian learning method, allowing the network to self-organise, having been presented with some initial training inputs. The results show that the model is capable of controlling, not only the joint angle (which is dependant on the difference in activation between the two muscles around the joint), but also the joint stiffness (which is controlled by the common activation of both muscles).

Another model of a reflex is used for robotic arm compliance (position and force) control, during constrained movements (such as is required on assembly lines). Described in [Wu 1997], it models the muscle, muscle spindle and reflex pathways found in biology. The parameters for each of these control blocks were calculated by fitting experimental biological data onto them. The muscles track the movements and the spindles measure the position and rate of change of position in the muscle. The only input to the system is the target position and the reflex system takes care of the low level movement. It was found that the movement of the robotic arm to the desired position was smooth and capable of recovering from disturbances. Force was also controlled using this model, without the need for expensive force sensors, due to the model of the spindle.

3.5.4. Reflexes in Control Applications

At the most fundamental level, reflexes are simply methods of controlling the muscles of biological systems; therefore, control systems can be seen as functional equivalents for technological systems.

Control systems can be implemented using a variety of ANN types including feedforward multilayer neural networks (the most commonly used), recurrent neural networks, CMAC networks and Kohonen networks. The three main ways ANNs are used for controlling a plant are forward and inverse modelling of the plant, followed by training a neural network controller, and predictive control [Hunt 1995]. Forward modelling involves learning the plant outputs for given inputs and inverse modelling is where the plant inputs are learned for the current outputs. This reduces the problem of training the neural network controller to a pattern recognition one, which ANNs are particularly suited [Rovithakis 2000]. Predictive control attempts to predict how the plant will react to the current control signals so as to minimise future errors.

Using an ANN as a reflex to control the speed of a D.C. motor has been presented in [Jarandanan 1998]. Inputs to the network are the error in the current speed and change in error of the current speed of the motor. The output is the control signal for achieving the desired speed response. Fuzzy logic methods are used to create the training examples, which are taught to the ANN using the Back-Propagation algorithm (with adaption and momentum). The ANN is capable of controlling the motor, even with variations in torque and motor parameters.

3.6. Context of Current Research

This section discusses differences between the research presented in this thesis and the different architectures and systems mentioned in previous sections. The context and originality of the research presented is also demonstrated.

SA and CAMB architectures are both quite similar to each other in that they are behavioural systems. The sole purpose of these systems is to control a robot at the behavioural level. The ANS presented here differs in that it is a framework for an entire nervous system; the behavioural controls make up only a part of it. The behaviours in the ANS are selected by the higher level modules, rather than more specific behaviours as with SA. Unlike SA and CAMB, the behaviours in the ANS framework do not control the robot fully; they rely on sending signals to the lower layers, which then take care of the specifics of controlling the robot.

The GenNets, ANS and CAM-Brain of de Garis are different from the ANS in this research for the same reason as above. The modules of the ANS presented here make use of services from lower layers to control the robot, whereas with de Garis' work there is no explicit hierarchy.

The three layer architecture is a general class of architecture and can cover the ANS model used in this research (it follows the principle of increasing precision decreasing intelligence) as it is a layered architecture. However, the ANS model presented here does not have a unidirectional flow of information through the architecture, as TLAs most commonly do.

Digney's hierarchical control architecture is created using a variant of reinforcement learning (Q-Learning) but in the work presented in this thesis, EAs are the focus for creating the ANS components. The other difference is that Digney uses control sequences which are either primitive actions or other control sequences; the work described here studies the use of ANNs to implement parts of the ANS.

Dellaert and Beer's model of development is different to that used for the ANS model presented here. In their work, the entire nervous system is evolved using cellular mechanisms, with no structure imposed by the designer. The ANS model presented here has obviously been structured manually, and no cellular evolution has been used in this work.

Jacob’s ANS model does not have the same formal hierarchical structure to it that the ANS model presented here does. Jacob’s work also describes the networks as being used to solve tasks whereas the modules in this work are used to implement functionality rather than solve tasks.

The modelled CPGs and most of the neuron models that are typified in the simulated lamprey research are more complex than the CPGs which are envisaged to be used in this ANS. The lamprey work involves the use of parameters measured from real animals, which is something that cannot be included in the ANS as it is a general framework.

Benbrahim's CPGs, created from CMAC neural networks, are implemented differently from the way which CPGs are implemented here. Firstly, the CMAC CPGs are self-contained systems, capable of controlling the robot directly, whereas the CPGs in this research are part of a hierarchy and so can be simpler. Secondly, Benbrahim’s CPGs use inputs such as the cycle period and current time within the cycle. The CPGs created here use less specific information than that (e.g. a simple activation signal). Finally, Benbrahim uses Reinforcement Learning to train the CPGs, but a specific part of this research examines the evolutionary development of parts of the ANS.

CPGs in the ANS model used in this thesis differ from the CPGs presented in the majority of the legged locomotion work (Taga, Miyakoshi, Kimura, Akiyama, Berkemeier, Ventakamaran, Golubitsky and Stewart). The units used in those CPGs oscillate automatically whereas the focus here is on neurons (which must be coupled in a network to produce any oscillations).

CPGs which do use neurons as the processing unit (Lewis, Cao and Reeve) were all applied to walking robots. The main difference between those CPGs and the work presented here is that the CPGs directly control the robot; they are not part of a hierarchical system to the same extent as the CPGs are here. In the case of the CPGs which modelled human EMG patterns (Prentice et. al.), those networks were trained using known input and output signals rather than evolved.

The typical use of reflexes in mobile robots, as a simple reactive behaviour, is incorporated in the reflex layer of the ANS presented here. However, there are more types of reflex than the pain withdrawal, which is commonplace in those systems. They also only make up a small proportion of the whole system in this case, rather than a major component.

Learning behaviours based on the reflexes in the system, such as in the systems of Millan, Scutt and Damper, and Gaudiano are a task occurring at the behavioural level in this system. However, this type of learning would not be used exclusively in this system. Instead a combination of evolutionary and learning methods is used to acquire the different behaviours.

While the reflexes used here are based on biological reflexes, they do not necessarily need to be modelled as exactly as they are by van Heijst. The method of training the reflex is also different from that used by van Heijst, as an evolutionary approach is used here. Similarly, the control system used by Wu is more complex than is required for this ANS, as it requires complex data fitting of recorded biological data to computed system parameters. The reflexes in this ANS require no prior knowledge of any specific biological data.

The traditional methods for using ANNs in control systems differ from the way reflexes are used for control in the ANS model in the method they are trained. Typical control ANNs are used to control unknown plants, and so require the modelling of that plant (using another ANN) to be able to generate training sets. The reflexes are evolved without any use of modelling the actuators or specific training patterns.

Using an ANN to act as a reflex and control the low level components of the robot, as done by Jarandanan, is something which is included in the ANS. However, the method of training the ANN is different, as an evolutionary approach is investigated here. The structure of the network is also different, as to fit in with the layers above the reflex, there also needs to be additional inputs for the desired controlled value, and not simply the errors.

3.7. Summary

This chapter has reviewed the important and related work, which has previously been carried out in the areas of robot control systems, central pattern generators and reflexes. The research in this thesis has been put into context with the existing literature, and the originality of this work emphasised.

The next chapter will provide detailed explanation of the lowest level in the ANS model (the reflex system), with reference to literature on biology to support the ideas.

Chapter 4. Artificial Reflex Theory

4.1. Introduction

The previous chapter discussed other, published research in the areas of robot control systems, central pattern generators and reflexes. The research presented in this and the following chapters was then put into context in the summary.

Initially, in this chapter, biological mechanisms for movement are introduced. The simplest forms of controlling movements, reflexes, are then described. These will form the basis for the ideas behind the artificial reflexes, which are then presented and justified. The implementation details of the artificial reflexes are documented, followed by a brief overview of the areas which will be investigated in the next chapter. This chapter then closes with a summary of the information presented here.

4.2. Biological Movement

Most active biological movement is controlled by contractile elements, whether they are vacuoles for pumping liquid (as in amoebae) or arrangements of contractible cells (such as the walls of lower invertebrate, or muscles in more complex creatures). For this reason, and the fact that muscles are present in both vertebrates and many invertebrates, muscles will be the main focus here (as opposed to layers or walls of contractile cells).

4.2.1. Muscle Activation

Muscles are activated directly by groups of dedicated neurons, known as motor neurons. All the other neural structures which cause movement connect to the motor neurons rather than the muscle itself. This tends to support the idea that the nervous system is a layered structure, with higher layers using functions provided by the lower layers. The motor neuron connections terminate in boutons on the surface of the muscle fibres, as shown in Figure 4.1.

As neurons produce spiking outputs, the muscle produces a twitch of contraction for each spike, and then relaxes back to the uncontracted state [Ganong 1995a]. The muscle has an integrative effect, so a train of spikes sent to it will cause it to stay contracted at a length relative to the average firing frequency of the neuron

[Ganong 1995b].

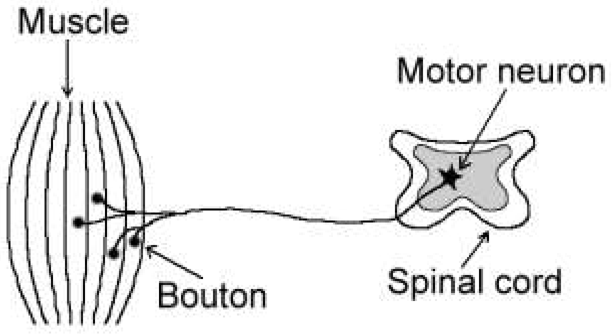


Figure 4.1: Connections between motor neuron and muscle

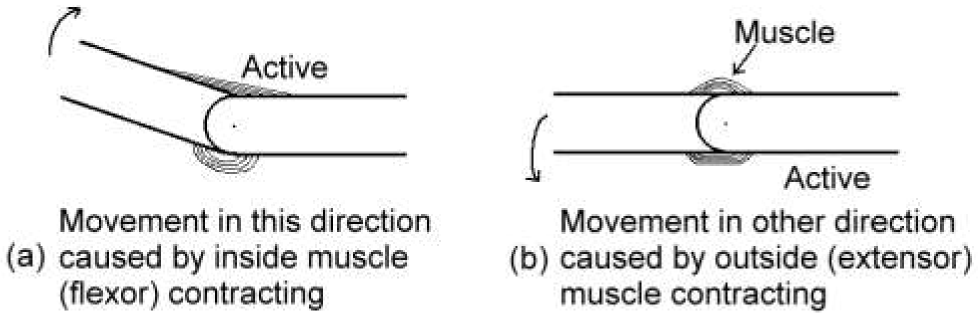
Muscles of different sizes are controlled by different numbers of motor neurons, making up a motor neuron pool, the size of the pool relating directly to the size of the muscle. Motor neurons in the pool have a threshold which is directly proportional to the size of the muscle fibre that they innervate. Small fibres are activated first, with the larger and stronger fibres only contracting if the level of stimulation on the motor neuron pool is high enough (known as the size principle) [Kandel 1991a]. This allows fast, fine movements to take place, although slower and stronger movements can be made if the activation is high enough. The activation of the muscle is therefore proportional to the level of stimulus applied to it.

4.2.2. Controllable movement

Since muscle only allows control of movement in one direction (only the length can be controlled by contraction; the relaxation is dependant on the muscle fibre properties), more complex arrangements of muscles must be used to accomplish more useful movements.

Most joints on a body are revolute, requiring at least two muscles for control [Kandel 1991b]. The muscles connect on opposite sides around the joint. This is shown in Figure 4.2.

Each muscle controls the angle of the joint in one direction; both muscles together control the tension of the joint. All nervous systems allow control at the muscular level; some advanced nervous systems also allow for joint control directly. In that case, another layer of interneurons activates the motor neurons while the higher levels need only provide inputs for the joint angle.


Active
Muscle
Active
Movement in this direction
(a) caused by inside muscle
(flexor) contracting
Movement in other direction
(b) caused by outside (extensor)
muscle contracting
Figure 4.2: Control of a rotating joint using opposing muscles

4.3. Biological Reflexes

The typical structure of a reflex is given by the *reflex arc*. This has the general form of a sensor on the body connecting to at least one interneuron, which in turn connects to one or more motor neurons. In a simple invertebrate, such as Hydra, there are no interneurons, since the cells on the body act as both sensor and actuator stimulus [Buchsbaum 1968]. An example of a structure like this is shown in Figure 4.3.

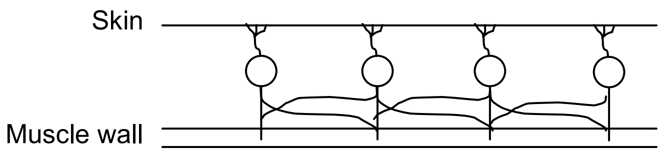


Figure 4.3: Example of reflexes found in lower invertebrate

More advanced invertebrates, such as the leech and other types of worm, have dedicated sensors on the body and localised interneurons which stimulate the muscles. Vertebrates have sensors on the body which connect to interneurons in the spinal cord (afferent connections) [Haines 1997a]. These stimulate the motor neurons which activate the muscles via efferent connections. The reflex arc for a vertebrate is shown in Figure 4.4.

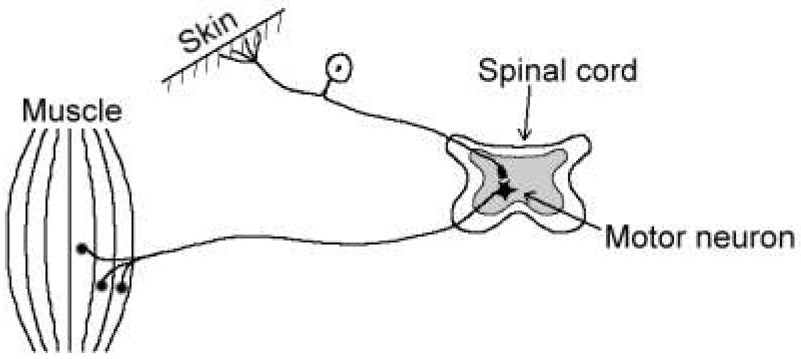


Figure 4.4: Generalised reflex arc of a vertebrate

The behaviour of reflexes can be modulated by higher levels of the nervous system [Brodal 1992], although this may only be a limited influence and could depend on the strength of the stimulus causing the reflex action. The behaviour and connectivity of the most common reflexes are described in the following sections.

In biological systems the term reflex gets applied to many different types of neural control systems and movements. It ranges from simple, single muscle loops as seen in the stretch reflex (explained in section 4.3.1) to more complex actions like the crossed extensor reflex (explained in section 4.3.3) and beyond. The one common factor in reflexes are that they operate automatically without intervention of the higher processing functions of the brain. It is possible to classify reflexes in two groups: controllable and automatic. The operation of the controllable reflexes is governed by descending controls from the higher centres of the nervous system but they remain automatic in the performance of their task. Automatic reflexes are those where the originating signal for reflex operation comes from the sensory systems and can be routed to the muscles without any intervention from the higher centres. However, it is possible to modulate the operation of the reflex to some degree, for example attempting to delay the removal of a body part from a noxious stimulus.

4.3.1. Stretch

The stretch reflex is only found in animals with muscles (as opposed to cellular contracting bodies). The purpose of this reflex is to regulate muscle length, which makes it a useful reflex for purposeful control of movement under the influence of higher centres of the nervous system. The nervous system connections and muscular structures required for this reflex are shown in Figure 4.5.

Anchored inside the muscles are two types of fibres, called spindles. The activity of nuclear chain spindles increases with muscle length, while nuclear bag spindles produce activity proportional to the rate of change of length of the muscle. The spindles also have contractile sections at the ends (therefore the centre of the spindles will stretch when the ends are contracted), the use of which will be explained later. Located at the end of the muscles are Golgi Tendon Organs, the activity of which increases with muscle tension [Kandel 1991c].

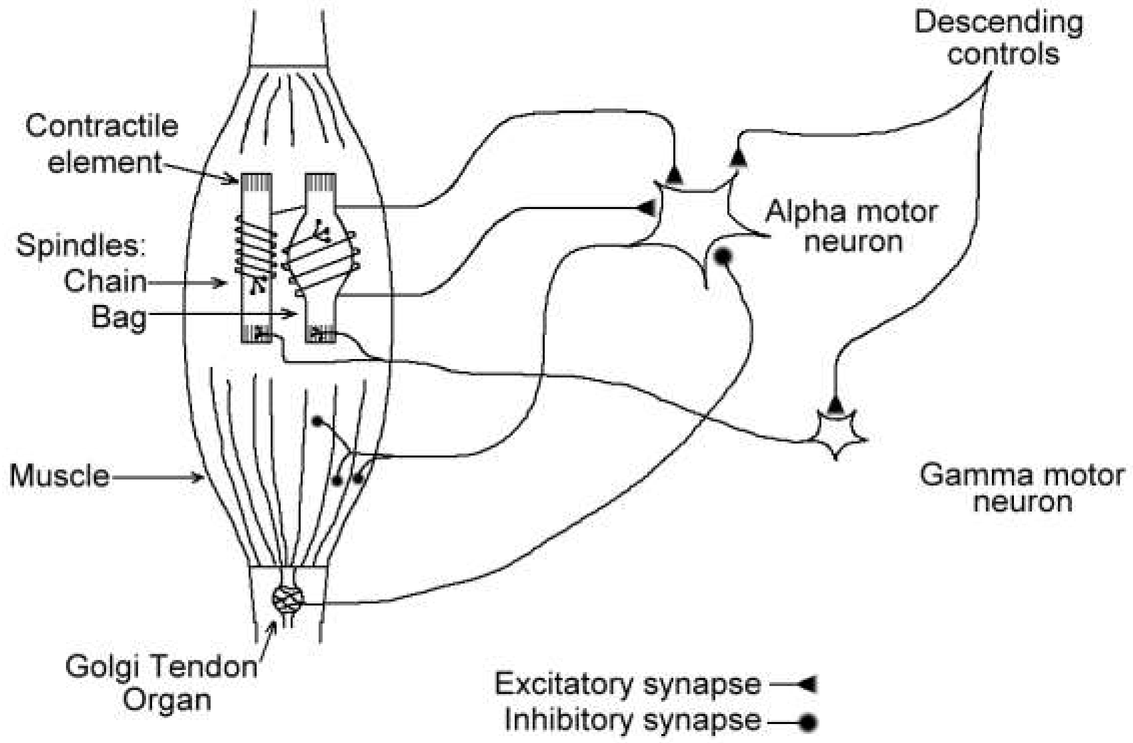


Figure 4.5: Nervous system connections and muscular structures for stretch reflex

The spindles eventually end in excitatory synapses on the alpha motor neurons of the muscle they are connected to, while the Golgi Tendon Organs have an inhibitory effect. The alpha motor neurons innervate on the muscle fibres while the gamma motor neurons activate the contractile parts of the muscle spindles. This system causes the reflex to operate in three different ways [Haines 1997b]:

1. Extension of the muscle causes increased activity in the spindles. In turn, these excite the motor neurons, causing the muscle to be contracted. This reaches an equilibrium point where the muscles no longer extend, due to highly active spindles. Both the nuclear chain and bag spindles prevent

excessive lengthening of the muscle, while the nuclear bag spindles also prevent muscles being extended too quickly.

2. Contraction of the muscle increases the activity of the Golgi Tendon Organ. This also has a balancing effect once the muscle has been contracted to a high degree, as the tendon organ inhibits the motor neuron, thus preventing it from contracting the muscle any further.
3. Active contraction of the muscle can be initiated by activating either the alpha motor neurons or the gamma motor neurons (contracting the muscle or stretching the spindles respectively). Stretching the spindles causes their output to increase, and the effect will also be like that in the first point; an excitation of alpha motor neurons. In systems where alpha and gamma motor neurons are present, both are activated simultaneously so as to contract the muscle and keep the spindle taut. This keeps the spindle in its most dynamic range of activity, which gives better control over the muscle.

This reflex can also be extended to control the angle of a joint, similar to the method mentioned in Section 4.2.2 [Haines 1997a]. The neural circuitry makes the reverse connections to the muscle on the other side of the joint; spindles make inhibitory connections and Golgi Tendon Organs excite the antagonist motor neurons. This would then provide active control over the joint tension and angle, while not allowing the muscles to move outside of their safe operational limits.

4.3.2. Withdrawal

The purpose of the withdrawal reflex is to remove part of the body from a noxious stimulus (most often pain) [Haines 1997a]. The sensory cells on the body stimulate interneurons which co-ordinate the activity of the motor neurons so that the body part is removed from the stimulus. These cells could be used for some other purpose simultaneously, such as touch. However, when the strength of the tactile contact becomes greater than some threshold, the withdrawal reflex is activated. The muscles used in this reflex are in close proximity to the point of stimulus on the body. For example, this reflex might cause the arm and hand to retract when a finger receives pain.

4.3.3. Crossed extensor

This reflex has the same purpose as the withdrawal reflex, although it uses more complex neuronal circuitry, to allow more advanced withdrawal sequences. The extra involvement comes from the need to activate muscles (typically) on the opposite side of the body, in an effort to remove the injured body part whilst keeping balance with the opposite side of the body [Haines 1997a]. One example of this type of reflex would be when an animal stands on something which causes pain. Not only must the leg of the foot which is receiving pain be lifted, but the opposite leg must be moved in order to keep the body balanced, which is now supported by one less foot. This is demonstrated in Figure 4.6.

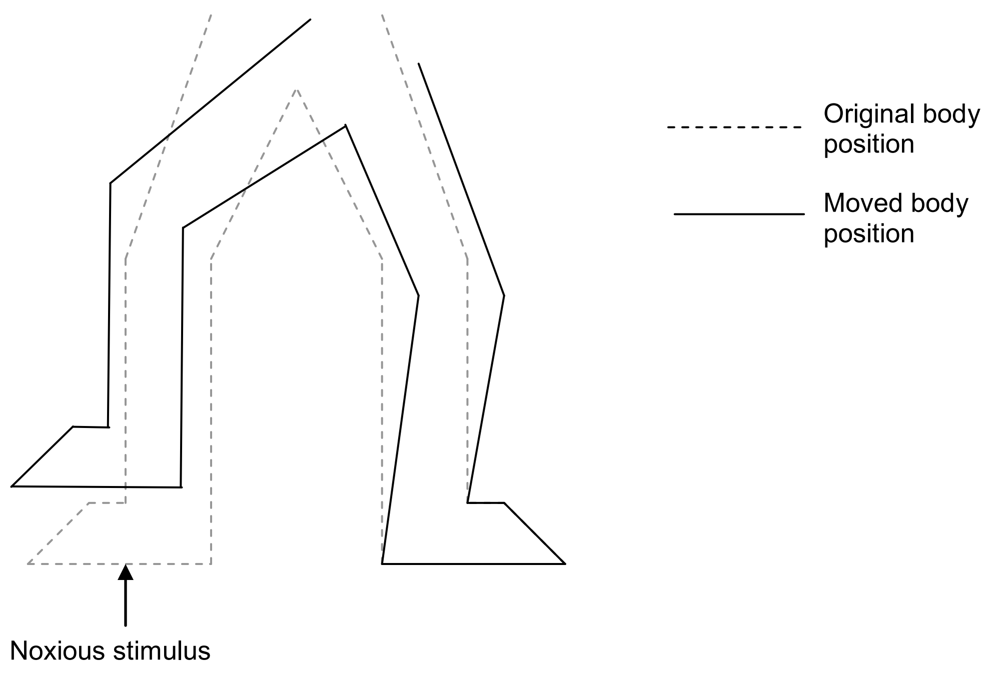


Figure 4.6: Example of body movement during a crossed extensor reflex

4.3.4. Other reflexes

There are more complicated reflexes than those previously mentioned, which involve more complex sensors and more muscle groups across the entire body. An example of this would be the startle reflex. Hearing a sudden, loud noise in close proximity, or seeing something moving fast towards it, a creature would “jump”.

Another action that might be thought of as a reflex, is one which triggers a specific action or series of movements. An example of this would be the initiation of a dog scratching due to a flea bite. However, this is not strictly a reflex; technically, it would be an innate behaviour.

4.4. Implementing artificial reflexes

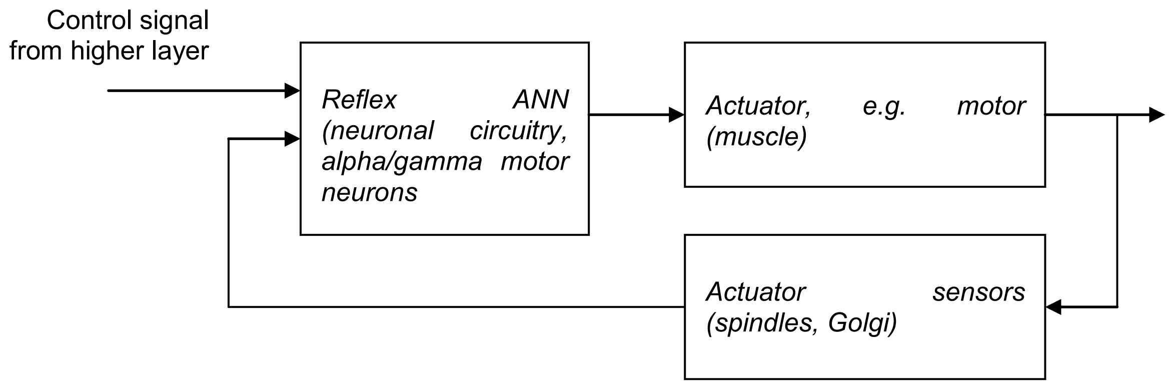
4.4.1. Type of reflex

Investigations initially concentrated on implementing the artificial equivalent of the biological stretch reflex. There are two main reasons for this:

1. It provides the useful ability of higher level control over the actuators of the system (as it would be classified as a controllable reflex). Other reflexes such the withdrawal reflex are not directly controlled by commands from the upper layers of the nervous system. Instead, they are initiated by some external stimulus; descending signals only serve to modulate these reflexes.
2. It is a simple reflex, as only one muscle and the associated neural circuitry is involved.

This will allow a simple reflex to be created in isolation, yet it will be capable of fitting into the ANS to provide an interface for the higher layers to control the hardware of the robot easily.

Figure 4.7 shows a block diagram of the functionality of the artificial reflex. The biologically equivalent parts are marked on the diagram.



Control signal from higher layer
Reflex ANN (neuronal circuitry, alpha/gamma motor neurons
Actuator, e.g. motor (muscle)
Actuator sensors (spindles, Golgi)

Figure 4.7: Functional block diagram of artificial stretch reflex, with biological equivalent parts marked

The reflexes such as the withdrawal and crossed extensor reflex can be added at a later stage. Due to their autonomous nature (in that some external stimulus initiates these reflexes but the higher levels do not), adding these reflexes will not require changes to layers immediately above that of the reflexes. However, before adding these to the system, it would be appropriate to have at least one controllable reflex. Although the withdrawal could operate as expected without any controllable reflexes, it would be more informative to observe the reactions of the system when a body part is first moved in a way that causes the withdrawal reflex to activate, rather than simply applying the noxious stimulus directly. Additionally, since all muscle movement in biological systems is controlled through the motor neurons, the withdrawal reflex uses the stretch reflex without having any knowledge of it. The system diagram for the connectivity between the stretch reflex and other reflexes is shown in Figure 4.8.

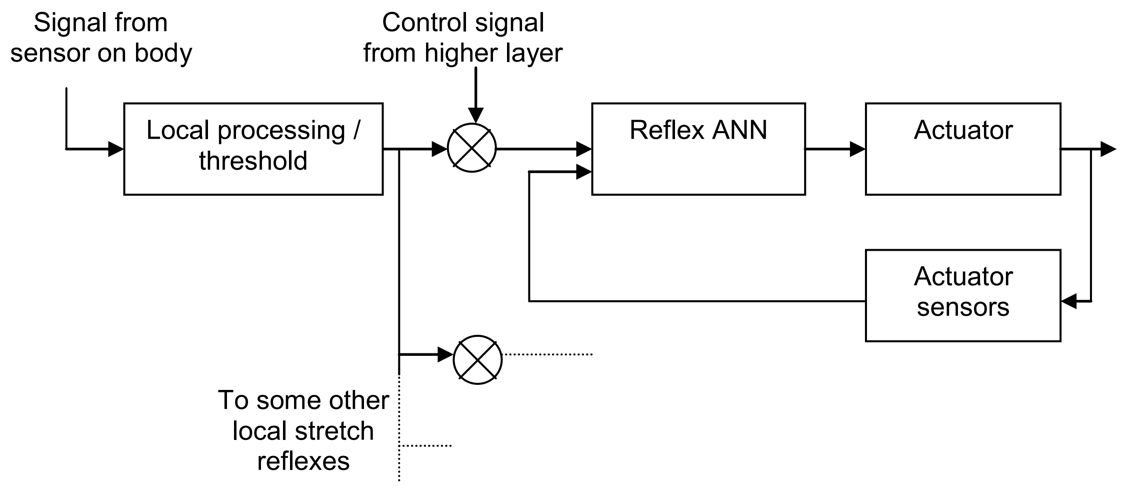

Signal from sensor on body
Control signal from higher layer
Local processing / threshold
Reflex ANN
Actuator
Actuator sensors
To some other local stretch reflexes

Figure 4.8: Additional reflexes interfacing with stretch reflex

The output from the sensor local processing would need to act as an additional input to the stretch (and therefore artificial) reflex, since only the motor unit directly controls the muscle.

More complex reflexes (such as the startle) require advanced sensors and the high level processing that goes with them. For this reason, and the fact that it would require parallel connections directly to the stretch reflexes (instead of working the signals through the hierarchy of the ANS) as shown in Figure 4.9, the addition of this type of reflex would be a target for future investigation.

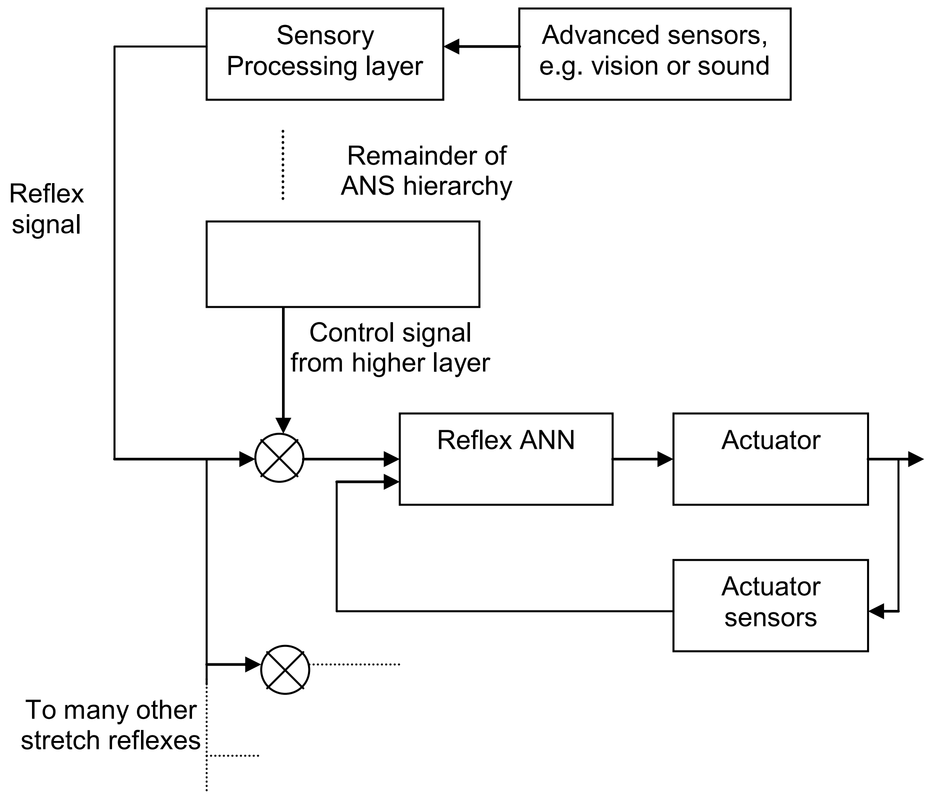

Sensory Processing layer
Advanced sensors, e.g. vision or sound
Remainder of ANS hierarchy
Reflex signal
Control signal from higher layer
Reflex ANN
Actuator
Actuator sensors
To many other stretch reflexes

Figure 4.9: Complex reflexes interfacing to stretch reflex

4.4.2. Hardware to be used for movement

The artificial reflexes will be created using a simulation of the actuator it must control. This has two advantages:

1. Simulation can be performed quicker than running in real time
2. Real systems can be worn out due to excessive repeated use, which will be the case when training or evolving ANNs

However, simulation also comes with the disadvantage that it will never be as exact, or have all the subtle variations, that using a real actuator could have. Since neural networks are being used, their generalisation ability can be taken advantage of so that the transfer from simulation to real actuator will not require completely different solutions. Therefore simulation will be used when performing the experiments and constructing the ANNs.

A simple linear approximation for a single voltage supply D.C. motor will be used initially. This was the most basic simulation of an actuator, but allowed the reflex to be evolved quickly and with a good representation of the behaviour of a real actuator. The state equations for this simulation are:

$$\frac{d\theta}{dt} = \dot{\theta}$$
$$\dot{\theta} = V \qquad \text{Equation 4.1}$$

Where θ and $\dot{\theta}$ are the position and speed of the actuator respectively, and V is the input voltage.

A more realistic simulation of a D.C. motor will be used later. The state space equation for this is given in Equation 4.2.

$$\frac{d}{dt}\begin{bmatrix}\theta \\ \dot{\theta} \\ i\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 \\ 0 & -\frac{b}{J} & \frac{K}{J} \\ 0 & -\frac{K}{L} & -\frac{R}{L}\end{bmatrix}\begin{bmatrix}\theta \\ \dot{\theta} \\ i\end{bmatrix} + \begin{bmatrix}0 \\ 0 \\ \frac{1}{L}\end{bmatrix}V \qquad \text{Equation 4.2}$$

Where θ and $\dot{\theta}$ are the position and speed of the actuator respectively and V is the input voltage. The variables b, i, J, K, L and R are measurable parameters of a real motor and are described in Table 4.1.

Table 4.1: Motor parameters used in simulation of actuator

Parameter	Description	SI Units
b	Damping ratio of mechanical system	Nms
i	Armature current	A
J	Rotor moment of inertia	kgm^2s^{-2}
K	Electromotive force constant	NmA^{-1}
L	Armature inductance	H
R	Armature resistance	Ω

The limits of the actuator are normalised to the range 0…1. This includes such measures as position, speed and acceleration. Normalised values have two advantages over actuator specific values. The first is that normalising all actuators to the same scheme allows one reflex to control different actuators (taking advantage of the generalisation abilities of ANNs). The second is that the values are in a range which allows an ANN to deal with and interface to them more easily.

When the speed is normalised in this scheme, 0 means full speed in one direction and 1 means full speed in the opposite direction (in the case of these simulations, backwards and forwards respectively).

The range 0…1 also represents the range that the actuator can be positioned at, with 0 being the minimum possible position and 1 as the maximum position, in the opposite direction. If the actuator has no positional limits (such as with a motor), then 1 will represent a well defined position on the actuator, and on passing this, the position will continue to increase. For example, if the range 0…1 mapped to a single revolution of a D.C. motor, then moving past position 1 would be passing one revolution. A position of 1.5 would be one and a half revolutions. This also applies to positions less than zero, for example a position of –0.5 would be half a revolution backwards.

The actuator control signal from the reflex will also need to be scaled to cover the range of possible input values that the actuator can accept.

4.4.3. Neural Network Implementation

As required by the project objectives set out in section 1.6, a basic single reflex system will be implemented. To achieve this, ANNs will be developed which perform the same task as the muscle stretch reflex.

Different connection strategies will be employed in the ANNs. Simple feedforward only networks will be tested as the simplest type of network available. Due to their simplicity and their operation being a mapping function from a set of inputs to a set of outputs, these networks are likely to be the largest. Networks which have the possibility of full recurrent connections will also be investigated. By allowing this type of connection, the network will not need to work as a simple mapper, and can be smaller. Extra information can be generated within the network, with the recurrent connections acting like time delays; previous values in the network can be fed back to neurons on the next iteration of the network. This will also allow the network to have integral and differential abilities (to a small degree) which will be useful in this application, since the reflex is performing a similar task to that of a PID (proportional + integral + differential, or other variation) controller. The last method for connecting neurons which will be studied is a compromise between the two strategies already mentioned - no backwards or sideways connections will be permitted within the network, but connections can branch forward to any layer (not only to the successive layer). This medium ground might prove a useful alternative as the networks would not be as complex as fully recurrent ones, yet would allow for better performance over the standard feedforward networks.

Inputs to the network will be fixed during evolution, although different combinations of inputs will be investigated, to find out which are required for good control over the actuators. The possible inputs can include any of these:

- Desired position (control signal from higher layer)
- Desired speed (control signal)
- Current position of actuator (feedback from actuator)
- Position of actuator integrated (feedback)
- Current speed of actuator (feedback)
- Current acceleration of actuator (feedback)
- Error between desired and current values (combination of feedback and control signal)

These are suitable for covering the possible inputs for both biological reflexes and traditional controllers.

The output from the ANN is always the normalised control signal for the actuator. The evolution of the reflex ANN can be shown as in Figure 4.10.

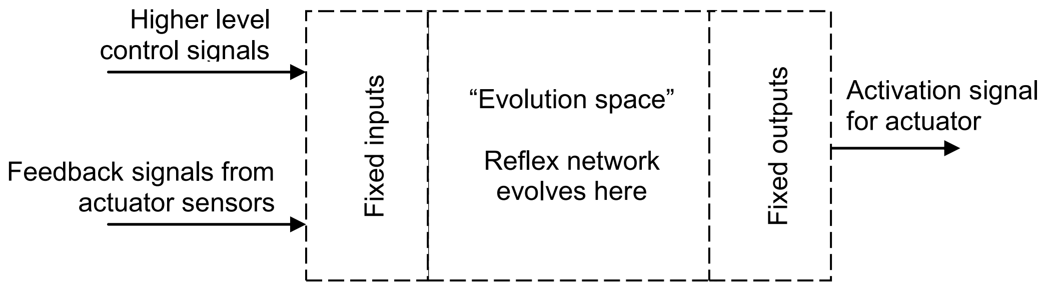

Higher level control signals
Feedback signals from actuator sensors
Fixed inputs
"Evolution space"
Reflex network evolves here
Fixed outputs
Activation signal for actuator

Figure 4.10: Structure of ANN reflex

The type of neuron used in the reflex will be limited to a McCulloch-Pitts neuron with a sigmoid transfer function. This is to keep the reflex ANN as simple as possible, and the McCulloch-Pitts neuron is easily capable of performing this task.

The input nodes are fixed during evolution, as are the number of neurons in the hidden layer(s) of the network (although both of these parameters can be changed between evolutionary runs). All networks evolved as a reflex will have a single output neuron. This is due to the fact that it is controlling a single actuator, which usually require only a single input. Reflexes will be evolved within the limits shown in Figure 4.10.

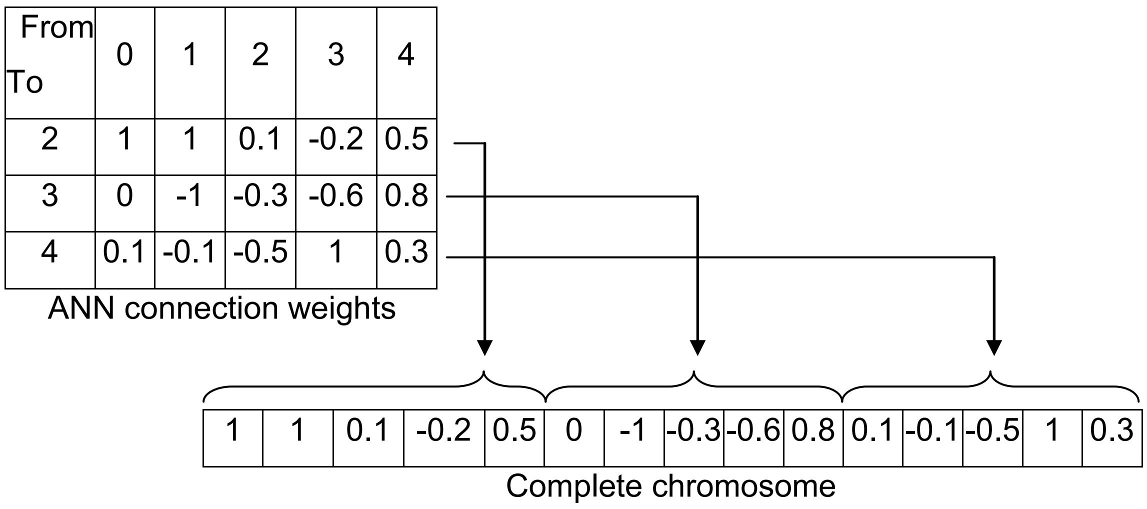
4.4.4. Training method choices for NN (GA)

Since this research is investigating evolutionary methods for creating an ANS, the ANN reflexes will be evolved. The three main evolutionary systems (GA, EP and ES) will be tested, as described in section 4.5.1.

The only property of the ANN which must be encoded into the chromosome is the weights of the connections between neurons. Due to the simplicity of the topology (fully connected), and the estimated small size of the network, the chromosomes encoding the networks can be directly encoded. Each gene in the chromosome will therefore be a single real value which represents a weight in the ANN. Negative and positive values represent inhibitory and excitatory connections respectively, while a value of zero means no connection.

Although a GA traditionally uses bit strings in the chromosome, there are problems with this when representing real valued numbers [Baeck 1996]. To encode real valued numbers, the bit strings are split into equal sized segments to represent an integer, which then maps linearly onto a range of real valued numbers. Therefore, in all the EAs each gene will be a real valued number representing the weight. This gives each chromosome the structure shown in Figure 4.11. Since the possibility exists for recurrent connections, any neuron can connect to any other neuron. However, no recurrent connections are made to the input nodes. The genes are grouped together so that all the connections from a given neuron (and going to all the other neurons) are arranged linearly in the chromosome. The chromosomes are initialised by setting each gene to a uniformly distributed random value between ±1.

The fitness for each gene was measured using the simulation described in section 4.4.2.



From / To	0	1	2	3	4
2	1	1	0.1	-0.2	0.5
3	0	-1	-0.3	-0.6	0.8
4	0.1	-0.1	-0.5	1	0.3

ANN connection weights

1	1	0.1	-0.2	0.5	0	-1	-0.3	-0.6	0.8	0.1	-0.1	-0.5	1	0.3

Complete chromosome

Figure 4.11: Structure of chromosome used in EA for an example network with 2 inputs, 2 hidden neurons and 1 output. Since there are no recurrent connections to the inputs, the first connections must go to neuron 2.

Each chromosome is given a fitness score based on how well it can move an actuator to a range of positions. The actuator must be moved from a start position (of zero) to a target position and then back to the start position. The target position is incremented from zero to one in a number of steps, and at each step the actuator must be moved from zero to the current target step and then back to zero. For example, if there were five steps (as shown in Figure 4.12), the fitness for a single chromosome would be the average of the fitnesses from the tasks:

1. Move actuator from 0 to 0.2 and back to 0
2. Move actuator from 0 to 0.4 and back to 0
3. Move actuator from 0 to 0.6 and back to 0
4. Move actuator from 0 to 0.8 and back to 0
5. Move actuator from 0 to 1.0 and back to 0

During each movement between the start and target positions, the ANN decoded from the chromosome is given a fixed period of time to perform the task. This fixed time is broken down into sampling periods and at each sampling interval, the inputs are passed forward through the ANN and then the actuator is updated using the output from the reflex as the control signal.

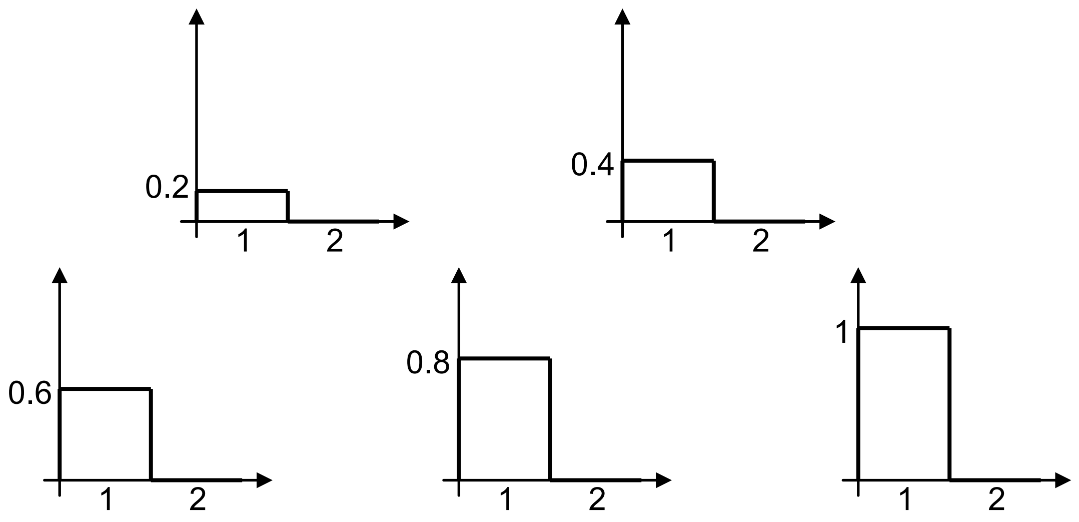


Figure 4.12: Stages in fitness evaluation. The target position to reach is shown in each diagram.

The fitness is calculated as the sum of three values which are commonly used to specify traditional control systems. These are offset, overshoot and rise time, which are combined as shown in Equation 4.3 to give the fitness score for that movement phase.

$$fitness = \left(\frac{offset}{target} + \frac{overshoot}{target} + \frac{rise\_time}{total\_time} \right) \div 2 \qquad \text{Equation 4.3}$$

Both the offset and overshoot are normalised to the current target position. The rise time is normalised to the fixed period of time. The measurements for the parameters of offset, overshoot and rise time are shown in Figure 4.13, which contains an example position of an actuator as it is moved. The fitness in each movement phase (labelled 1 and 2 in Figure 4.12) is combined so (which is the reason that each component in Equation 4.3 is divided by two).

The offset is any difference between the final steady-state position of the actuator and the target position (in Figure 4.13 the target position is 1). The overshoot is shown by the m_1 measurement, which is the difference between the target and the first peak. If there is no overshoot peak, then the overshoot fitness component is set to 0. Finally the rise time can be measured in a number of ways, and is shown by t_r on the diagram. Possible measurements for it are the time difference between the actuator moving from 10% to 90% of the target position or the time between

0% and 63% of the target position, or simply the time it takes from the start to reach the target position. In the fitness scores here, it will be taken as the time taken to reach the target position from the start.

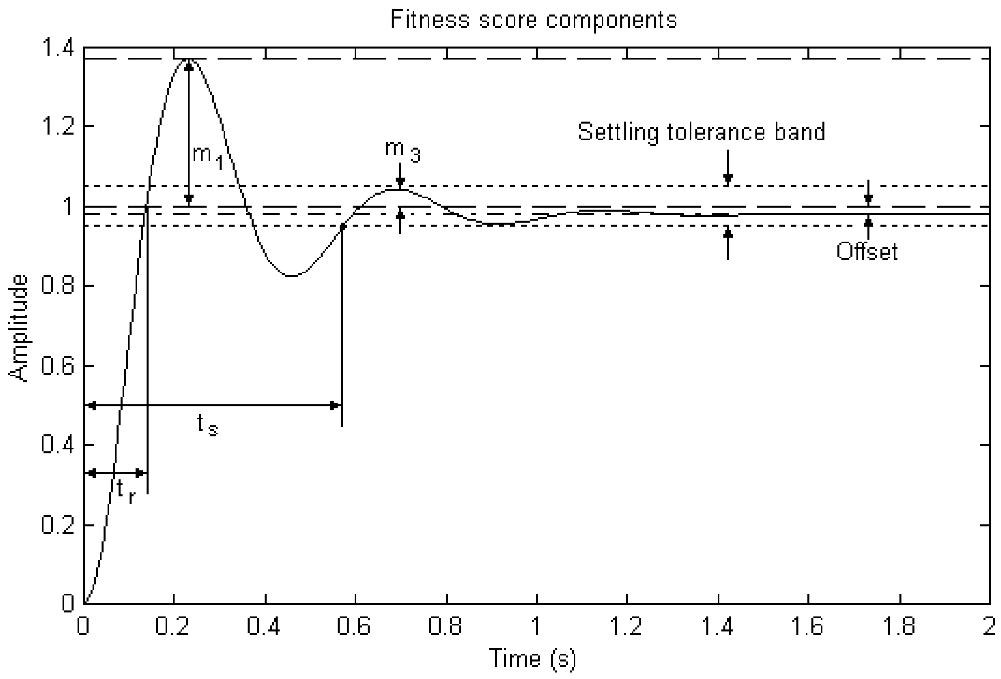


Figure 4.13: Example actuator position showing fitness component measurements

Obviously the lower the fitness score, the better the solution.

The termination criteria was decided as either the best fitness in the population reaching a specified threshold, or a maximum number of generations having passed. The maximum number of generations is used to stop evolution when there is no significant progress in the evolution, and continuing would not be worthwhile.

4.5. Experiments

4.5.1. Evolutionary Method

There are three main types of evolution, as discussed in Chapter 2. The first problem is to discover which one of these performs the best when used to evolve an ANN as a reflex. This can be found from how quickly the evolutionary method in question converges to a solution, and how good that solution is.

Normally, a GA would use bit strings for the chromosome. However, in these tests, the genes will be real valued numbers. Whilst this is not a canonical GA as in the original specification [Holland 1975], the other properties of this evolutionary method will be used. This will make it easier to compare the results of the three methods, although it may mean that the performance of the GA used here would not be exactly the same as if bit string chromosomes were used.

4.5.2. Evolutionary Parameters

Linked with the experiments in section 4.5.1, the evolutionary parameters for each method must be investigated to allow optimal or near-optimal performance. Only then can a fair comparison be made between methods. This also allows a comparison between certain parameters and the performance of the reflex.

The parameters which will be under investigation are:

- Population and elite sizes
- Mutation and recombination rates
- Selection method
- Number of steps in target positions

4.5.3. Fitness function variations

Since the fitness function is a vital part of the correct operation of an EA, the effects of changing parts of the fitness will be presented. This will range from changing how each fitness score component is measured to altering the basic fitness function calculation (as given in Equation 4.3).

4.5.4. Network Topology

The network topology is an important factor in the performance of the ANN. Various network sizes will be tested, by varying the number of hidden layers in the network and the numbers of neurons in each layer. The different types of connection strategies will be investigated at this point, and they are linked to the sizes of the networks.

4.5.5. Actuator Type

The reflexes will be tested against the two different types of actuator as described in section 4.4.2. Separate networks will be evolved for the different actuators and then their generalising ability will be tested by allowing reflexes to control actuators which they were not evolved on.

4.5.6. Feedback

Variations in the feedback taken from the actuator, which are presented to the ANN inputs will be studied. Different combinations of the possible feedback measurements (such as position or speed) will be tested, in order to find out what the minimum feedback for good operation is, as well as the optimum. The different configurations can be compared (in function) to biological stretch reflexes and also to traditional controllers which have different combinations of the proportional, integrative and derivative components.

4.6. Summary

This chapter initially described the biological aspects involved with reflexes, which are the lowest, most autonomous and simplest kinds of movement. Different reflexes have been described, and their links to the nervous system have been discussed, as well as their possible placement within the ANS.

The system for artificial reflexes to be implemented in the ANS has been established from ideas based on the biological reflexes. Comparisons have been drawn between both, culminating in a specific type of reflex being chosen to be implemented. The system used for testing the artificial reflexes has been described. The implementation details of how an ANN will be used as a reflex have been described, along with the evolutionary methods which will be used to create it.

Finally, the aspects of creating the ANN reflex which will be investigated have been described. The results of these investigations will be reported in Chapter 5, with conclusions being drawn upon these results.

Chapter 5. Artificial Reflex Creation

5.1. Introduction

The previous chapter described the experimental setup used to create the artificial reflexes. This included details of the experiments to be performed on EAs and their parameters, fitness functions, ANN topology and type of actuator.

This chapter describes the results of the experiments conducted in these areas. These are presented in section 5.2, with each experimental aspect having its own subsection. Section 5.2.1 deals with finding the most appropriate Evolutionary Algorithm (EA) for evolving an ANN reflex and 5.2.2 optimises the parameters for the selected EA. This class of EA is then used in all the other experiments from sections 5.2.4 to 5.2.6. A discussion of the results is presented in 5.3. Conclusions are drawn from the results and reported in 5.4. Finally, the chapter is summarised in section 5.5.

5.2. Results of Experimentation

5.2.1. Evolutionary Method

This section deals with finding the most suitable evolutionary algorithm (EA) for evolving a reflex ANN. The three main EAs (Genetic Algorithm - GA, Evolutionary Programming - EP and Evolutionary Strategies - ES) have been tested in their ability to evolve networks of this type. Only the basic operation of the EA will be considered at this point, using parameters based on heuristics gained from the literature. Further optimisation of these parameters will be considered in the next section, together with the specifics of how the genetic operators function.

The network topology was chosen heuristically, although at this stage (finding the most suitable EA) it will not be too significant. The topology chosen will affect all EAs in the same way, regardless of whether it is capable of controlling the actuator or not. Initially the network topology was set to three input nodes, one hidden layer with three neurons and a single output neuron. Each neuron was a standard McCulloch-Pitts neuron with a sigmoid activation function.

The feedback used in this first set of experiments was similar to the feedback present in biological reflexes. The current position and speed of the actuator are recorded at the actuator and used as inputs to the ANN. Since this is the type of feedback given by the muscle spindles in biology, it is a logical starting point. The results of modifying the feedback configuration are presented in section 5.2.6.

Results of experiments on how well the network can be trained using these simple (and more complex) simulations, and how well they transfer between different actuators are reported in section 5.2.5.

If the network was producing unstable control signals instead of reaching a steady state, the fitness function could be modified as reported in section 5.2.3.

The optimum fitness score depends on the difference between the initial and target positions (and therefore directly on the number of steps for the target position), and the maximum speed of the actuator. This optimal value is purely made up from the rise time, since the overshoot and offset can be reduced to zero. When the number of steps is one, the optimum fitness value will be 0.2.

For a distance of 1 revolution (initial position=0, target position=1) and maximum motor speed of 30 RPM (0.5 revolutions per second), the shortest possible rise time is 2 seconds ($1 \div 0.5$). Since the rise time is normalised to the time the ANN has control over the actuator, the rise time component of the fitness function is 0.2 (total control time=10s). As more steps are added (thus making the sets of initial and target positions closer together), the lower this optimum value becomes. The optimum fitness scores for different numbers of steps in the fitness calculations are shown in Figure 5.1. In the initial set of tests on the most appropriate EA, the number of steps was one.

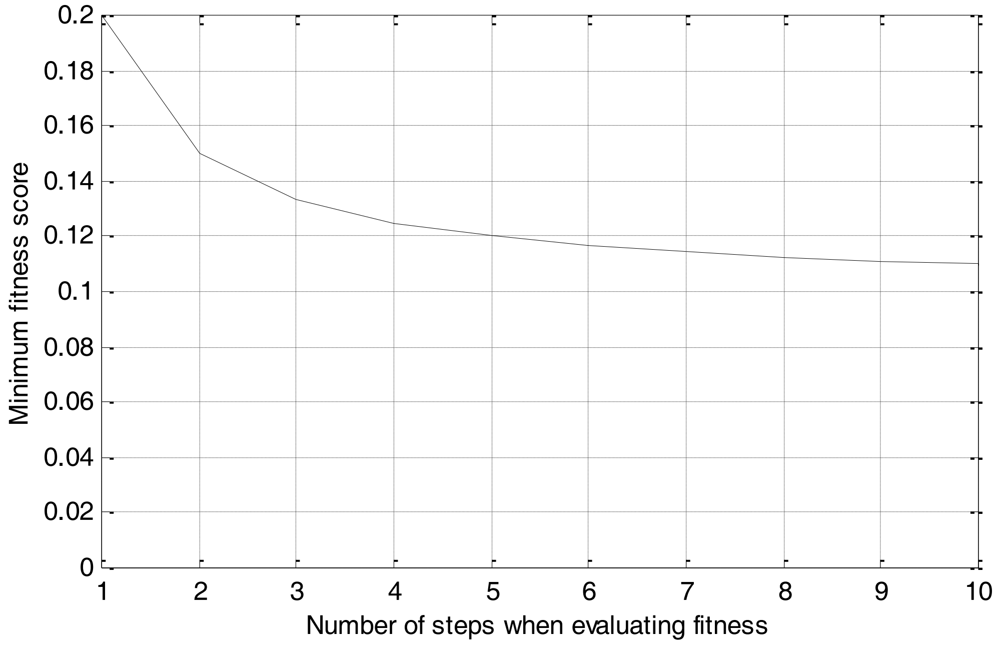


Figure 5.1: Optimum fitness value against number of steps in fitness calculations

The canonical implementation of each EA was not used. The differences were that the GA was encoded using real valued numbers instead of bit strings and the ES did not have its strategy parameters encoded in the chromosome. The mutation and recombination rates were kept constant during each evolutionary run. However, the reflexes were evolved many times, using a combination of different probabilities for mutation and recombination, in an attempt to find the optimum for each EA. This would allow a fairer comparison when selecting which EA could most successfully evolve a reflex.

The GA was implemented with the number of parents set to 10 and number of offspring set to 20 (to make the population sizes similar to the sizes used in the ES). Each gene in a new offspring was mutated with a probability set by the mutation rate (p_m=0 was no mutation, p_m=1 for always mutating). Mutation was performed by adding a normally distributed random number (mean of 0 and deviation of 0.5) to the gene. This was to allow mainly small changes to a mutated gene although occasional large changes could take place. Recombination was carried out as a single point crossover between two randomly chosen parent chromosomes to produce a single offspring. The recombination rate was used as the probability for which parent each gene came from. When this was 0 all genes would come from the first parent and when it was 1 all genes came from the second parent. When set to 0.5, there was a 50% probability that the gene would come from either parent. Selection of a chromosome was probabilistic, based on its fitness and was not elitist.

A $(\mu + \lambda)$ ES was used since the elitist nature of selection means that the mutation and recombination rates do not need to be optimal (and therefore do not need to be part of each chromosome). This allowed the mutation and recombination rates to be set to fixed probabilities. Mutation was performed in the same way as in the GA and recombination was a discrete recombination with dual parents. Selection was deterministic and simply chose the μ chromosomes with the best fitness. The population sizes were $\mu = 10$ and $\lambda = 20$.

The EP was similar to the ES in that it can be seen as a $(\mu + \mu)$ ES, but with no recombination. To make the selectable population sizes similar to the other two EAs, μ was set to 15. Mutation was performed by adding a normally distributed random value to a gene (as with the other two EAs). Selection is tournament based, where each chromosome is pitted against ten others chosen at random from the population. For every occasion where the chromosome's fitness is better, a win is recorded. The parents for the next generation are those with the highest number of wins.

Each EA was tested for a range and combination of mutation and recombination rates from 0 to 1 (although as recombination is not used in the EP, different recombination rates had no affect on the evolutionary progress). The final best fitness scores from these evolutionary runs are shown in Figure 5.2.

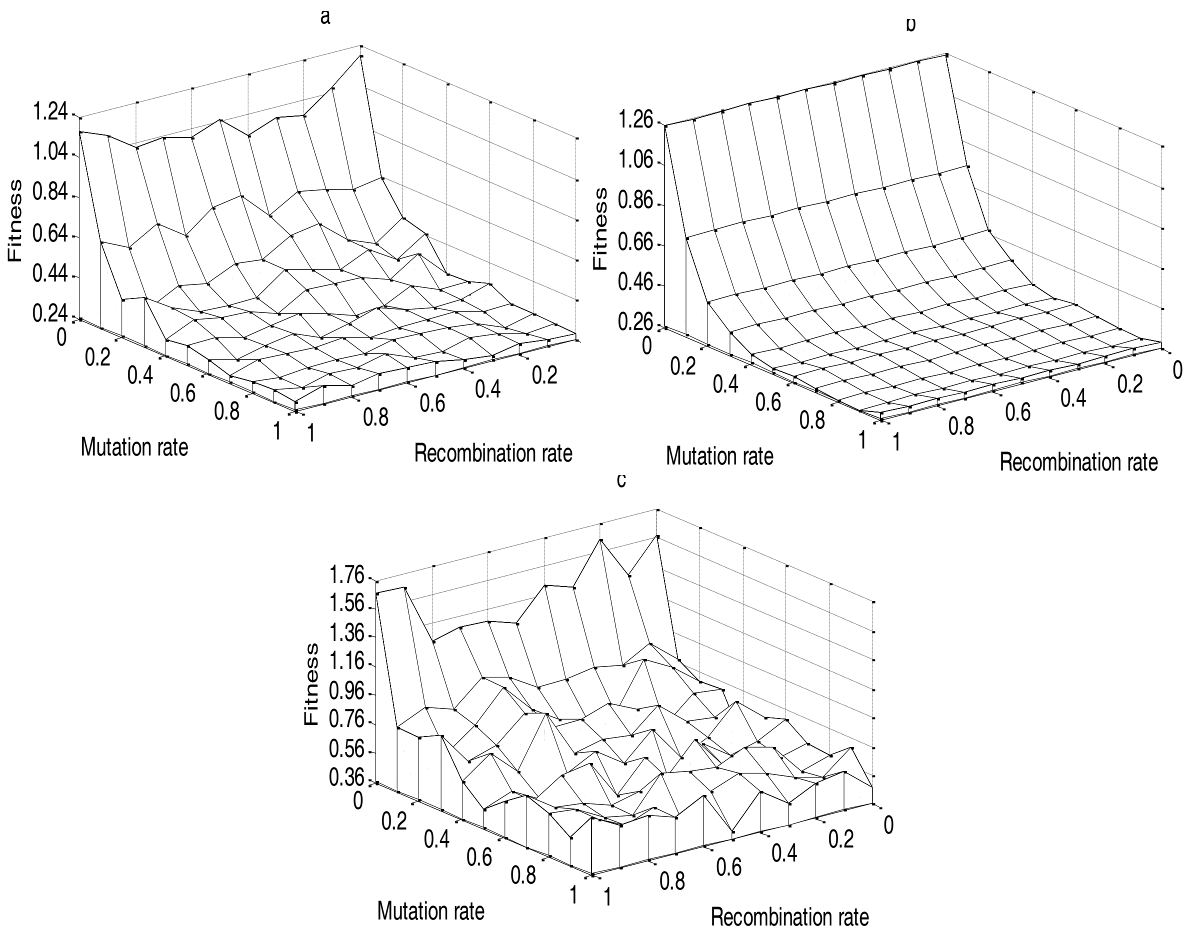


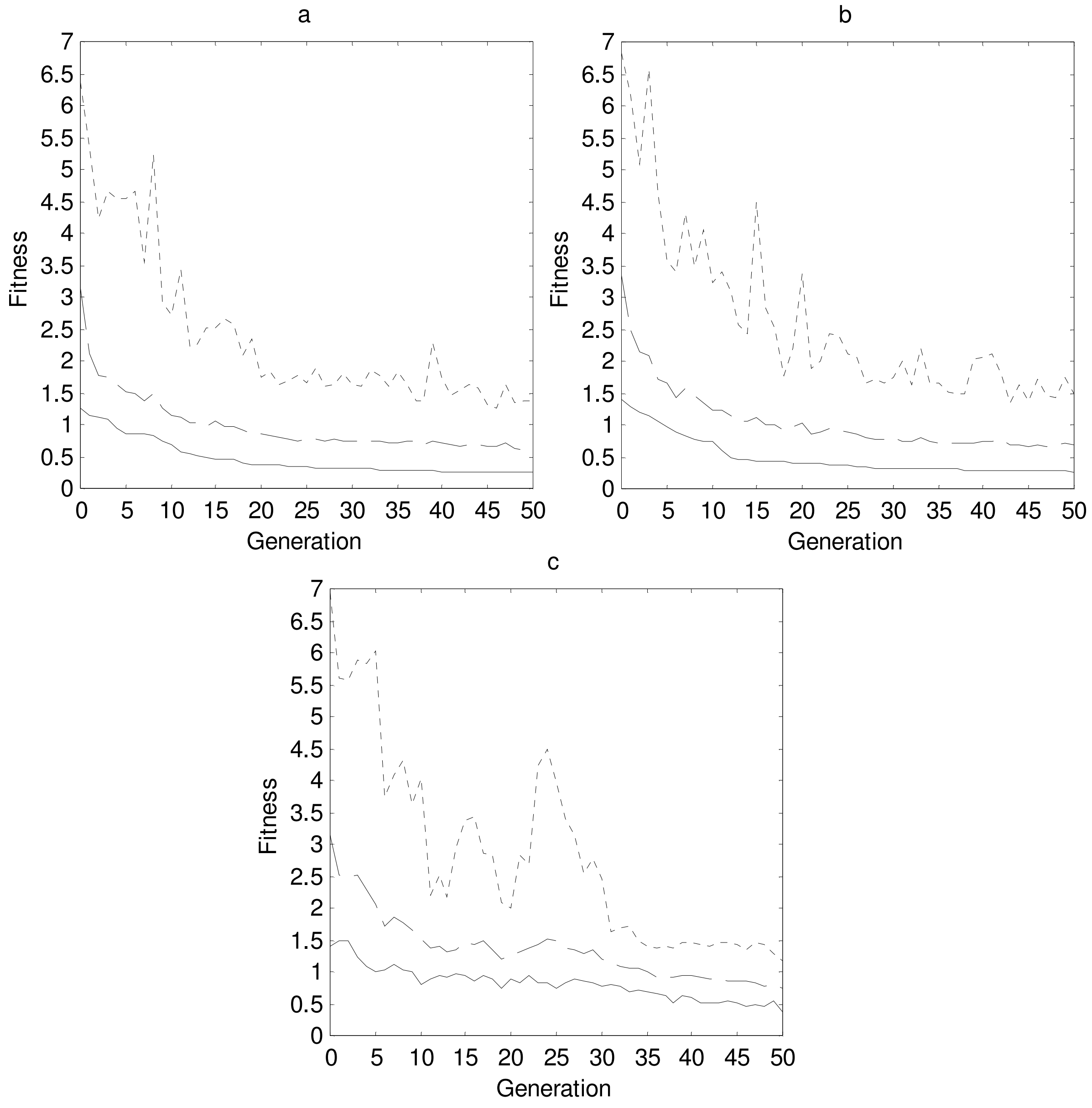
Figure 5.2: Best fitness after evolution given the specified mutation and recombination rates. a) ES b) EP (Recombination rate is always 0) c) GA. Lower fitness scores are better.

Table 5.1 shows the fitness score at the termination of evolution and what value each component (offset, overshoot and rise time) contributed to this fitness score. Since the threshold for termination was set to the optimum fitness, all evolutionary runs continued for the full fifty generations.

Table 5.1: Final fitness scores of different EAs, for given best mutation and recombination rates. Lower fitnesses are better.

EA	Fitness	Offset	Overshoot	Rise time	p_m	p_c
ES	0.2466	0.0077	0.0151	0.2237	0.8	0.1
EP	0.2680	0.0216	0.0332	0.2133	0.9	N/A
GA	0.3672	0.0660	0.0810	0.2201	0.6	0.4

The evolutionary progress, which caused the best fitness scores, for each EA is shown in Figures 5.3. In each figure, the best fitness is shown as a solid line, the average fitness as a dashed line and the worst fitness as a dotted line.



Figures 5.3: Evolutionary progress for each EA evolving a reflex. In each graph, the solid line is the best fitness, the dashed is the average and the dotted line is the worst fitness. a) ES b) EP c) GA. Lower fitness scores are better.

The positions of the actuators under the control of these evolved reflexes are shown in Figure 5.4. As can be seen in all the graphs, there is very little offset or overshoot. The factor that separates the abilities of the reflexes is the rise time. The ES scored better than the other EAs, but because it minimised the rise time so much, instability was introduced into the control of the actuator (causing oscillations around the target positions). At this stage there was no part of the fitness function that would penalise instability. Section 5.2.3 will examine the development of a more stable system by modifying the fitness function.

As can be seen in the results presented above and below, the $(\mu + \lambda)$ ES allows the reflex to be evolved more successfully than with the other EAs. This was used as the main evolutionary method for all the remaining experiments, and this decision is discussed further in section 5.3.1.

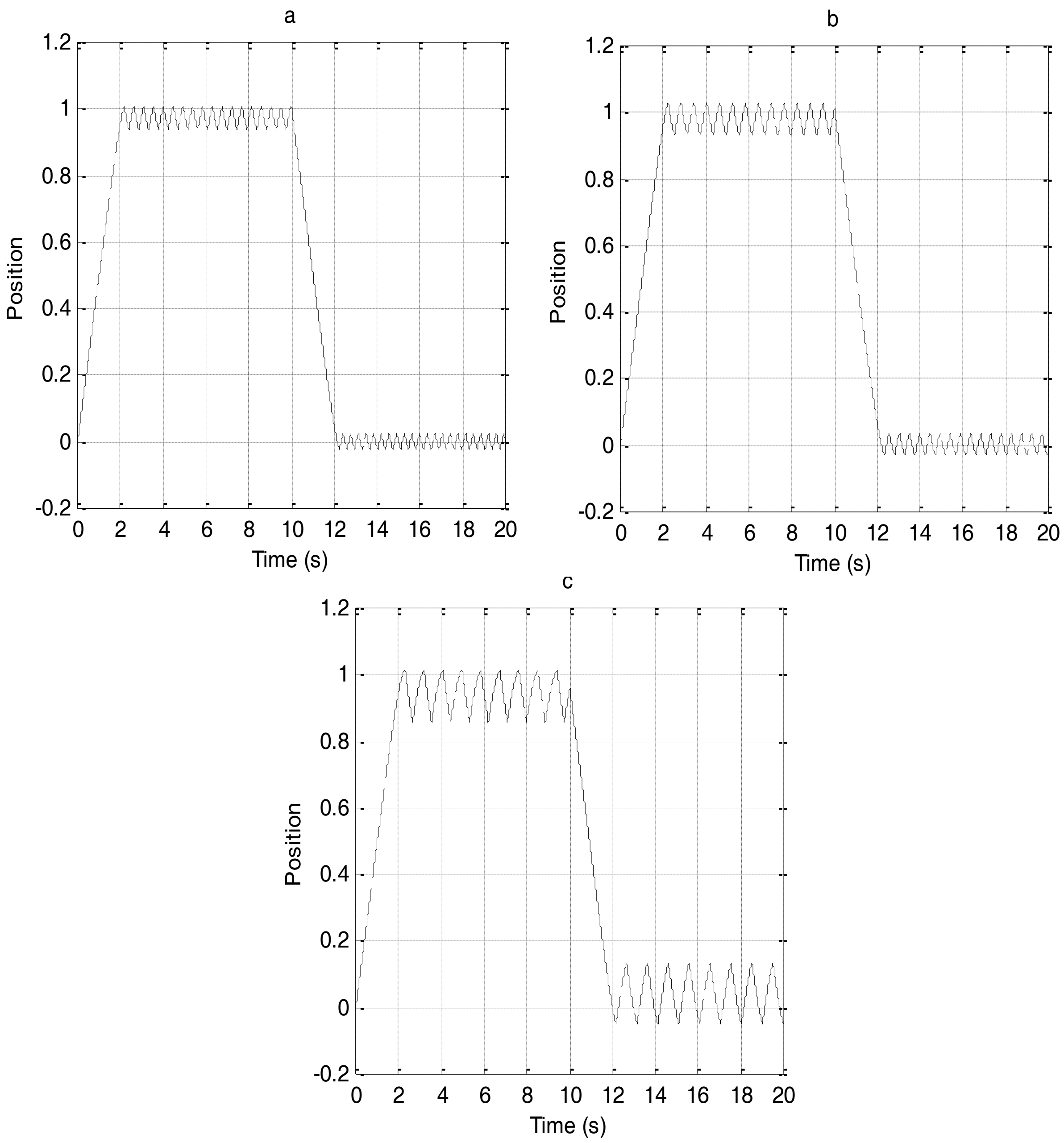


Figure 5.4: Time plot of the position of the linear actuator controlled by the reflex. Control by best solution evolved by a) ES b) EP c) GA.

5.2.2. Evolutionary Parameters

The initialisation of the genes in the chromosomes before the start of evolution can affect the convergence of the EA if the fitness landscape is complex and the EA used has only a small probability of searching the entire solution space. The use of low mutation rates (such as in a standard GA) will make the EA especially sensitive to the initial conditions of the chromosomes. Indeed, increasing the initial range of random real values for the chromosome (to ±4) allowed the GA to evolve a network to a similar fitness score of the ES and EP (and it is these results with the new initial conditions that are shown in Figures 5.1 to 5.4).

The population sizes (of parents and offspring, and therefore total size) will also affect how well the EA converges to a solution and how good that solution is in comparison to the global optimum. If the number of parents is too low, more generations will be required to search enough of the solution space, to give an optimum result. If the offspring population size is too low, a similar problem will be faced, since there would not be sufficient variation between generations. Increasing the population sizes will not have an adverse effect on the ultimate convergence of the EA (in fact, the number of generations required should decrease, since more of the solution space is being searched in every generation). However, the amount of real time required increases, which is not always desirable or feasible. Figure 5.5 shows how varying the total population size and number of parents (therefore also number of offspring) affects the best fitness score of the evolved population.

The initial populations had 30 chromosomes in total and 10 parents (33%). These sizes allowed reflexes to evolve quite successfully, as shown in the previous section (even though Figure 5.5 would suggest otherwise). However, there were still small improvements which could be made to reach the optimum.

With a small percentage of the total population being used for the parent chromosomes, the final fitness is quite good. It reaches an optimum and then has a steady gain as the percentage increases past that point. The optimum fitness occurs around 10% for the parent size (the recommended ratio for parents to offspring for an ES is 1:7≈12.5% of total population size should be parents).

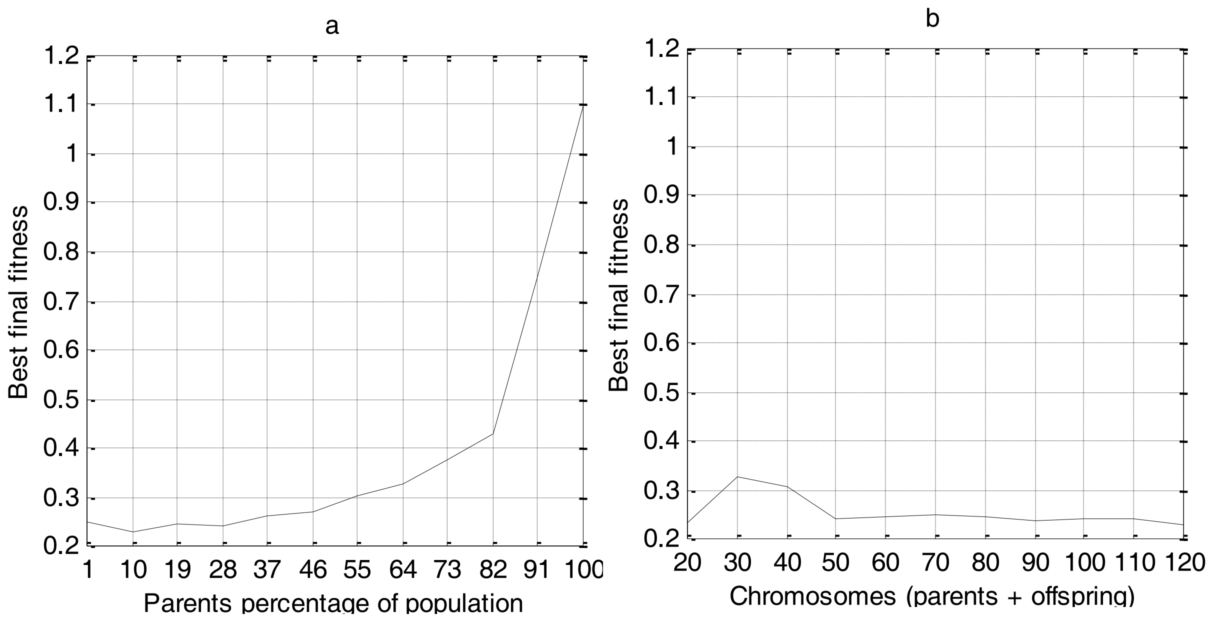


Figure 5.5: The effect of population size and number of parent chromosomes on best fitness after evolution (lower fitness is better)

The fitness score when varying the population size (the percentage of parents was fixed at 20% in these cases) decreased (which is an improvement with the fitness scored as it is) as population size increased. However, the improvements were only slight, which may not be enough to justify the additional computation time required for larger populations.

While the experiments conducted previously have used fixed rates throughout the course of each evolutionary run, ESs actually have the mutation and recombination rates as part of each chromosome, therefore allowing them to evolve and reach an optimum value. Extensions to the standard GA use a mutation rate which starts off very high and decreases as evolution progresses (similar to the temperature of the system in simulated annealing). This lets the EA find the optimum range of solutions early during the evolution while still permitting the fine tuning required towards the end. Figure 5.6 shows the effects of a variable mutation rate decreasing with number of generations passed and fitness respectively. In both graphs, the best evolved fitness is shown against the initial mutation rate. A GA was used as the EA in these cases.

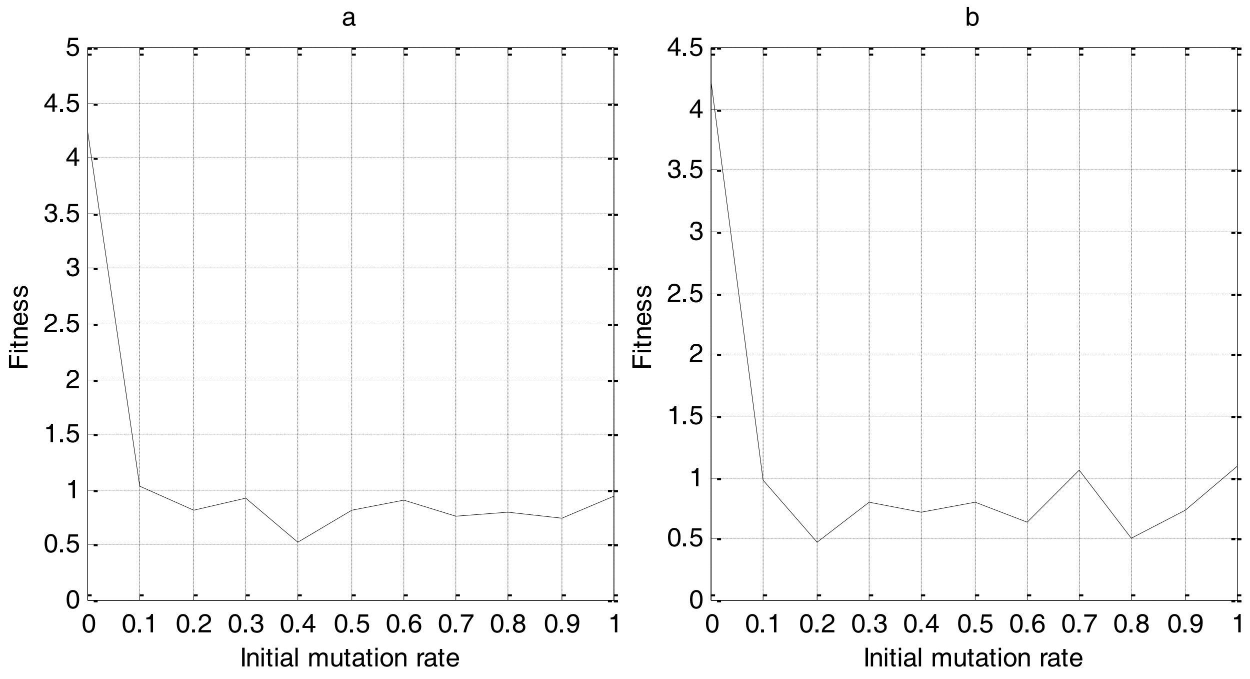


Figure 5.6: Best fitness against initial mutation rate, decreasing linearly with

a) Number of generations passed b) Best fitness. Lower fitness scores are better.

As can be seen from Figure 5.6, all initial mutation rates result in networks of similar fitness to the original GA, which had a fixed mutation rate through the entire evolutionary process. A mutation rate of 0 will never work, since no new gene values can be introduced into the population. All other values for the mutation rate should produce working reflexes, since the mutation rate decreases as evolutionary progress is made, and therefore cover a range of mutation rates which are suitable.

The performance of the EA is affected differently by mutation and recombination rates depending on whether the population is elitist or not. When elitism is used, the parents from the current generation are eligible for selection to become parents in the next generation. This means that mutation and recombination rates can be higher with less chance that chromosomes with good fitness scores will be lost to genetic operations. Although EAs which are elitist can become stuck in local optima, the higher genetic operator probabilities can help in avoiding this problem. However, setting these rates too high can mean too many changes take place and offspring are never any better than the parents.

The easiest way to test the affect of the rates with and without elitism is to test them using a $(\mu + \lambda)$ ES (elitist) and (μ, λ) ES (not elitist). Figure 5.7 shows the best fitness scores achieved for a range of mutation and recombination rates.

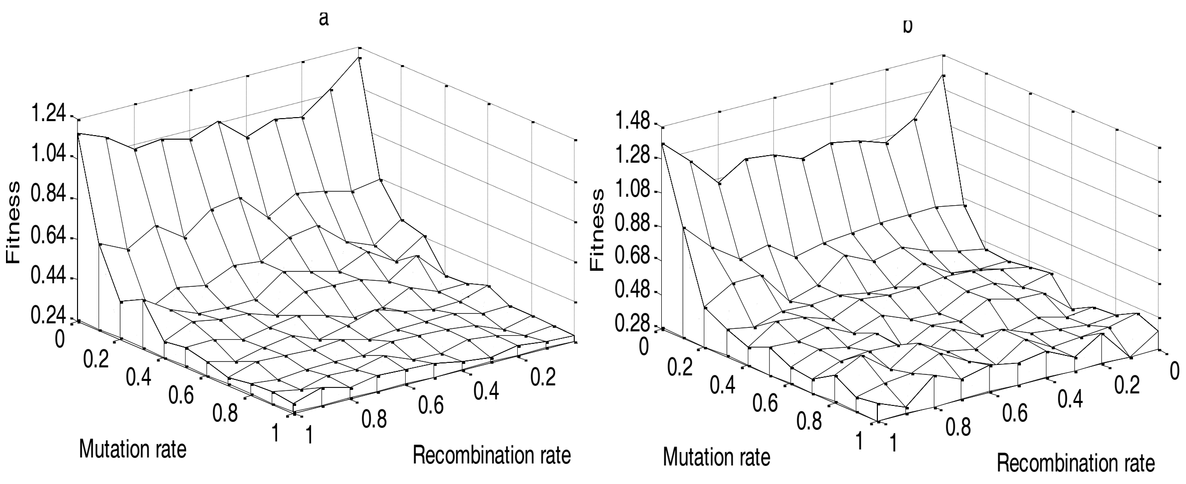


Figure 5.7: Comparison of final fitness scores for different mutation and recombination rates, when using a $(\mu + \lambda)$ ES (a) and (μ, λ) ES (b). Lower fitness scores are better.

The lack of elitism, demonstrated by the (μ, λ) ES in Figure 5.7b, has a slight impact on the fitness score of the final reflex which is evolved, which is consistently higher than the fitness score achieved using the elitist $(\mu + \lambda)$ ES.

The number of target position steps to control the actuator for, during fitness evaluation, was used to see if the network would require additional training when the tested target was not one which the network was trained for. A range of different steps was investigated, with the best evolved fitness for each number of target steps shown in Figure 5.8.

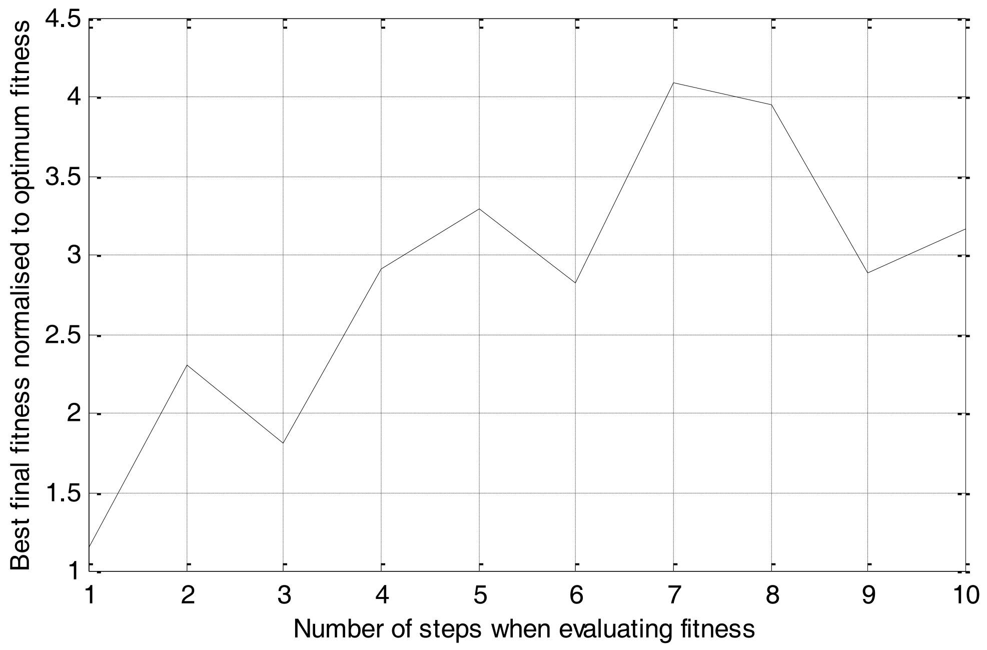


Figure 5.8: Best final fitness scores when evolving with different numbers of target positions (normalised to Figure 5.1 - lower fitness scores are better.)

As can be seen from the above graph, the networks evolve better with lower numbers of target positions. However, this does not take into account the abilities of the network to control the actuator to target positions which they were not trained for. Figure 5.9 shows this information for three of the networks. The three graphs represent the networks being tested in the task of controlling the actuator from position 0, to either 0.1, 0.5 or 1.0 (a, b and c respectively) and then back to 0. In each graph the solid, dashed and dotted lines represent the position of the actuator as controlled by networks evolved with 1, 5 and 10 target positions.

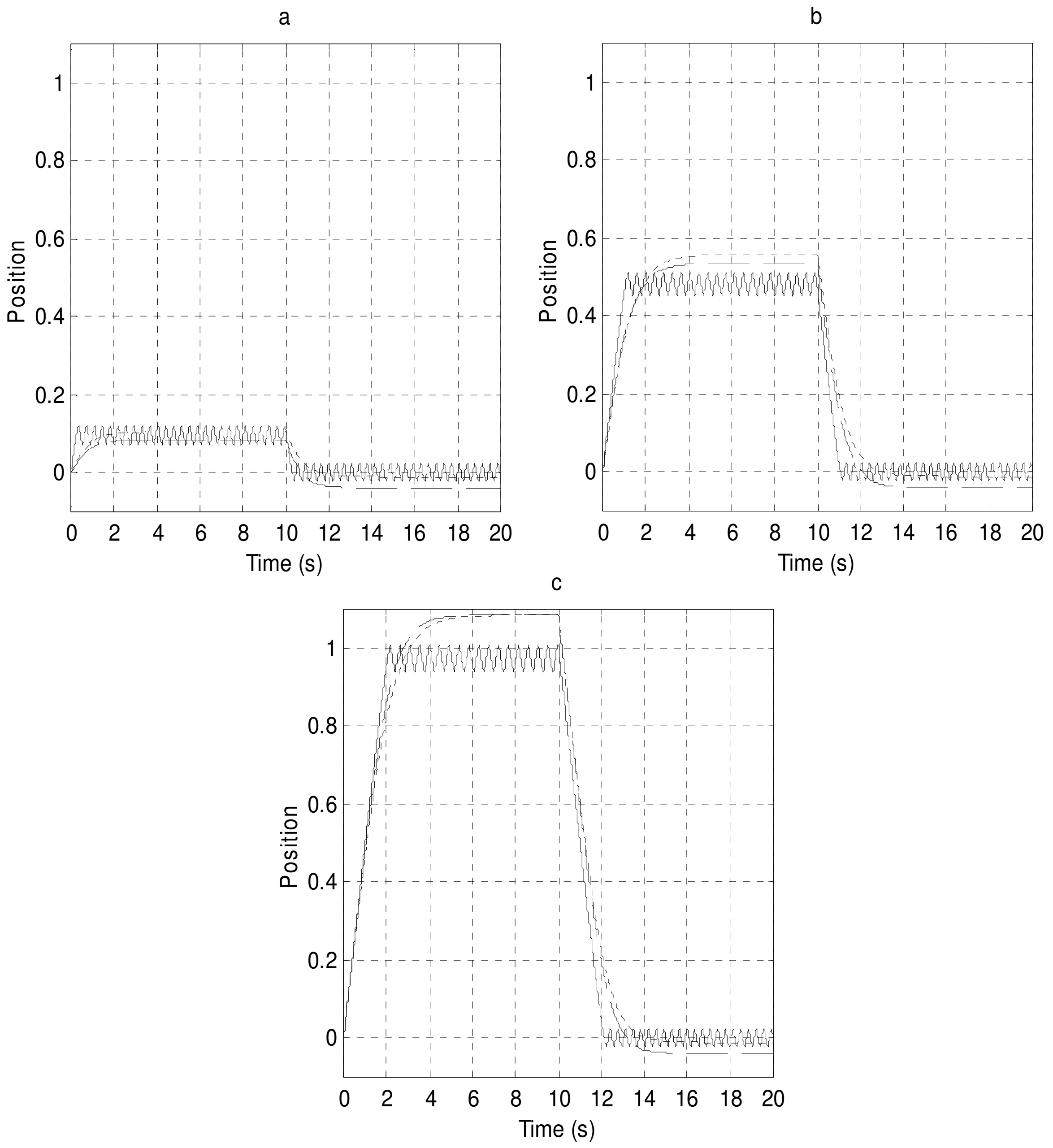


Figure 5.9: Time domain performance of reflexes evolved using and tested in controlling to different numbers of target positions. In each graph the solid line represents a reflex evolved with one target position, the dashed line is a reflex evolved for five target positions and the dotted line is a reflex evolved for ten target positions.

The solid line in Figure 5.9 is from the network which was evolved in the first section of this chapter, which is why it oscillates. The networks which were evolved for more than one target position did not oscillate, although they had an offset and slightly longer rise times (this was also true of the other networks with multiple targets, although they are not shown in Figure 5.9).

The graphs display a lack of ability to generalise by the networks. This manifests itself in the offset and overshoot which are quite low for the early target positions (graph a in Figure 5.9) but get worse for higher values (graphs b and c). This is most likely due to an over-fitting problem; looking at section C.2 it can be seen that the network is bigger than it needs to be (the output of node three is not used meaning that node could be removed from the network without any change in performance).

5.2.3. Fitness Function

Probably the most important element for evolving the reflex is the choice of fitness function; an inappropriate scoring mechanism will either lead to the solutions not converging to the optimum, or will produce solutions which do not perform the desired behaviour. This has already been demonstrated with the instability of the initial ES evolved reflex.

As described in the previous chapter, there were many variables which could be used to score the fitness of each chromosome. Previously the fitness has been calculated as the summation of the scores for rise time, offset and overshoot. This is the main reason for the instability and oscillation of the reflexes until now. The rise time always contributed the largest values to the fitness score, so if the evolutionary progress could minimise that component the fitness score would almost definitely be better. Reducing the rise time means that the speed of the actuator will be high for as long as possible, which caused overshoots, and some networks in trying to correct this were unstable, causing oscillations of the actuator position.

The correct approach would be to tune the fitness function by weighting each of the fitness score components. This would mean that the fitness function would now become

$$fitness = \left(k_{off} \times \frac{offset}{target} + k_{over} \times \frac{overshoot}{target} + k_{rt} \times \frac{rise\_time}{total\_time} \right) \div 2 \qquad \text{Equation 5.1}$$

where k_x is the weight factor for the multiplied fitness component.

The effect of increasing the weight of a component is to increase the fitness scores dependency on that component. Since the evolutionary task here is to minimise the fitness score, increasing the weight of a fitness component will force the evolutionary process to optimise that value more than the others (if the weight is more than 1).

This is shown in Figure 5.10. Three new reflexes were evolved, but each one had a greatly exaggerated (multiplied by 100) weight for a single fitness component. Graphs a, b and c represent the ANNs that were evolved with large multipliers for offset, overshoot and rise time respectively.

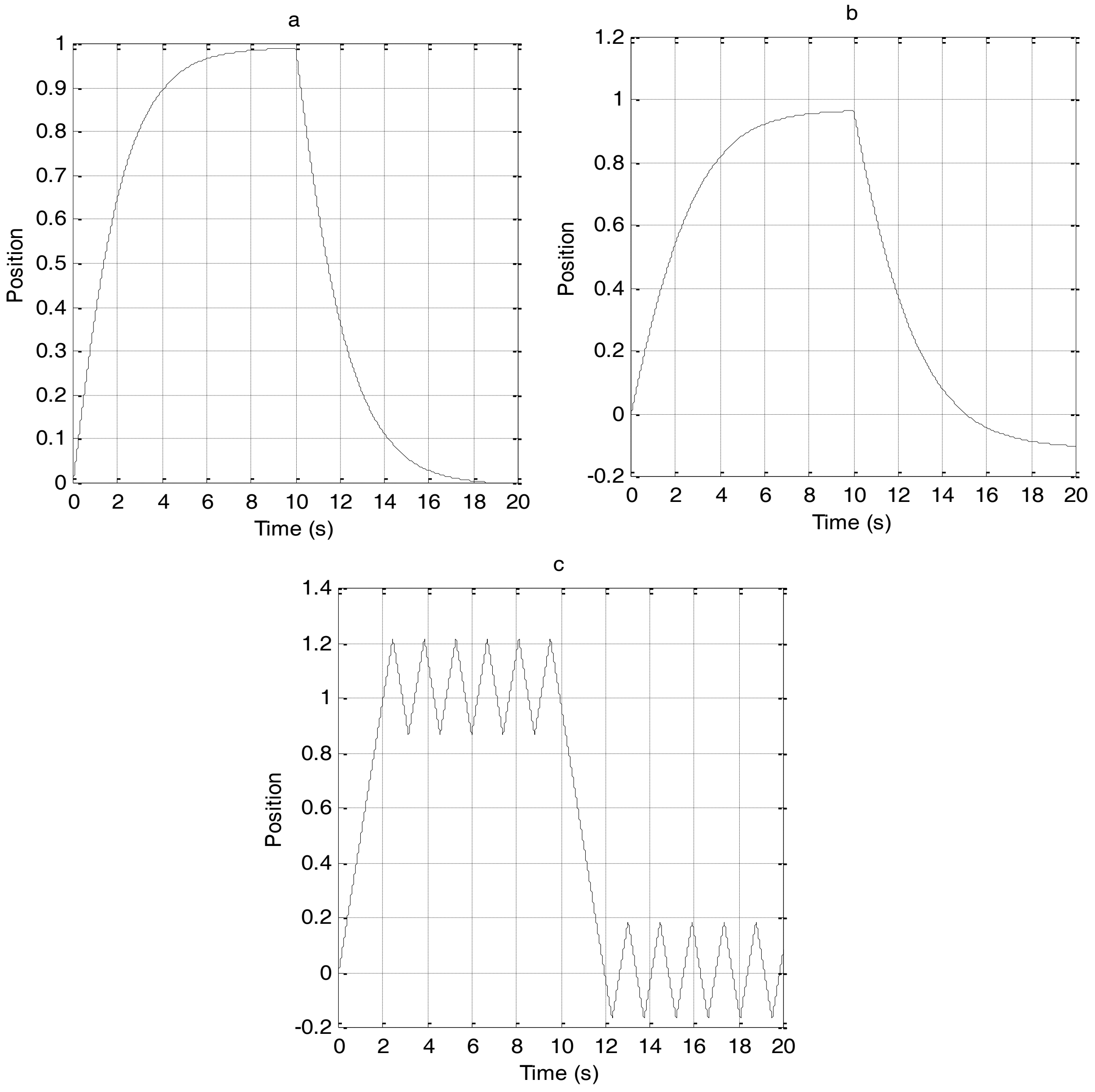


Figure 5.10: Effect of increasing weighting of fitness components. Fitness components weighted heavily in favour of a) offset, b) overshoot, c) rise time.

In the first plot in Figure 5.10, the weight of the offset component has been increased. Since the offset is only measured at the end of the test time, the EA has created a reflex which has a minimal final offset when it is tested. There is no overshoot, but the rise time has suffered (because there will be very little improvement in fitness score for shorter rise times). Graph b shows the effect of increasing the weight for overshoot. The EA has created networks which have no overshoot for the first phase, but a slight overshoot in the second phase. The rise time is long again (in fact, the target position is never reached in the first phase). Finally, when the weight for rise time is increased, the reflexes can reach the target position in the shortest possible rise time. However, this leads to large oscillations around the target positions. Therefore to achieve the best time domain performance from the reflexes, the weightings for the fitness components should be chosen to reflect the desired behaviour of the actuator position.

Apart from changing the weights of the fitness components, the other option is to change how each fitness component is scored. The most obvious is to change the offset so that it becomes the integral of the offset with time, rather than the final offset. Oscillations would worsen the fitness score when the offset is scored like that, and would be eliminated as better solutions are found.

Another change to the way fitness components are scored can be made to the rise time (since this was the component which causes the oscillations in the actuator position). If the rise time is scored like the offset and overshoot, i.e. it is normalised to the optimum rather than the full scale, then its influence on the fitness score will be reduced. Therefore all of the fitness components would need to be minimised in a more equal manner to achieve a good score.

The effects of changing how these two fitness components are recorded is shown in Figure 5.11.

As demonstrated in Figure 5.11, slight changes to how the fitness function is scored can result in quite different results. In the case of the offset being integrated, the rise time is short, and there is no offset or overshoot, making it a potentially good method for scoring fitness. When the fitness component for rise time is measured against the optimum, there is a small overshoot and offset, and the rise time is even closer to the optimum. However, this has led to oscillations again.

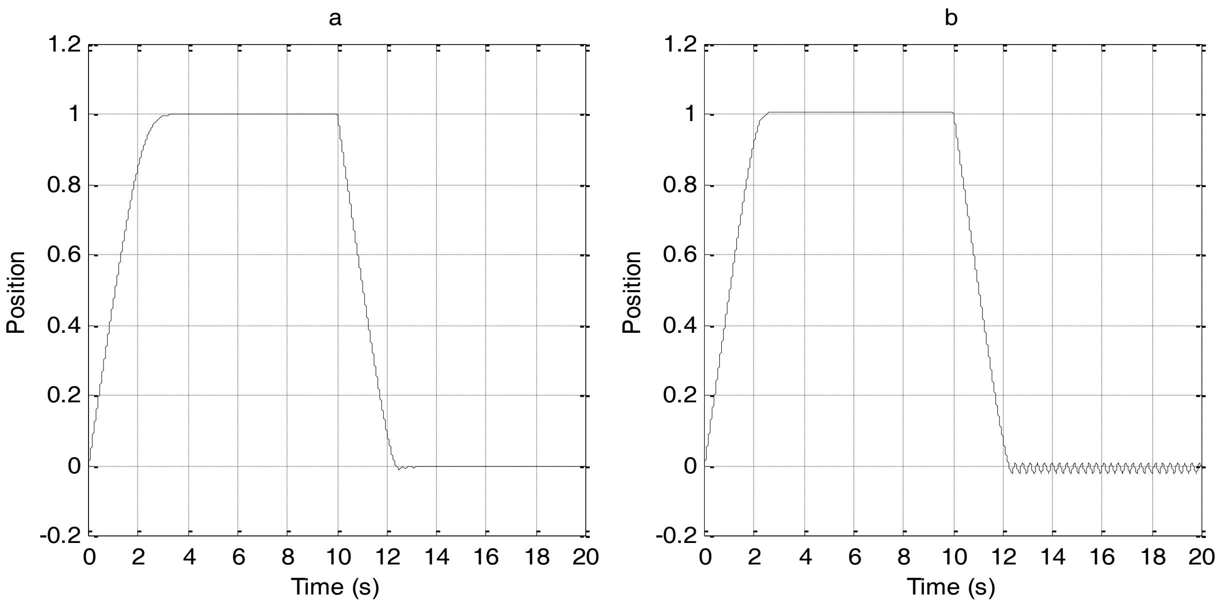


Figure 5.11: Time response of actuator when fitness components are changed. a) Offset integrated with time. b) Rise time measured as time past optimum rather than time from start.

5.2.4. Network topology

The network topology will have a large impact on the performance of the reflex. Some topologies will never[1] be able to control the actuator, while others will have a range of performance. The fitness landscape is not shown at this point, since all the scores were of a similar magnitude to the reflexes evolved in the first section of this chapter.

Graphs a, b and c of Figure 5.12 show the position of the actuator controlled by networks with one, two and three hidden layers respectively. Within each graph the solid, dashed and dotted lines represent networks with one, three and five neurons per hidden layer. Although the fitness scores were similar for all networks, there is noticeably more oscillations in the networks with more hidden layers.

As shown in section 5.2.1, small ANNs were able to act as a reflex with good results. It may also be possible to use smaller networks, since one of the networks

[1] Or at least take an excessively long time that makes it impractical to continue to evolve the reflex until it does work.

was able to evolve in such a way that one of the nodes was not required (see section C.2). This left little room for improvement by increasing the size of the networks. However, larger networks would be required for changes such as more complex actuators, more inputs (therefore more information to deal with in the ANN) or lack of recurrent connections. The required size of networks to attain good fitness scores will be reported in each section where they are required.

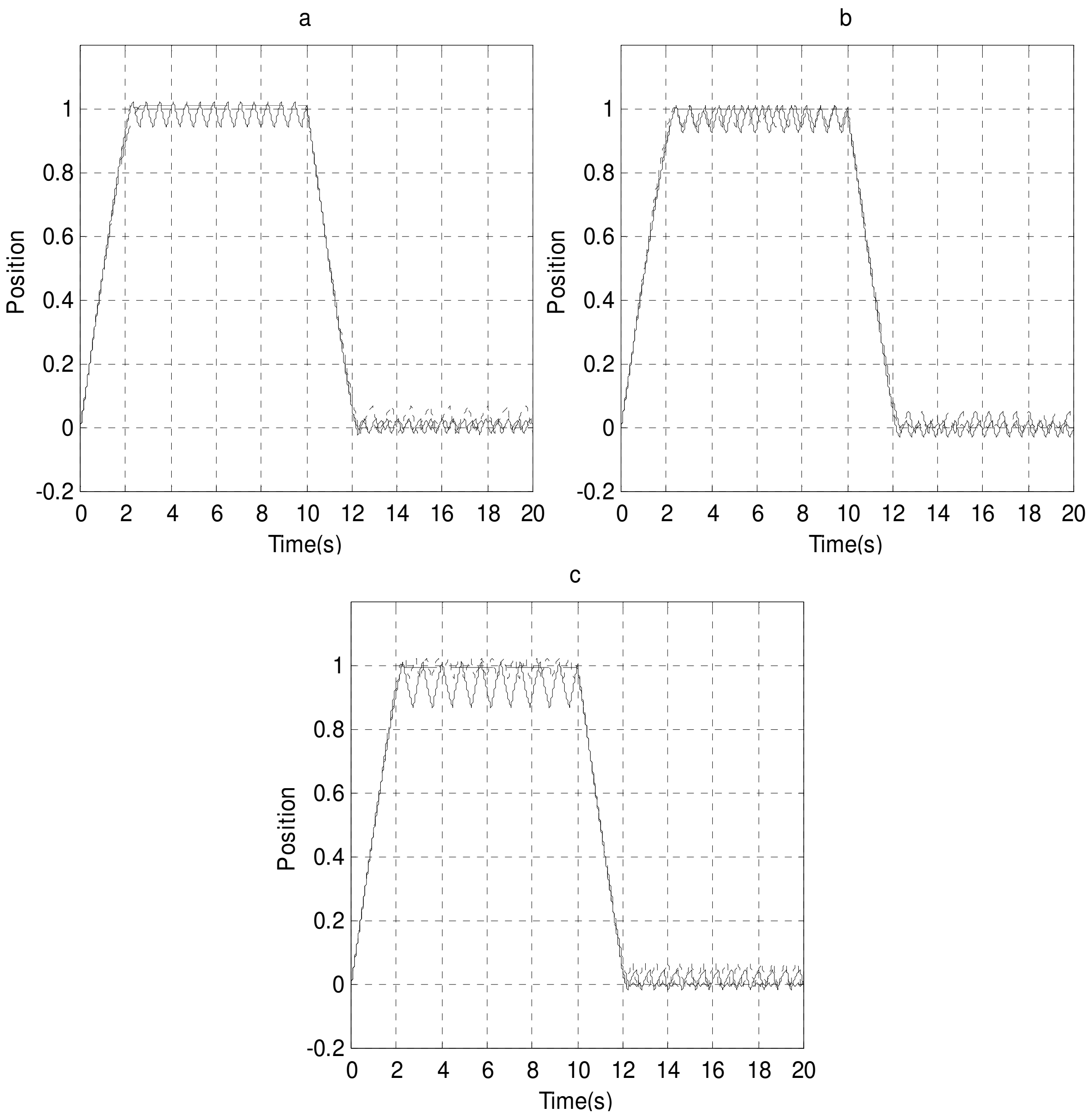


Figure 5.12: Time responses of actuators controlled by reflexes with different network sizes. a) one hidden layer b) two hidden layers c) three hidden layers. In all graphs, the solid, dashed and dotted lines represent one, three and five neurons per hidden layer.

Networks were evolved which had no recurrent connections (but were still allowed multiple forward connections). Other networks which had only connections from one layer to the subsequent layer were also evolved. This made the problem for the network similar to that of a pattern recognition or mapping task. Comparisons between networks with different connection strategies are shown in Figure 5.13.

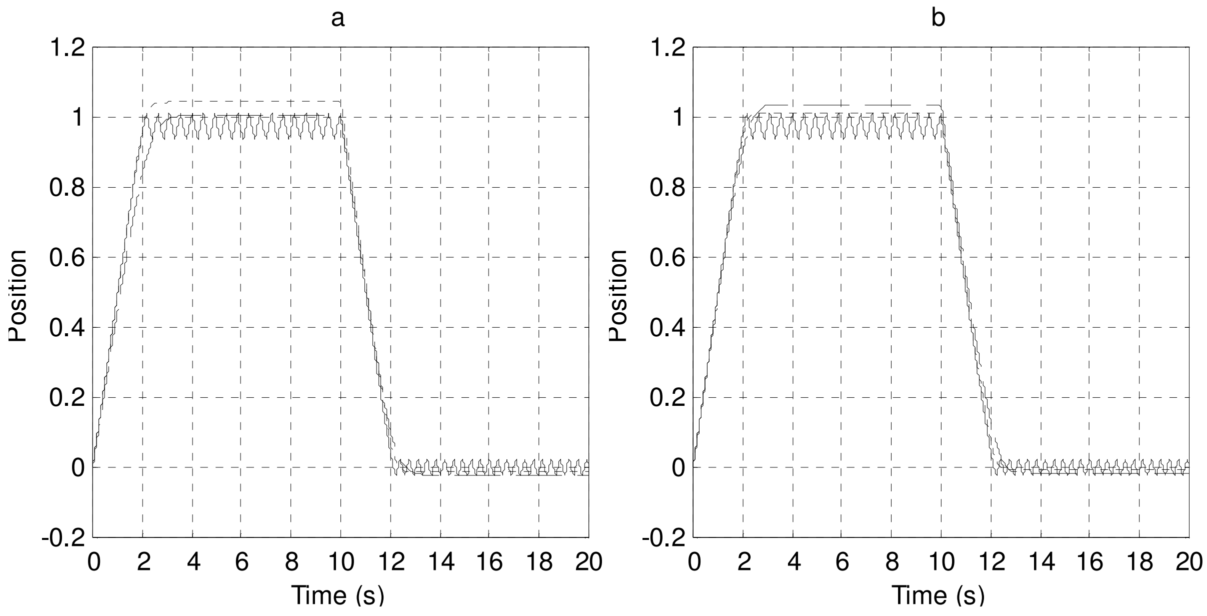


Figure 5.13: Comparison of effect of different types of recurrent and forward connections. a) Only forward connections between adjacent layers. b) No backward or sideways connections between layers. Solid line is fully recurrent network in both a and b. Dashed line is networks the same size as with recurrent connections (one hidden layer with three neurons). Dotted line is network with best fitness for that type of connections (a has one hidden layer with seven neurons and b has one hidden layer with four neurons).

Although the networks without recurrent connections performed well, their fitness and time domain control was not as good as networks with recurrent connections (for the same network topology). The networks which had only forward connections between adjacent layers typically had a long rise time, and a larger offset than the networks with recurrent connections. In the case of the network with no recurrent connections (but any amount of forward connections), they typically had the largest offset although the rise time was not as long as the reflexes with only forward connections to adjacent layers.

5.2.5. Actuator Type

Two different actuators were used in evolving the reflex. The investigation so far has used a simple linear actuator, described in the previous chapter. A more realistic actuator (a simulated D.C. motor, as described in Chapter 4) will be implemented at this point, and comparisons between the ability of the reflexes to control both types of actuator will be performed.

Initially the reflex which had been evolved using the simple actuator was tested to see how well it could control the D.C. motor. The motor parameters used ($b=0.1$, $J=0.01$, $K=0.01$, $L=0.5$, $R=1$) were such that the simulated motor had a low maximum speed of 0.5 rad/sec. This would enable the generalisation ability of the reflex network (in controlling an actuator with similar speed but different dynamics) to be evaluated. To compare this, another reflex was evolved directly on the D.C. motor. The remaining parameters were the same as when the reflexes were evolved on the linear actuator. The resulting time domain responses from the D.C. motor under reflex control is shown in Figure 5.14.

When the reflex (evolved using a linear actuator) controlled the D.C. motor, large oscillations were present on the position of the actuator, making it quite unsuitable. Conversely (as would be expected) the reflex evolved using a D.C. motor, when controlling the linear actuator, produced a damped response and had a small offset.

The reflexes were then tested on their ability to control a faster version of the same type of actuator they were evolved on. The linear actuator's speed was increased to 35 rad/sec, so that it would be similar to the faster D.C. motor (with parameters $b=3.5077\times10^{-6}$, $J=3.2284\times10^{-6}$, $K=0.0274$, $L=2.75\times10^{-6}$, $R=4$) which had a maximum speed of 35 rad/sec. As before, reflexes were evolved "natively" on these faster actuators to allow comparisons between the control abilities.

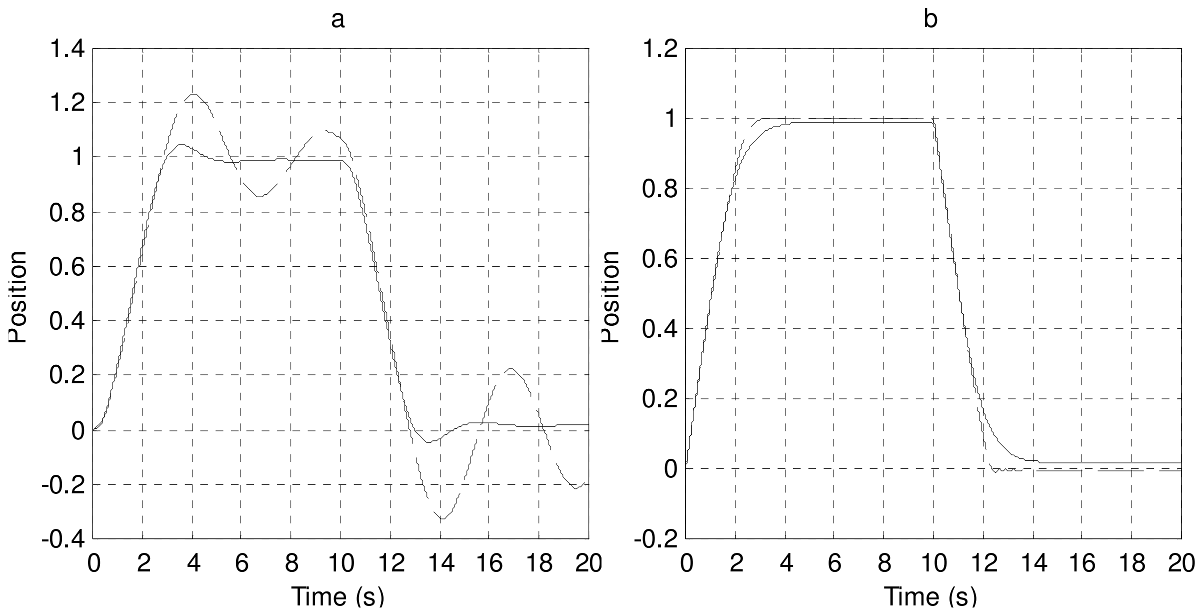


Figure 5.14: Comparison of reflex generalisation ability. a) Control of a D.C. motor by a reflex evolved using a D.C. motor (solid line) and a reflex evolved on the linear actuator (dashed line). b) Control of a linear actuator by the same two reflexes (solid line reflex was evolved on a D.C. motor, dashed line was the reflex evolved on the linear actuator).

Figure 5.15 shows the abilities of the reflexes to control actuators at different speeds than they were evolved with. The reflex evolved with a slow linear actuator (dashed line, graph a) performed well, having very little offset or overshoot although the rise time was longer (than the reflex evolved on the fast actuator). The reflex which was evolved using a slow D.C. motor controlled the fast D.C. motor similarly to the reflex which was evolved with the fast D.C. motor (although there was more overshoot).

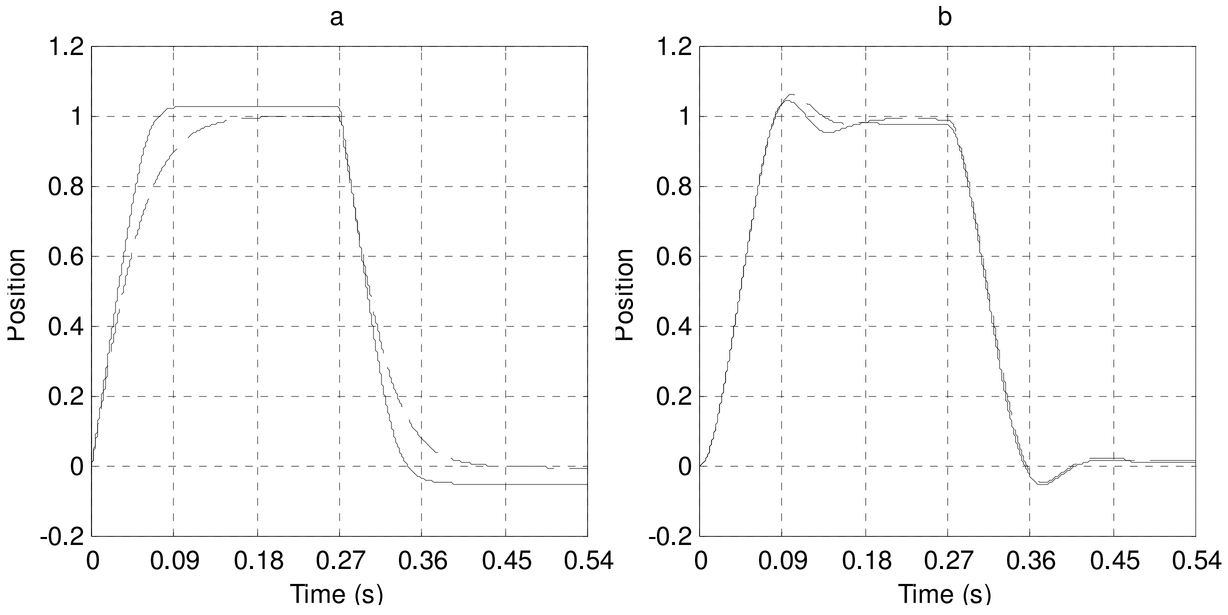


Figure 5.15: Reflex ability to control different speed actuators. a) Fast linear actuator controlled by a reflex evolved on it (solid line) and a reflex evolved on a slow version (dashed line). b) Fast D.C. motor controlled by a reflex evolved on it (solid line) and a reflex evolved on a slow D.C. motor (dashed line).

One reason that the slower reflexes may have been able to control the fast actuators so well is because at the sampling times used for testing the networks, both types of actuator would take similar numbers of sample periods to reach a position of 1. To check whether this was the case, the reflexes which had been evolved using slow actuators were tested again on the fast actuators, but using a longer time period (in fact, the same time period as used with the slow actuators, which is about 37 times longer). The performance of the networks running to this time scale are shown in Figure 5.16.

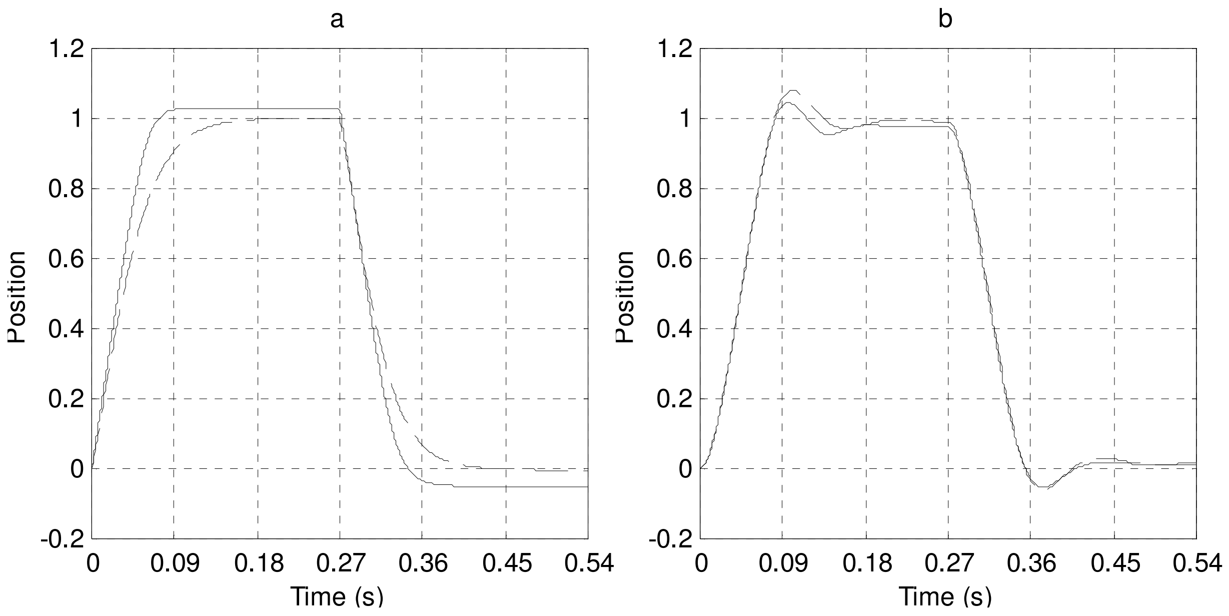


Figure 5.16: Comparison of networks running at different sample periods. a) fast linear actuator (solid line is reflex evolved to use fast linear actuator, dashed line is reflex evolved with the longer sample times). b) As a, but for a D.C. motor

The time domain responses shown in Figure 5.16 are similar to those in Figure 5.15, showing that the reflex is capable of controlling the actuators to the same level when the sampling time of the motors is different from when the reflex was evolved.

5.2.6. Network inputs and Feedback

There will always be at least one input to the reflex, and that is the target position for the actuator. As previously described, in section 5.2.1, the target position input will be normalised to the range 0 to 1. This presents a standard interface to the higher layers in the ANS, allowing them to be ignorant of the specifics of the actuator.

The feedback from the actuator was designed to give the reflex the properties that traditional (proportional (P), integral (I) plus differential (D), or some other combination) controllers have. An example of this type of controller is shown in Figure 5.17. As is shown, the control signal for the actuator is a combination of the proportional, differential and integral of the error between the reference (input) signal and the controlled variable being fed back from the actuator. Multiplier constants for each component of the control signal are denoted K_x where x is the type of component (P, I, or D).

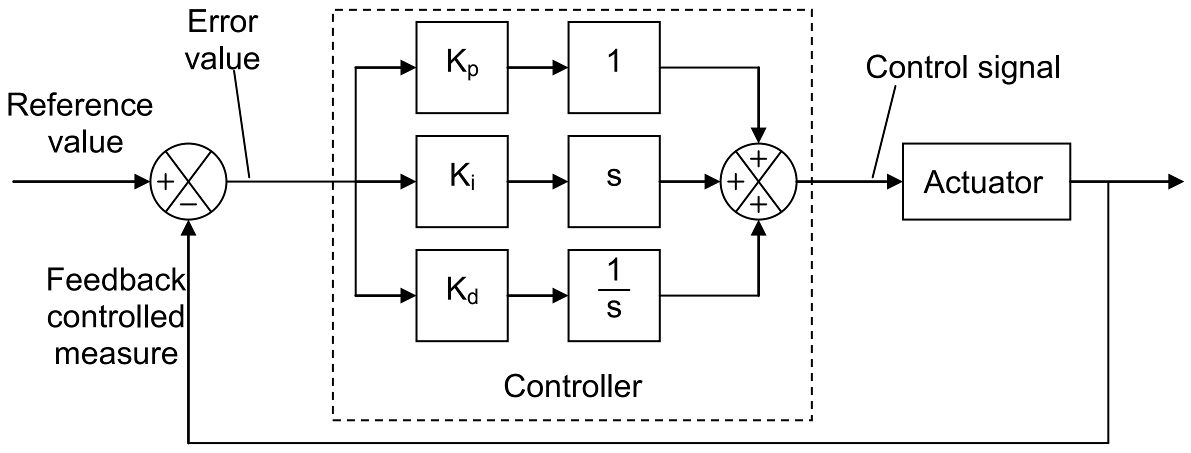


Figure 5.17: Example of a traditional controller

The elements in a traditional controller (assuming we want to control the position of the actuator, as is the task here) correspond to the feedback inputs, from the actuator, in the reflex shown in Table 5.2.

Table 5.2: Feedback corresponding to elements in a traditional control system

Feedback from actuator	Traditional control element
Position	Proportional
Integral of position	Integral
Speed	Differential

The traditional control elements listed in Table 5.2 have approximate effects on the time domain response of the actuator as described in Table 5.3. Normally when designing a system with a traditional PID-like controller, a set of minimum specifications are given in terms of offset, overshoot and rise time. The PID component constants (K_p, K_i and K_d) are tuned so that the time response of the actuator meets the requirements.

Table 5.3: Effects of traditional control elements on time response of actuator

Element to be tuned	Effect on Offset	Effect on Overshoot	Effect on Rise time
Proportional	Decrease	Increase	Decrease
Integral	Remove	Increase	Decrease
Differential	Small change	Decrease	Small change

The acceleration feedback from the actuator was not included in Table 5.2 since it is the second derivative of the controlled measure (position in this case) and would not usually have an equivalent in a PID controller for position. However, this extra feedback value can be added to the reflex to see what, if any, difference it makes to the ability of the ANN to control the actuator.

Traditional controllers were created for comparison against the evolved reflexes, in the task of moving a D.C. motor from position 0 to position 1 and then back to 0. The objectives for these controllers were the same as the evolved reflexes (to minimise rise time, offset and overshoot). This was achieved by selecting initial values for the controller constants (K_p, K_i and K_d, depending on the corresponding input configuration for the reflex) and then fine-tuning them. The control action for the best traditional controllers have been overlaid (on the graphs) by the reflex control action. The position of the actuator as controlled by the reflex is shown by a solid line. The traditional control of the actuator is given as a dashed line. Table 5.4 shows the inputs to the reflex networks, the corresponding type of traditional controller and what figure the control of the actuator is shown in. The controller constants are also given in Table 5.4. In the cases where K_i was zero, it was found that the performance of the controller was not improved by the addition of the integral component. All reflex networks have one hidden layer with three neurons in that layer.

Table 5.4: Configuration of reflex inputs and equivalent controllers

Figure 5.18	**Reflex inputs**	**Equivalent controller**	**Controller parameters**
A	Position	P	K_p=10.5
B	Position, integral of position	PI	K_p=10.5, K_i=0
D	Position, speed	PD	K_p=125, K_d=30
C	Position, integral of position, speed	PID	K_p=125, K_i=0, K_d=30

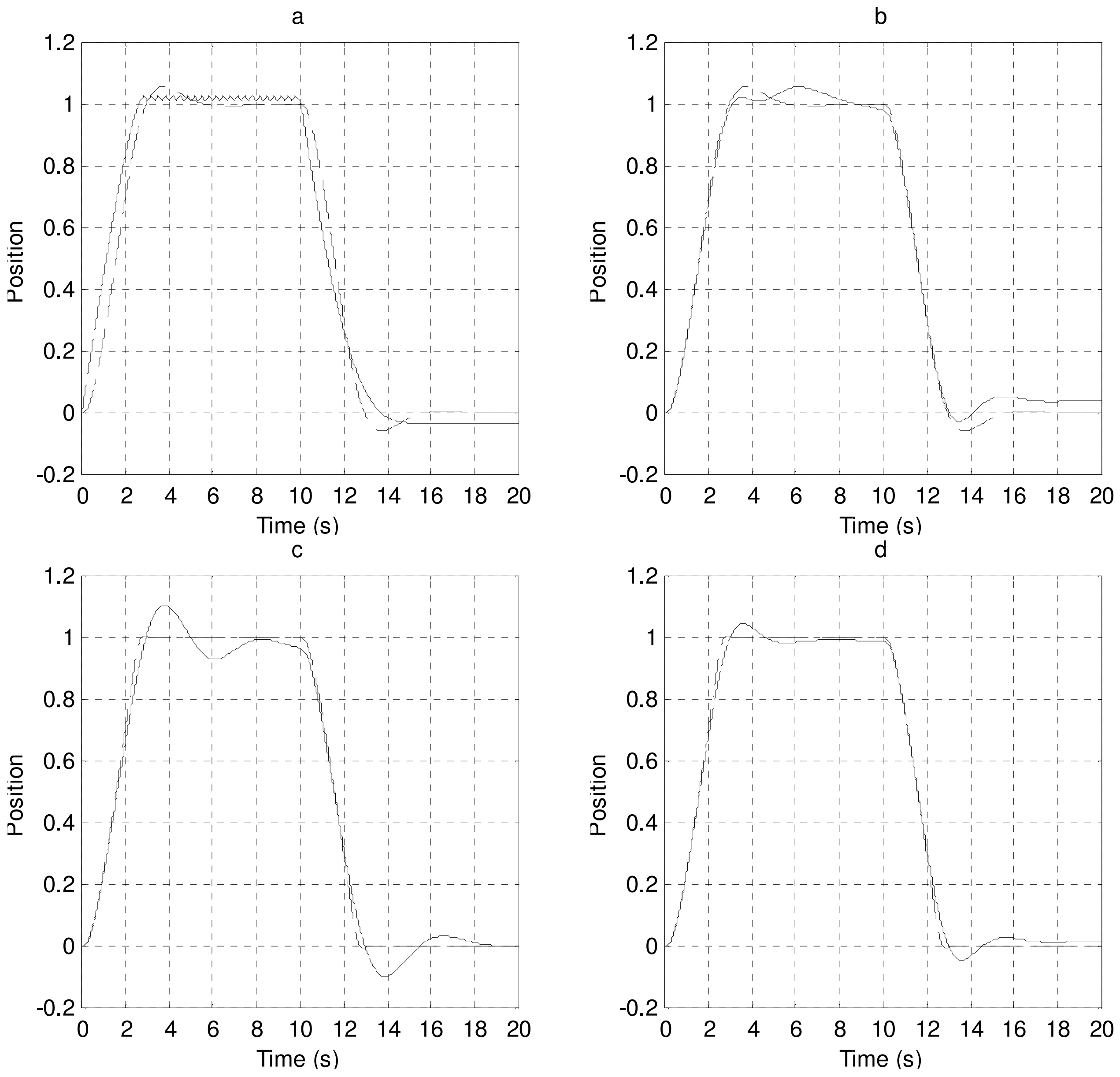


Figure 5.18: Comparison between best traditional controllers (dashed lines) and reflex controllers (solid lines). a) Reflex with position feedback vs. P controller. b) Reflex with position and integral of position feedback vs. PI controller. c) Reflex with position, integral of position and speed feedback vs. PID controller. d) Reflex with position and speed feedback vs. PD controller.

In all the graphs the traditional controllers out-perform the reflex networks in some way. The traditional controllers never have any offset while the reflex networks have varying amounts of offset, the best being the reflex with position and speed inputs. All networks had an average rise time which was similar to the traditional controllers, although the reflexes were slower in graphs c and d. Every reflex showed some overshoot, while only the traditional controllers with an absence of derivative component had any overshoot.

Another class of network was evolved which had only the error signal (difference between target and current position of the actuator) as an input. Now the reflex ANN takes the exact same functionality as the controller block of Figure 5.17. The actuator position as controlled by this network is shown in Figure 5.19 (as a dot-dashed line). The PD and P controllers are shown in the same figure as a dotted line and dashed line, respectively, for comparison.

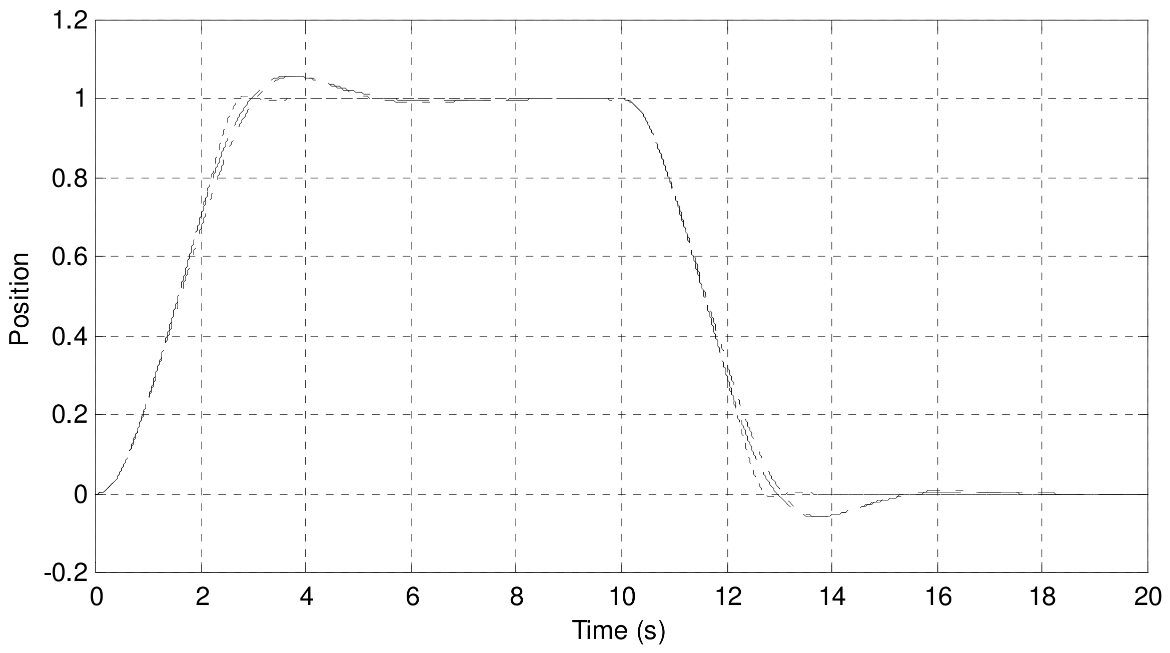


Figure 5.19: Comparison of reflex using only error input (dot-dash), PD controller (dotted) and P controller (dashed)

The position of the actuator under control of the reflex is nearly the same as when controlled by the P controller. Since this reflex had fully recurrent connections, it indicates that the recurrent connections are not being used to create integral and derivative values inside the network. If they were, the network should be capable of reducing the overshoot and rise time to something comparable with the PD controller.

Finally, the reflexes and traditional controllers created in this section will be compared in their ability to generalise across different actuators. Since the controllers have only been tested so far on slow D.C. motors, they will be used to control fast D.C. motors without changes to the controller constants. The reflexes will also be tested on the fast D.C. motors without being changed.

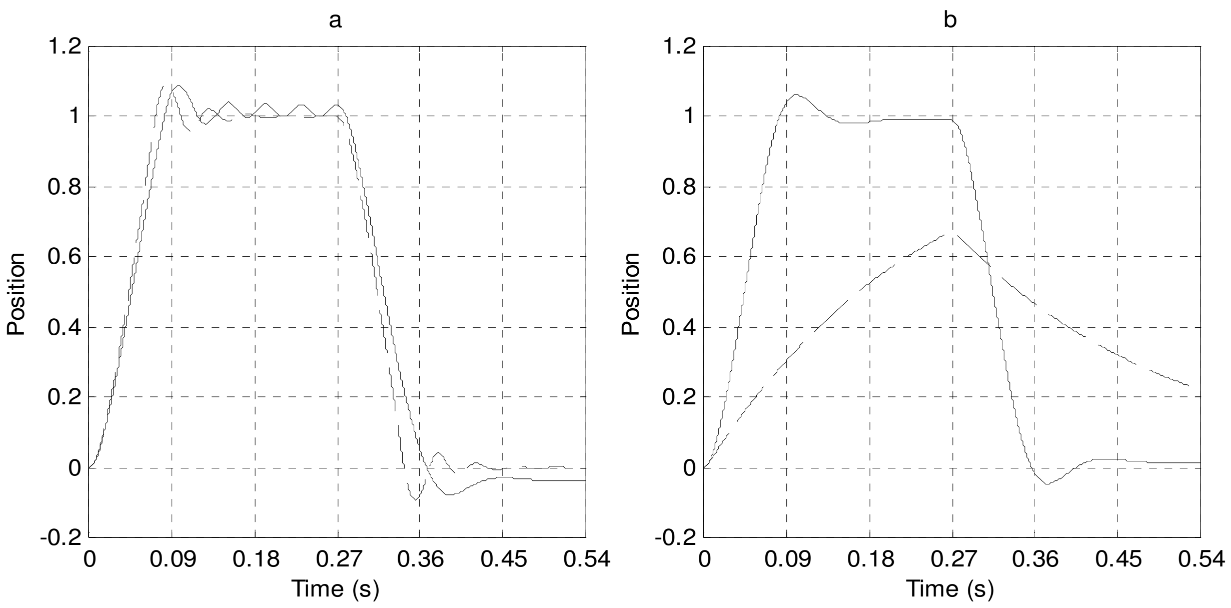


Figure 5.20: Reflexes and controllers controlling faster actuators. a) Solid line is reflex with position input, dashed line is P controller. b) Solid line is reflex with position and speed inputs, dashed line is PD controller.:

Figure 5.20 shows the comparison between reflexes and traditional controllers when applied to a faster D.C. motor than they were evolved or tuned for. Only the P and PD controllers have been shown (dashed lines in both diagrams), since the K_i constant was zero in both occasions during Figure 5.18. The reflexes shown (solid lines) have a position input for (a) and a position and speed input for (b). Figure 5.20 shows that when the traditional controller is simple (i.e. only has a P component) the performance has decreased (there is more oscillation on the actuator position) but is still marginally superior to the reflex network. However, the controller which controlled the slow D.C. motor the best (PD controller) now has severe problems. The reflex with the position and speed inputs is able to control the faster motor as well as it could the slower. This shows that the better reflexes are capable of generalising across different actuators of the same type, more than well tuned traditional controllers.

5.3. Discussion

5.3.1. Choice of EA and Parameters

Although the EA which was chosen as the best able to evolve a reflex ANN, it was not an exact ES as described in the original documentation. The strategy parameters (mutation and recombination rates) were not encoded in the chromosomes. The GA used was a real valued GA instead of using bit strings.

However, all the EAs tested were capable of evolving reflex networks with near optimum fitness scores. The ES was chosen since it almost always reached the near optimum values quite early during evolution and was less dependant on evolutionary parameters than the other two EAs tested. Chromosome initialisation (when not covering the entire range for optimum solutions) only affected EAs which had a low mutation rate. Elitism (or lack of it) was tested in the two different types of ES, and both were able to evolve a reflex to similar fitness scores.

The large mutation rates (opposite of the recommended values when using a GA) which still permitted good solutions to be found is due to the small effect of the mutations. In all EAs presented, the mutation was performed by changing a weight by a random number with normal probability (mean of 0 and deviation of 0.5). Since the random number for the most part would be quite small, a single mutation by itself would have little effect. This is the reason that large mutation rates were successful.

5.3.2. Fitness Function

When the networks were evolved which had fitness scores close to the optimum value, overshoot and eventually oscillations were present on the position of the actuator. This was similar in effect to increasing the proportional multiplier in a traditional PID controller to reduce rise time, and was due in part to the way that the rise time was being recorded. This made the rise time the largest part of the fitness score, and as such all networks were evolved mainly to minimise the rise time. This led to overshoot and oscillations because the actuator could not be stopped before passing the target position, and there was no explicit part of the fitness function which penalised oscillations.

Alternative methods for scoring the fitness were presented which aimed at removing the oscillations and were successful. Weightings for each part of the fitness score were added to show the effects of each component and to allow the fine-tuning of the desired response. The higher the weight the more importance that feature had, meaning the EA would attempt to optimise it (in this case, the optimum would be to reduce all the fitness components).

Allowing the time response to be fine-tuned by the weights of the fitness components is a similar task to tuning the PID constants in a traditional controller in order to achieve a specific performance target. One advantage of using this evolutionary system to create the reflexes is that the parts of the time response can be directly tuned, rather than changing them indirectly given the behaviour in Table 5.3. (However, this could be as easily applied to traditional PID controllers if, for example, they were evolved with fitness scores as measured for the reflexes here.)

5.3.3. Network Topology

In general, all the reflexes provided good control over the actuators and had a small size. Networks with fully recurrent connections could be evolved with the smallest size and still be able to control the actuators. ANNs which either had no recurrent connections (i.e. no sideways or backwards connections) or had only forward connections between adjacent layers could also control the actuators to the same levels of fitness, although in these cases the networks needed to be slightly larger.

The differences in sizes and numbers of connections could be important depending on how the reflexes were being implemented. For example, if a serial processing device was being used (in other words a typical computer microprocessor) it might be advantageous to reduce the number of computations required per forward pass of the networks. This would be important if the processor used was a simple microcontroller type processor, without much computational power. In that case, the reflex to use would be the best with no backward or sideways connections, since this has a low numbers of neurons (four neurons in one hidden layer, as reported in 5.2.4) and connections. If the reflex was to be implemented on a FPGA or ASIC (for example), where the amount of space used on the device should be minimised, the better reflex to use would be the fully recurrent network, as it has the least number of neurons. However, in the cases where the reflexes would be implemented on a powerful microprocessor (which are available in all desktop personal computers) or a specialised DSP chip, the best reflex to use would be the one which provides the best performance, as both these types of processor have plenty of computing power and storage space.

5.3.4. Actuators

The reflex networks were able to control both types of actuators (a linear actuator and a D.C. motor) tested here. Near optimum control over linear actuators was offered by the reflexes, although controlling a D.C. motor proved to be more of achieving the correct balance between the different fitness objectives to minimise it overall.

The generality of the networks was demonstrated by testing the evolved reflexes on different actuators. In the case of using networks evolved using linear actuators to control a D.C. motor, the control was poor with large oscillations, overshoot and offset. The opposite was true of reflexes evolved on D.C. motors controlling linear actuators; the rise time was long, but there was little offset and no overshoot.

Reflexes which were evolved using one type of actuator could control other actuators of that type (i.e. both fast and slow D.C. motors), and were independent of time. The speed of the actuator had no effect on the ability of the reflex to control it and neither did the sampling time at which network updates were performed. However, it would be obvious that increasing the sampling time so much that features of the motor parameters (i.e. position, speed, etc.) would be lost, would have an adverse affect on the performance of the reflexes.

5.3.5. Feedback

The reflexes were compared to traditional PID type controllers. The feedback paths from the actuators were chosen to correspond to the different types of controller for the comparisons. However, because the networks were being supplied with pre-calculated (measured from the actuators) position, speed and integral of speed (which would correspond to P, D and I components respectively), their task became more like acting as the controller constants. These values are normally calculated within the controller itself, which normally receives a single input (the error between reference and controlled values). However, the networks have some integrative and differential capacity through the recurrent connections.

When the evolved reflexes were compared to a set of traditional PID controllers tuned to control the D.C. motor, it was found that the reflexes did not perform as well as the controllers. Although the control asserted by the reflexes was good, there was typically more overshoot and offset than with the traditional controllers.

However, in the case of general control ability, the reflexes could be used to control other actuators of the same type but different speeds. When testing the traditional controllers using the same constants across different actuators, they performed poorly. This shows that although the reflexes have a slightly worse control ability than tuned PID controllers, they are more generally applicable as they require no changes to control different devices.

5.4. Conclusions

The EA which is best suited to this task (evolving an ANN which controls the position of an actuator) is an ES. It was able to evolve to a more optimum fitness score than either a standard EP or GA. The individual parameters of the ES (including mutation and recombination rates, population sizes) had little effect on the performance of the EA, except at the extreme limits to their settings. This is a benefit of the ANS, in that the individual ANNs used within it can be smaller and therefore easier to evolve (or train).

Choosing the fitness function is the most important aspect of evolving the reflex. Even subtle changes can have a serious affect on the result. This was demonstrated with the initial reflex evolved by using an ES. A seemingly acceptable fitness function (addition of offset, overshoot and rise time) ended up with oscillatory control. Only a minor modification was required (using accumulative offset with a higher weighting) to achieve stability.

Networks with recurrent connections and a low number of neurons were capable of controlling simulated actuators to a high degree of accuracy. The offset and overshoot were typically less than 1% and 2% respectively (for the best evolved reflexes). Networks which either had no recurrent connections or only forward connections to the adjacent layer were also capable of controlling the actuators, although those reflexes required more neurons to reach the same accuracy as the ANNs with recurrent connections.

The evolved reflexes were capable of controlling different types of actuators (both fast and slow linear actuators and D.C. motors). The reflexes were time invariant, with networks evolved on a slow actuator capable of controlling the fast actuator to the same level, and vice versa.

The reflexes had similar performances to traditional controllers, in their ability to control the actuators. However, the reflexes could be used on different actuators without requiring any changes, whereas the traditional controllers would need to be re-tuned. The ANN reflexes are capable of generalising better than PID controllers, which are set up with specific constants for a certain actuator. When the motor parameters are changed (as was the case here) the PID controllers do not work as well. This is shown to varying degrees in Figure 5.28 where the P controller is only slightly degraded but the PD controller is badly affected. The derivative component of the PD controller (which is related to the action of the motor through time) is affected by the change from a slower motor to a faster one.

5.5. Summary

This chapter has described the basic experimental set-up for evolving an ANN to act as a reflex, controlling the position of an actuator. Three different EAs have been tested for their ability to evolve such a reflex. Other parameters which would affect the performance of the reflex, such as evolutionary parameters, network topology and inputs, actuator type and feedback have all been explored. Results of the performance of the reflexes at each stage have been presented, and the discussion sections have explained the reasons for the results obtained.

The next chapter will proceed to the layer above the reflexes – the action layer. While this is essentially a separate layer, the reflexes evolved here will be used during evolution and testing of the actions.

Chapter 6. Action Layer Theory

6.1. Introduction

The previous chapter described the creation and operation of the lowest layer of the ANS (the reflexes). This chapter discusses the action layer, which is built upon the functions provided by the modules in the reflex layer. Within the action layer, a number of parallel modules exist, each module controlling the reflexes to perform a single action. The actions create automatic, rhythmic patterns of activity, and are used for repetitive tasks such as walking, swimming and breathing. The name given in biology to the fundamental neural circuitry which provides the basis for all of these functions is Central Pattern Generators (CPGs).

This chapter will initially describe the biological background to CPGs. The history of the discovery and functional overview of CPGs will be given. Specific details about the neural components (including neurons, synapses and network configuration) is presented. Following that, the implementation details of the artificial CPGs is explained. Again, as with the reflexes, these were implemented in this project using EAs to create ANNs. The explanation will include neuron functionality, network topology, design method and evolutionary setup. An overview of the experimental approach is presented, including the simulations used for the CPG testbed. Finally, a summary will review the main points from this chapter.

6.2. Biological Central Pattern Generators

6.2.1. Introduction – History and Discovery of CPGs

Theories to explain nervous system control over locomotion were put forward as early as the start of the twentieth century. The two main proposals were that locomotion (most early observations and experiments were performed on legged vertebrates) was controlled either centrally (in the central nervous system (CNS)) or peripherally (initiated and controlled by sensory feedback from the limbs). The main theory in the peripheral group was that of the chain reflex. This involved reflexes for all the required muscles to be serially connected so that they co-ordinated the limbs in sequence to create the correct locomotion pattern. Figure 6.1a shows the functional setup of a chain reflex. The central pattern generator theory states that there is a neural circuit in the CNS, which produces patterned outputs to control the reflexes simultaneously and therefore the locomotion patterns. The organisation of this is shown in Figure 6.1b. An overview of these different mechanisms is given in [Shik 1976] and [Grillner 1981].

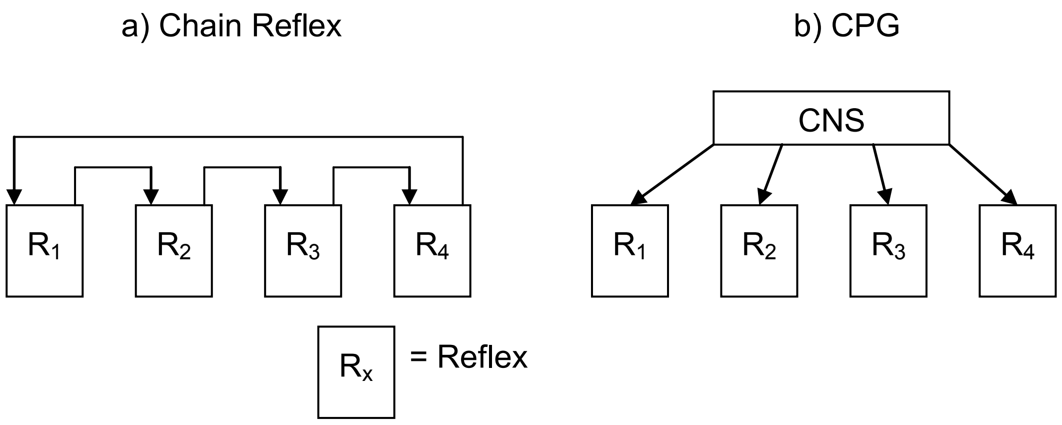


Figure 6.1: The functional layout of chain reflexes and central pattern generators

Shik et. al. demonstrated the existence of dedicated rhythmic neural circuits by showing that simple electrical stimulus of specific areas of the brain could also cause locomotion [Shik 1966]. It was found that increasing the level of stimulus increased the speed of locomotion and also selected the appropriate gait. This showed that dedicated neural circuits existed, since simple high level control signals were able to produce more complex patterns of activity, although the neural circuits for selecting or transferring between gaits was not known (but was a lower level function).

The neural circuits for generating rhythmic patterns (for locomotion) were shown to exist in the spinal cord of fish by Grillner [Grillner 1974] and Wallen [Wallen 1987]. This was discovered by making the fish spinal – dissecting the nerves connecting the spinal cord to the brain – so only the spinal cord had control of the body (and not the higher centres of the nervous system). Locomotion could then be evoked by chemical stimulus of the spinal cord. This type of locomotion is termed fictive since it is initiated by factors external to the animal.

Experiments similar to those on fish by Grillner (above) were performed on spinal cats [Grillner 1981] (for an overview). By supporting the cat and standing it on a moving treadmill, the cat was able to demonstrate gaits suitably related to the speed of the treadmill (without any stimulation to any nerves). This showed that the action of CPGs can also be initiated by peripheral feedback as well as signals from higher centres of the nervous system (although since the animal was spinal, influences from the higher centres which may override the peripheral feedback were not present).

Cutting the afferent connections from the limbs to the spinal cord was a way of showing whether the chain reflexes or CPGs were mainly responsible for locomotion. Initial experiments removed only a few of the connections. For example, deafferent hind limbs in a spinal cat would still take part in locomotion, when the fore limbs were on the moving treadmill. Stimulating the nervous system of a fully deafferented animal caused locomotion (as per the discovery by Shik et. al.), although the gait was not robust and was prone to breaking down. This proved that the main nervous circuit for controlling locomotion was a central pattern generator in the spinal cord, although feedback from the limbs (such as that on which the chain reflex theory was based) would enhance the performance of the gaits (a point which is described in section 6.2.2).

Such oscillating circuits have been found in many invertebrates, and some have been completely identified and mapped. Examples of these include CPGs for swimming in sea slugs and leeches, heartbeat CPGs in leech and lobster and stomatogastric rhythms of lobsters [Friesen 1978]. Supporting evidence for central pattern generators has been demonstrated in many types of fish, quadrupeds and newborn human babies [Carpenter 1990], although the actual neural circuitry has not been mapped [Stewart 1998].

6.2.2. Functions of CPGs

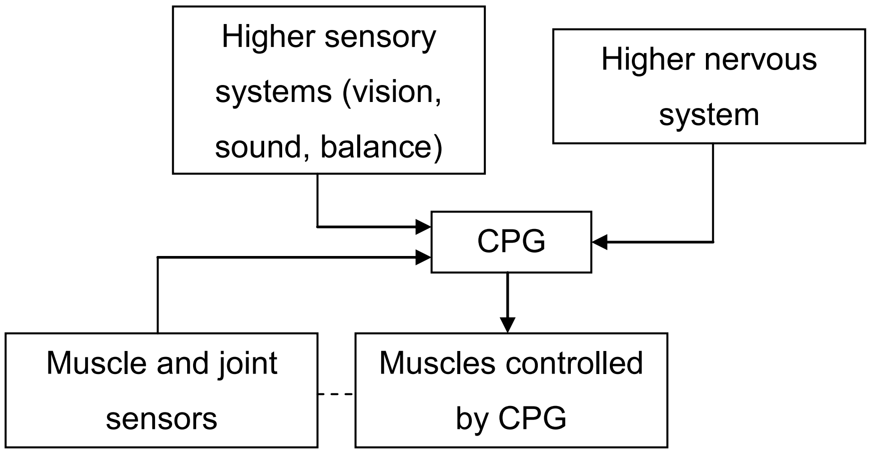
CPGs are responsible for producing the automatic, rhythmic patterns of activity in an animal. Depending on the type and function of the CPG, they can be involuntary (completely autonomous), voluntary (under control of the higher centres of the brain), or initiated by some other means (such as the reflexive response referenced below). Examples of voluntary and involuntary CPGs include heartbeat timing, locomotion (swimming, walking, running, etc.) and scratching. However, there is no reason why a CPG cannot be activated by more than one of these methods. For example, locomotion may be voluntary (for goal directed movements) or as a reflexive response (escape from noxious stimulus as in the escape swim CPG in leeches [Levitan 1997a]).

In the cases of voluntary and reflexive CPG activation, there must at least be some simple method for turning the CPG on and off. There may also be more complex connections to, from and between CPGs which enables more interactions between them.

Since the CPG networks produce only the fundamental oscillatory pattern of activity for a specific task, inputs to the networks (further to the on/off signal) may be required for complete or more useful operation. For example, most locomotion CPGs in vertebrates have feedback from a number of sensors, including muscle or joint position and sensors on the body for sensing the surrounding environment. This is shown in [Ijspeert 1998] for the lamprey swimming CPG. As a result of the presence of feedback from sensors on the skin, the lamprey can successfully swim in the desired direction through a water current. In the absence of this feedback the lamprey would simply swim without any knowledge of the fact that it was being pushed off course. Similar feedback mechanisms exist in stepping locomotion of legged animals [Grillner 1985]. These enable the animal to know where its limbs are positioned and can therefore adapt the stepping patterns to perform more efficient locomotion patterns.

This is an important point for future study in the ANS model. As mentioned above, only the fundamental patterns of activity are being sought from the CPGs. In the case of locomotion, the Artificial CPGs created here are capable of producing the appropriate gait patterns, but would be unable to navigate over rough terrain. These additional abilities of biological CPGs are due to the extra connections from other areas of the central and peripheral nervous systems. These influences are shown in Figure 6.2.

The outputs from a CPG either connect to the motor neurons of the muscles that are controlled during the rhythmic motion, or connect to groups of interneurons which in turn control the motor neurons. In higher vertebrates, the outputs from the CPGs are also sent to the brain and brainstem which assists in the learning and higher level control of motor programs [Kandel 1991].


Higher sensory systems (vision, sound, balance)
Higher nervous system
CPG
Muscle and joint sensors
Muscles controlled by CPG
Figure 6.2: Additional influences which extend the capabilities of CPGs

6.2.3. Biological CPG components (Neurons Revisited and Synapses)

There are three main functional components with important properties, which can be combined to produce central pattern generators [Getting 1988]. These are neurons (the processing units), synapses (connections between units) and networks (how they are all connected together).

The basic operation of a typical biological neuron was explained in Chapter 2. Cell specialisation and different species of ion within the neurons mean the currents flowing in and out can vary. Importantly, these differences in neurons give rise to changes in the time response which are important for rhythmic pattern generation [Getting 1988]. Unit functionality is an area which will be studied in future work.

Figure 6.3 shows some of the temporal properties of an action potential which are important in the generation of rhythmic activity in a CPG neural network. The following sections will discuss the features of the neurons and synapses which contribute to these temporal properties.

The first point (t_1) is the time elapsed between the onset of inputs to the neuron and the beginning of the positive going output spike from the neuron. This is controlled by a number of factors. The first is the firing threshold of the neuron. As the inputs charge the membrane potential, time passes until the threshold is reached, at which point the neuron completely depolarises. Higher thresholds take longer to reach, and so changing the threshold changes the length of t_1. The rate at which the internal activation increases (when in the resting state) depends on the inputs to the neuron and these inputs depend on the pre-synaptic neurons and the synapses which connect to them.

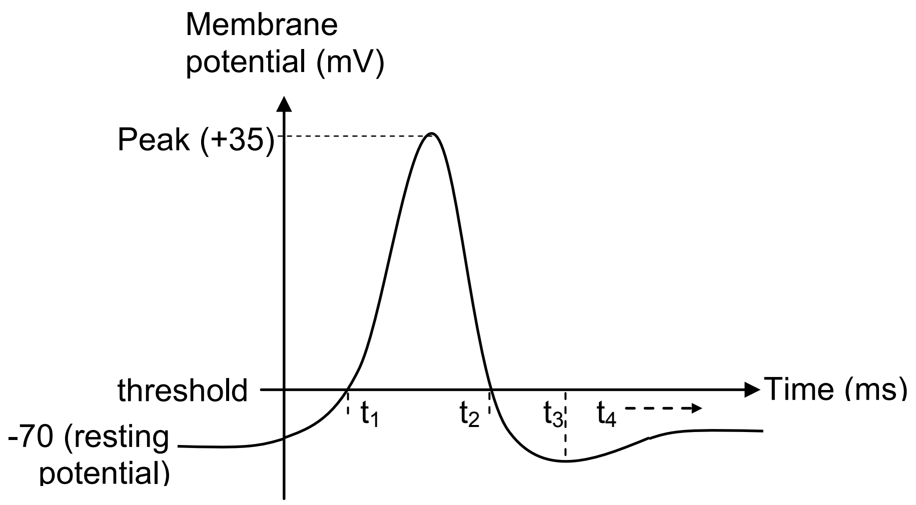


Figure 6.3: Significant times in the firing of an isolated neuron which assist in the creation rhythmic activity in networks

The main output pulse lasts from t_1 to t_2. The amplitude of this pulse is always constant (since the neuron will always fully depolarise). The t_1 to t_2 period is mainly controlled by the ionic current flow across the cell membrane, which is the cause of the depolarisation and repolarisation. However, it can be modified by the synaptic connections at the inputs to the neuron. Different neurotransmitters have different effects on the ion channels in the cell membrane, and can change the length of time for which the neuron produces an output [Brodal 1992]. Similarly, certain neurotransmitters can cause extended periods of inhibition. These are termed slow EPSPs (Excitatory Post Synaptic Potentials) and slow IPSPs (Inhibitory Post Synaptic Potentials) [Brodal 1992].

Finally, t_3 to t_4 represents the time period in which the neuron cannot produce another pulse (as it is hyperpolarized). The hyperpolarized time is a function solely of the neuron membrane properties. “Refractory period” is an expression used for the rest time of neurons. The time t_4 is effectively the time until the onset of the next output pulse, and could therefore incorporate the times t_3 and t_1 (of the next pulse). It could also include some special properties of neurons. The first of these is the adaption of the neuron to the input spikes. Adaption is the process of reducing output activity in the face of continued inputs, so as the length of time during which the inputs are active increases, the frequency of output pulses of the neuron will gradually reduce. Another neuron property which would affect the time

t_4 is that of post-inhibitory rebound. Sometimes, after a neuron has been hyperpolarized, and as the membrane potential decays back to the rest potential, the neuron can be more excitable than normal. The effect of returning from hyperpolarisation can sometimes be enough to trigger another output spike spontaneously. This simple reverberating type of neural circuit has been shown to control the swimming of Clione (a sea mollusc) [Levitan 1997b].

Due to the wide variety of types and properties of neurons and synapses, there is no single method of connecting them together to produce a central pattern generator. Some common, simple examples are given in Figure 6.4 [Getting 1988].

The temporal properties of these networks are similar to those mentioned above, but obviously operate at a higher level, now being the result of overall network activity, rather than that of a single neuron. The time t_a is the length of time for a burst of neuron pulses. T_b is the relative offset between connected neurons, and is a more controllable time (the times which can be used to construct t_b are controllable) than using post-inhibitory rebound to fire anti-phase bursts (as shown by (b) and (c) in Figure 6.4). T_c is the time between successive outputs from a single neuron, which can be comprised of the time t_4 from Figure 6.3 and inhibition of the neuron by others in the network.

These three attributes will be explored in detail in the following sections.

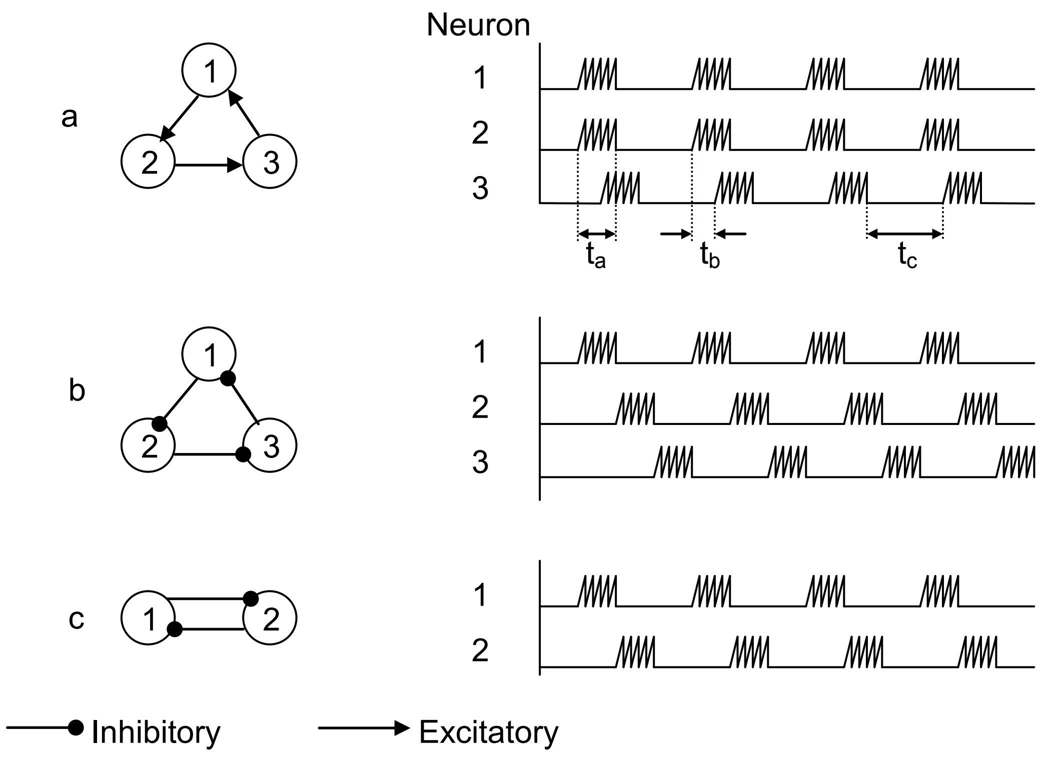


Figure 6.4: Examples of common CPG network models and the activity of the neurons. a) Cyclic excitation b) Cyclic inhibition c) Recurrent inhibition

6.3. Action Layer Central Pattern Generators

6.3.1. Basic Requirements

The Artificial CPG should be capable of producing signals with timings similar to those shown in Figure 6.3 and Figure 6.4, which will allow it to produce a wide variety of rhythmic patterns.

6.3.2. Neuron model

A new neuron model has been developed specifically to simulate the timings required from the CPG (as demonstrated in Figure 6.3 and Figure 6.4). Additional objectives behind designing this neuron model were to make the creation of ANNs for CPGs easier and the operation of the ANNs less computationally expensive. It is based on the functional time response properties of biological neurons, although it is not meant to model their exact behaviour.

The neuron is initialised with its internal activation (membrane potential) at a resting level. This behaves similarly to biological and leaky integrator neurons. The new internal activation of the neuron is calculated as the sum of weighted inputs added to the current internal activation which is multiplied by a constant decay ratio, as shown in Equation 6.1.

$$mp(t+\Delta t) = k \cdot mp(t) + \sum_{j=0}^{n} i_j \times w_j \qquad \text{Equation 6.1}$$

where $mp(t)$ is the internal activation (membrane potential) of the neuron, k is the (constant) decay rate of the internal activation, $i_j \times w_j$ is the weighted input from neuron j (including possible delays due to the synapse that connects the neurons) and n is the number of inputs.

Since the internal activation of the neuron must charge up to a threshold before firing, it allows the times t_1 and t_b (from Figure 6.3 and Figure 6.4) to be created by the neurons.

If the neuron reaches its firing threshold, it produces an output (of one) for a set period of time. At all other times, the output of the neuron is zero. This on period can be modified by one of the synapse properties, as explained in the following section. After firing, the internal activation is set to a level below the rest level (to act as though the neuron is hyperpolarised) and cannot produce output for a fixed time. These can be used to produce the times t_2 and t_3 from Figure 6.3 and therefore contribute to times t_b and t_c from Figure 6.4.

$$\mathit{if}\,(mp_j(t) \geq th_j) \Rightarrow out_j(t) = 1 \qquad (t_1 \leq t \leq t_2) \qquad \text{Equation 6.2}$$

$$mp_j(t) = hy_j \Rightarrow out_j(t) = 0 \qquad (t_2 \leq t \leq t_3) \qquad \text{Equation 6.3}$$

$$\mathit{else} \quad out_j(t) = 0$$

where th_j is the threshold for neuron j, out_j is the output from that neuron and hy_j is the hyperpolarisation level. All the times (t_x) are equivalent to those shown in Figure 6.3.

The time t_4 can be constructed from a combination of the hyperpolarisation time, a time where the neuron receives no inputs and the charge time (which depends on the activation of weighted inputs applied to the neuron). Although post-inhibitory rebound (described previously) is a verified biological mechanism for creating rhythmic activity, that ability has not been included in the neuron model presented here. The reason for this is because the rebound is a side-effect of the way neurons work, and although it can be used, it is not something which is directly controllable (i.e. you control the production of output pulses by supplying a neuron with inputs).

This neuron model has been chosen over more usual neuron models (such as a McCulloch-Pitts with sigmoid transfer function, or leaky integrator) because it produces outputs in the time domain rather than encoding the mean firing frequency of a neuron. When the outputs are working in the time domain it is easier to tune and specify the times of the neuron pulses to produce the required patterns of activity. This would not work at all if it did not have time domain behaviour.

6.3.3. Synapse model

The synapse model used in the artificial CPG networks has been designed to include features which make it more suitable for simple implementation of time dependant parameters. As with standard models of synapses in ANNs, these have a *weight* which represents the connection strength (with negative being inhibitory, zero meaning no connection and positive being excitatory). This will help to construct the times t_1 and t_b from Figure 6.3 and Figure 6.4.

Each synapse also has a *delay*, which is simply the time taken for the output of a neuron to travel along the synapse and be presented as an input to the post-synaptic neuron. This aids in the creation of the time t_b shown in Figure 6.4. The addition of this parameter should allow the network to evolve more easily, as the neuron parameters need not be the only place where this time can originate.

The final parameter of each synapse is the *type*, which performs the same function as the type of neurotransmitter in biological synapse, namely to modify the output of the post synaptic neuron. Since the main concern of the ANN here is to produce correctly timed activity patterns, the type of the synapse will signify the pulse time of the neuron output. Initially there will be two types of synapse, one which causes a short (or normal) length of neuron output and another which causes a longer output duration. This parameter can be extended up to any number of types, ultimately reaching the stage where the type itself explicitly specifies the duration of neuron output.

Using long and short pulses is simply an initial step. Plans for extending the system have been described above, but it is simpler to start from a limited number of pulse sizes. The reason for choosing longer and shorter pulses are that two different lengths would be capable of producing more variation in the patterns of activity than two lengths which were very similar.

If a neuron fires and a synapse with a type which specifies a long pulse has an active input to that neuron (i.e. non-zero), the neuron will output a pulse for a longer length of time. The neuron actually produces an output for the time specified by the active synapse with a type specifying the longest duration. However, the priorities of the types could be changed so that it is not always the longest time that is used. There is a biological equivalent which is the neurotransmitter responsible for slow EPSPs (Excitatory Post-Synaptic Potentials), as previously mentioned. The benefit of this parameter in the artificial model is that it allows control over the time for which the neuron produces an output, without having to include computationally expensive, precise models of real neurons.

6.3.4. Network topology

To make the CPGs as compact and self-contained as possible, the interface to the higher and lower layers will be minimised. There will be a single input, although it can be applied to any of the neurons in the network. The input will be tonic (i.e. a constant value) and is simply to provide an on/off signal and stimulus to the network (in the case of the on signal). Although biologically realistic inputs would be pulses, the CPG created here should not have to rely on any patterns in the inputs.

A rough guideline to the required network sizes can be extracted from literature. It is suggested in [Grillner 1985] and [Golubitsky 1998] that only twice as many processing units as controlled joints are required (although other groups of neurons may operate behind these). Neural circuits identified in the Clione use two neurons to control the flapping of one wing [Levitan 1997b]. By initially following this guideline, it is predicted that there will only be four neurons in the biped and eight in the quadruped. With such a low number of neurons it should be easier to analyse the operation of the network, even though the neurons used produce pulse outputs.

Connections in the network can be to and from any neuron; there is no imposed layered structure to the network. There is no difference between this topology and a fully recurrent layered network apart from the interpretation of the layout.

There will only be as many outputs as there are legs to control on the robot. Each output is taken from an unique neuron in the network and then used as an input to a single reflex. The linkage between CPGs and reflexes is described fully later, in section 6.4.1.

6.3.5. Evolutionary Algorithm parameters

The artificial CPG networks will again be created using EAs, specifically an ES since it consistently evolved reflexes with better fitness scores than the other EAs tested. In reality the EA is not a canonical ES as defined in [Schwefel 1995], but it uses the features of an ES more than the features of other EAs. For example, there is no distinction of a main evolutionary operator (mutation or recombination) as there is with an EP or GA. Real numbers are used in the genes rather than bit strings as they are more applicable to the problem and deterministic selection is preferred to probabilistic selection. It is these features which make the EA more generally useful, and they just so happen to be features of an ES.

The genes will be grouped in the same way as those used for the reflex chromosomes (explained in Chapter 4). All the connections to the outputs of a single neuron are adjacent in the chromosome, in the order of which neuron they connect to. Each gene in each chromosome must also encode the three parameters for the synapses; weight, delay and type. Since every possible synapse in the network can be connected, and every synapse has all three components, the genes will be fully encoded, as shown in Figure 6.5.

The *weight* is a real valued number representing the synapse strength as explained in section 6.3.3. The *delay* is a non-negative real number which specifies the time delay of the synapse. The *type* encodes the type of output pulse induced by the synapse when active.

All the genes are initialised with the delay set to zero (the delay was not used in the initial experiments). The weights are initialised to uniformly distributed random values between selectable positive and negative limits. The type of each synapse is chosen randomly with a probability of 0.5 for both types (long or short pulses).

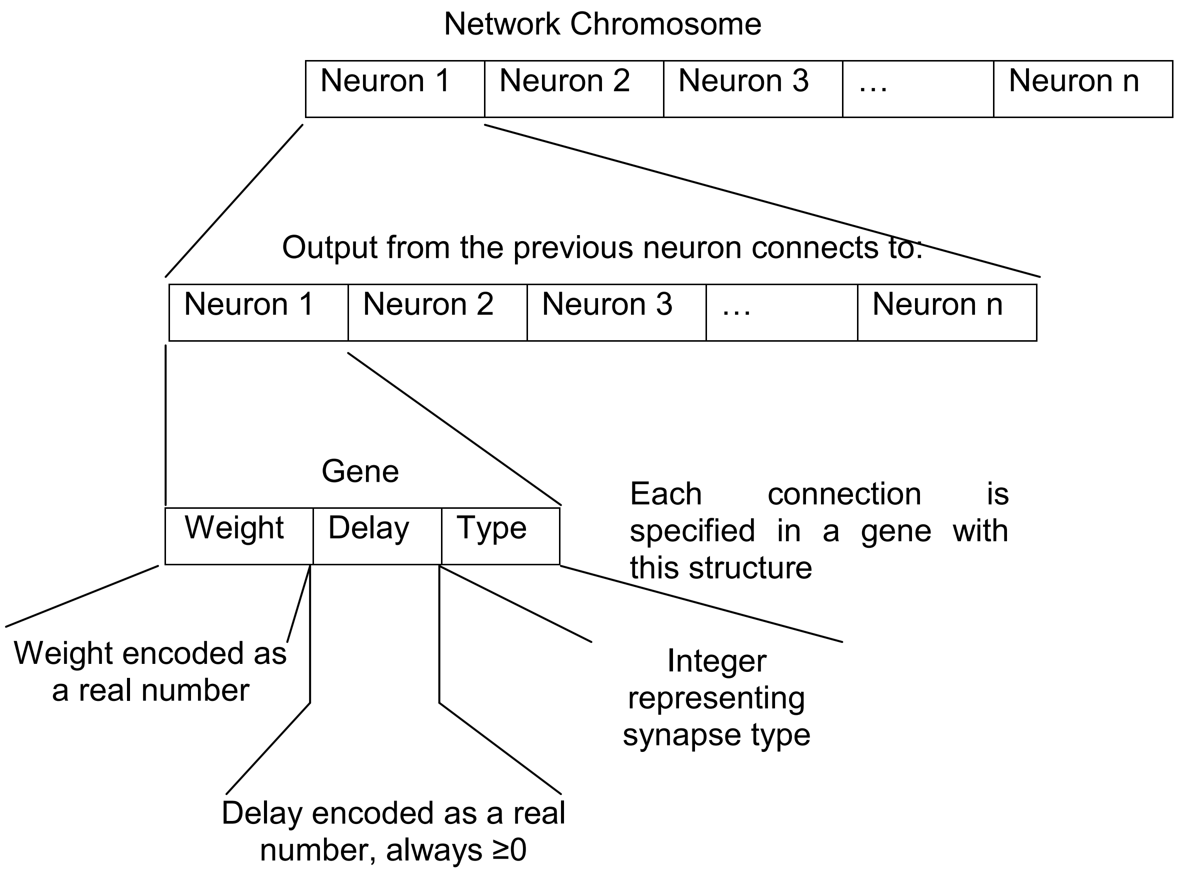

Network Chromosome
Neuron 1
Neuron 2
Neuron 3
…
Neuron n
Output from the previous neuron connects to:
Neuron 1
Neuron 2
Neuron 3
…
Neuron n
Gene
Weight
Delay
Type
Each connection is specified in a gene with this structure
Weight encoded as a real number
Integer representing synapse type
Delay encoded as a real number, always ≥0

Figure 6.5: Encoding of genes in CPG chromosomes. Top – arrangement of genes. Bottom – structure of genes.

Recombination operates by swapping genes between chromosomes, with the entire gene being used, not only the component parts (i.e. weight, delay and type taken as a unit). It operates as a dual discrete crossover [Baeck 1995], with a fixed probability during evolution. Recombination is applied to all offspring, but the recombination rate is the probability that a gene would be swapped between two chromosomes (on a per gene basis due to the discrete crossover being used). Mutation is also performed with a fixed probability during evolution although when it is to be performed, a further probability distribution selects which component of the gene should be mutated (weight, delay or type). The weight and delay are mutated by adding a normally distributed random number (with mean of zero), although the delay has a low limit of zero. The type is randomly selected from all the available types with a uniform probability.

The fitness evaluation will be explained in the next section, which also explains the tasks for which the CPGs will be evolved. The ES terminates after a specified number of generations rather than when a certain fitness level is reached. This is because there is no guarantee that a particular configuration of the combination of EA, network and neuron properties will converge to any working solution and there needs to be some way to stop evolution in that case. By using a fixed number of generations it is also possible to compare how successful the different system setups were, as they will all have a common reference.

6.4. Experimental Setup

6.4.1. Robotic Simulation

For the same reasons that the actuators were simulated during evolution of the reflexes (time taken and preservation of equipment), the robots used for testing the evolved CPGs will also be simulated. Once the CPGs have been evolved successfully using the simulation, their operation can be verified on the real robot that the simulation is based on (and this has been successfully done using the results on the following pages). The operation of the real robot is shown in Figure 7.14.

The simulated robots will be assumed to be dynamically stable in all directions, meaning that the robots will be completely stable at all times and cannot topple over. This is because we are only interested in producing the fundamental gait patterns at this stage. This can be thought of as the same as the experiments described earlier (on the legged animals, using treadmills) which supported the existence of central pattern generators. The simulated robot in these experiments can be thought of as being supported over the ground while the nervous system is stimulated to evaluate the basic gait patterns.

The actual output devices of the simulated robot will be the actuators described in the previous two chapters. One of these will be used per leg (i.e. two for the biped, four for the quadruped), and the position of the actuator will be used as the position of the leg. These actuators will be controlled by the artificial reflexes which were evolved in the previous chapter. The input to the reflexes for target position will come from the outputs of the CPG. Since the CPG neurons produce pulse outputs in the time domain (with a value of zero or one) and the reflexes require continuous input values, there must be an extra stage of conversion added to the chain. This is achieved by adding a leaky integrator (see Equation 6.4) between the output of the CPG and the input of the reflex, which has the effect of converting from discrete pulses to an average firing frequency (suitable for the reflex inputs). Output pulses from the CPGs are integrated (to increase the average firing frequency and therefore the output of the integrator) and in the absence of CPG outputs the average firing frequency is decayed.

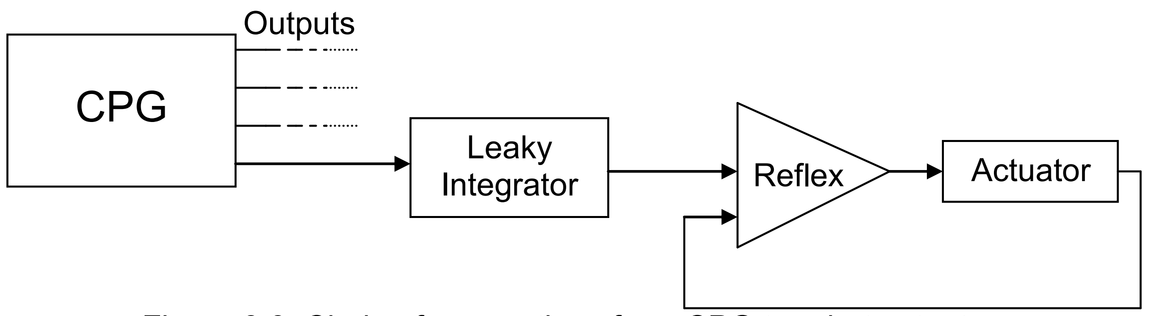


Figure 6.6: Chain of connections from CPG to robot actuator

The leaky integrator used to link the CPG outputs to the reflex inputs was described by the following equation:

$$y^{+} = k \times y + x \qquad \text{Equation 6.4}$$

where *y* is the current output of the integrator, *k* is a constant between 0 and 1 used to decay the current value of the integration, *x* is the current input to the integrator and y^{+} is the next output.

The tasks for which artificial CPGs will be evolved are biped and quadruped locomotion. The exact task and robot simulation will be detailed in the following sections.

6.4.2. Biped locomotion

The first task for which artificial CPGs will be evolved is the generation of activity patterns for bipedal locomotion, effectively the creation of an alternate stepping pattern for the hip positions.

The simulated robot is shown in Figure 6.7. For each leg there is one active degree of freedom – the revolute joint at the hip – and one passive degree of freedom (at the knee). The knee can only bend forward, and locks when bent backwards. This was based on the construction of a simple robot available within the department (also shown in Figure 6.9). As mentioned earlier, the simulation which will be used to evolve the CPGs is based on this robot, which can then be used to verify the correct operation of the CPGs.

The robot body has been allocated a co-ordinate system given by its x,y position and the angle at which it is facing.

The position measured at the actuator when the leg is on the ground is between 0.2 and 0.8 (with 0.2 being towards the rear of the robot), as shown in Figure 6.7. While a leg is on the ground (and moving backwards therefore locking the knee) it can be used to propel the body of the robot forwards. The equations used to update the legs and robot position are explained in Appendix E.

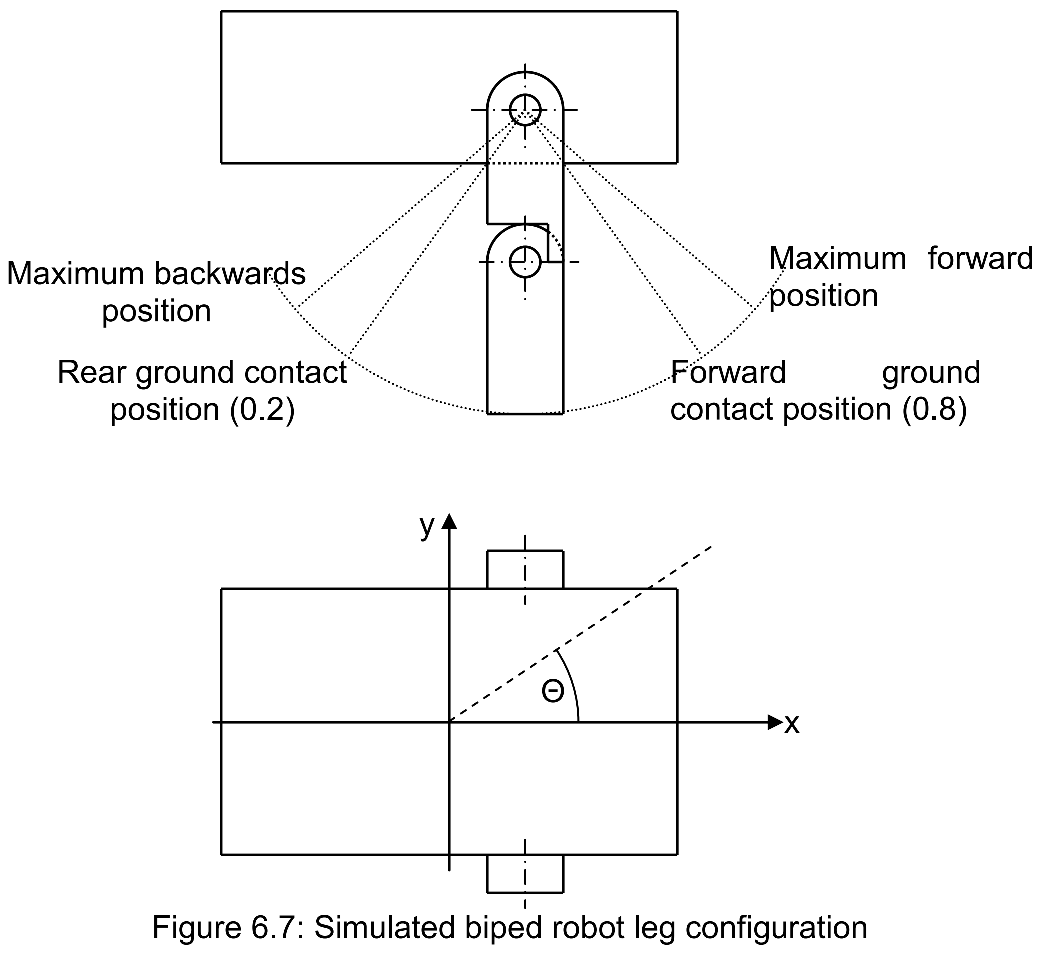

Maximum backwards position
Maximum forward position
Rear ground contact position (0.2)
Forward ground contact position (0.8)
y
x
Θ

Figure 6.7: Simulated biped robot leg configuration

The fitness function used to evaluate the performance of the CPG in controlling the simulated robot was simply the distance the robot could walk (the robot was only allowed to move when the legs were moving in opposite directions) from an initial position in a fixed period of time.

6.4.3. Quadruped gaits

The second task which the CPGs were evolved for was the production of primary gait patterns for a quadruped [Golubitsky 1998]. Since these gaits are the fundamental pattern again, it is the general limb co-ordination pattern that is being sought. The primary patterns are listed below and shown in Figure 6.8 [Gray 1968a]:

- Walk (each leg moves on its own phase)
- Pace (pairs of legs at the same side of animal move in the same phase)
- Trot (diagonal pairs move in same phase)
- Gallop (front and rear pairs move in same phase)
- Pronk (all four legs move in one phase)

□ Swing phase (leg off ground and moving forwards)

■ Stance or power phase (leg on ground, pushing backwards)

LF = Left Front leg, RF = Right Front leg, LH = Left Hind leg, RH = Right Hind leg

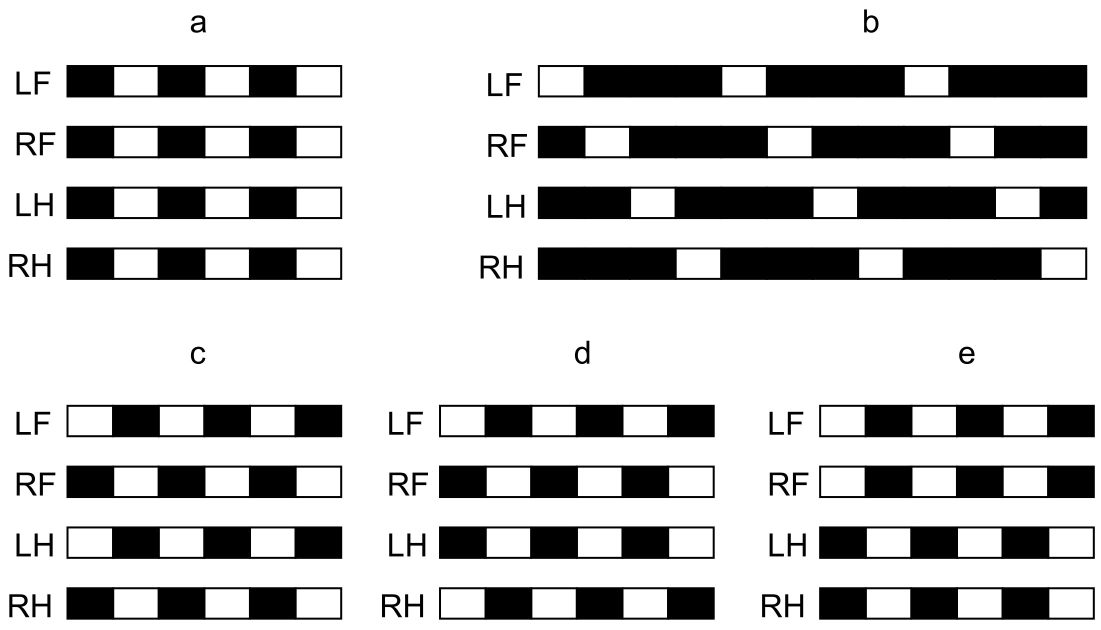


Figure 6.8: Stepping patterns of various quadruped gaits. a) Pronk. b) Walk. c) Pace. d) Trot. e) Gallop.

Gaits used by quadrupeds in nature can be used for different purposes [Gray 1968b] and some are preferred at different speeds [Grillner 1981]. For example, walking is a slow but very stable gait. Pacing and trotting are medium speed gaits, which are used depending on the physical structure of the animal; these are used as an efficient compromise between speed and energy usage (e.g. migration). The fastest gait is the gallop, and has obvious advantages for predator avoidance or prey capture. Finally, the pronk gait is a simple hopping gait. The patterns for all of these gaits are shown in Figure 6.8.

The simulated robot is similar to the biped (in fact, it is two biped units joined together). The legs work in the same way, and use the same equations to update the position of the robot (except n=3 for the quadruped). This is shown in Figure 6.9, along with the real robot on which it was based.

The CPGs have been evolved here to create gait patterns for the same tasks mentioned above. One gait is slow but stable (walk), one is geared for efficiency (trot or pace), another for speed (gallop) and the final gait is evolved for simplicity (pronk).

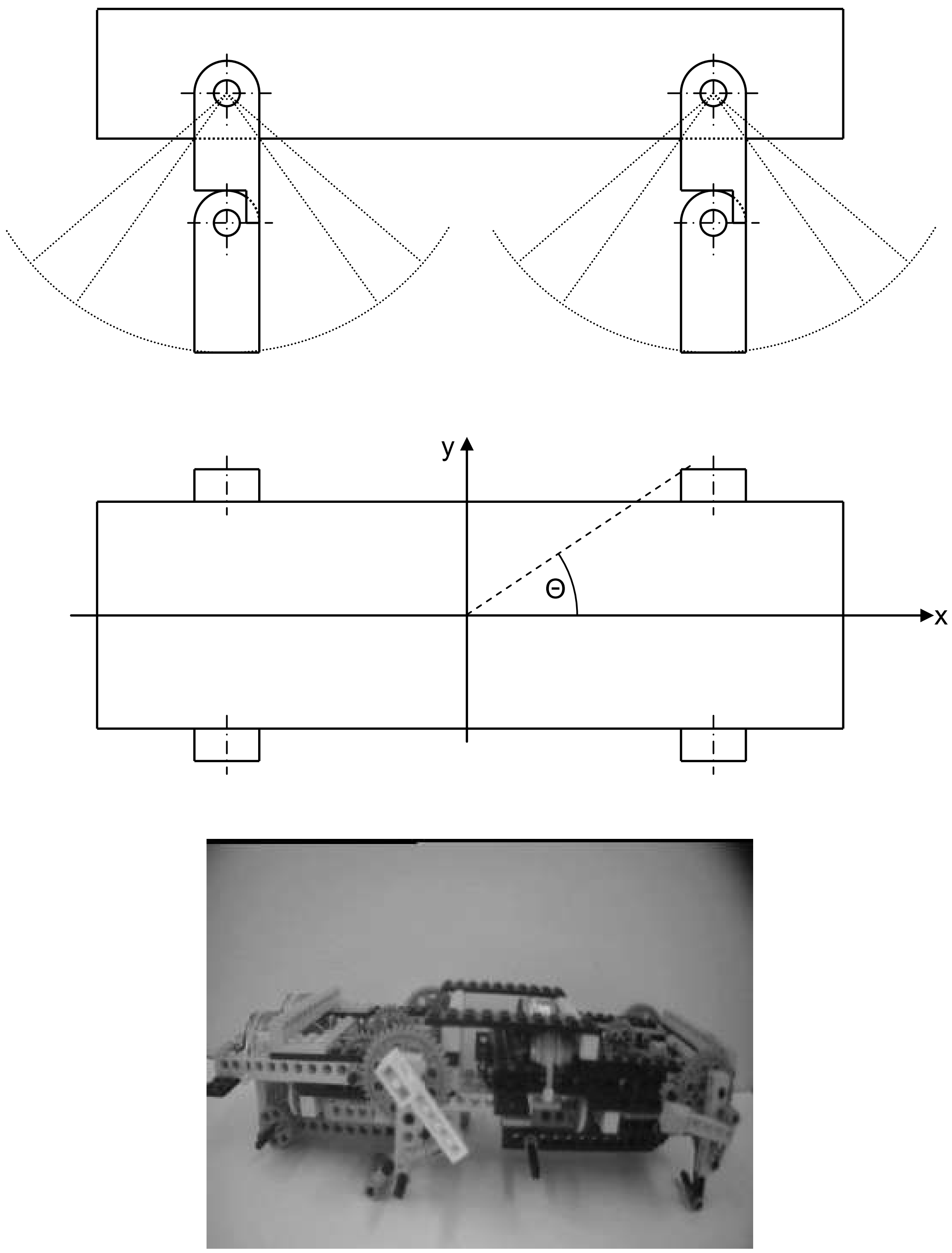


Figure 6.9: Top - Simulated quadruped robot leg configuration. Bottom – the real robot upon which the simulation was based

6.4.4. Alternate CPG strategy

In order to check if there is a simpler or more optimal method for evolving CPG networks, an alternative strategy for structuring the networks was investigated and compared to the methods described in section 6.4.3. The evolutionary tasks remained the same. This alternative strategy was based on the observation that neural networks (like many systems) perform best when large homogenous networks are split into several smaller modular ones, each of which can operate as an independent unit, but work together to form a larger whole [MacLeod 2001]. To accomplish this, the CPG networks will be separated into two functional units or modules. The first will perform the task of an oscillator and the second will modify the oscillations to form the appropriate gait patterns of activity. The connection between the two units is shown in Figure 6.10, and is similar to the work presented in [Prentice 1995].

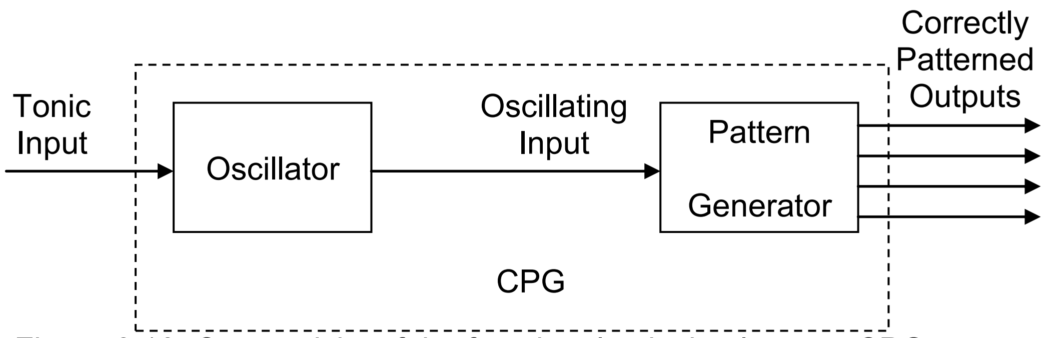


Figure 6.10: Connectivity of the functional units in alternate CPG strategy

This problem of producing gait patterns is an interesting task for an ANN and illustrates the potential benefits of modularity. Without any modularisation the entire network is required to both oscillate and produce rhythmic patterns. Since the whole network is being used for both tasks, there is a possibility of undesired connections between neurons which could interrupt either the oscillator or pattern generator. The EA used should ultimately remove the connection, but by designing modularity into the CPG this problem can be completely avoided; therefore, adding modularity might be expected to make the evolutionary task easier.

6.5. Summary

The initial sections of this chapter described the biological background to CPGs. This included the discovery that CPGs were responsible for the fundamental patterns of activity in locomotion. After this, the basic building blocks of CPGs were discussed. These include neurons, synapses and how they connect together to create networks. The required functionality of CPG networks was then specified.

Following on from that, the system for implementing artificial CPGs was presented. Initially, the important timing information required from the artificial CPGs was described, and likened to the timing patterns present in biological systems. Artificial components (neurons or processing units, and synapses or connections) were then explained which will be connected together to construct artificial CPG networks. These will be created for specific tasks using ESs as explained in section 6.3.5. The simulation used when evaluating the fitness of the CPGs was then illustrated and included the use of the artificial reflexes created in the previous chapter. Finally, the tasks for which CPGs will be evolved have been outlined.

The next chapter will present results of evolving artificial CPG networks for the tasks presented here (producing the fundamental patterns activity for biped and quadruped locomotion).

Chapter 7. Central Pattern Generators for Locomotion

7.1. Introduction

The previous chapter described the evolutionary ANN system and methods for creating CPGs to be used in the action layer of the ANS. This chapter presents the results of this task and makes use of the reflex ANNs described in Chapter 5 to control simulated legged robots.

Initially, the effect on the evolutionary system of the complexity of the simulated robot will be discussed. Following that, the results of evolving CPGs to produce a number of basic gait patterns are presented. The patterns to be created are walking and jumping for bipeds and walking, jumping, trotting, pacing and galloping in quadrupeds. After this, an alternative strategy for creating the quadruped CPGs will be investigated and compared to the original method. Finally, a discussion of the results presented in this chapter is given, followed by a summary of the chapter.

7.2. Biped locomotion

The first problem to be investigated was the evolution of CPGs which could produce the basic gait patterns for bipedal locomotion. The CPGs were tested on (and during evolution, their fitness was evaluated using) a simulation of a bipedal robot, as described in section 6.4.2. The simulated robot was stable in all directions because it is only the production of the appropriate gait patterns that are currently under investigation. These are most easily created when the animal is stable, as with the treadmill experiments on biological animals (described in [Grillner 1981]). Those treadmill experiments also suggest that stability of locomotion in legged animals is controlled by higher centres of the nervous system (which would correspond to higher, as yet unimplemented, layers of the ANS model). This is supported by the fact that newborn infants can produce walking gaits after birth that get suppressed by the CNS until it can learn to walk properly [Carpenter 1990]. Since the simulated robot was assumed to be fully stable, and the legs only have one degree of freedom, the gait pattern would only need to consist of the hip positions.

Additionally, the biped CPGs evolved here will act like a single segment of a fish; that is, alternating oscillations on each side [Gray 1953]. It would be possible to create a chain of these CPGs and then evolve synapses between each segment to produce the required delay and create a travelling wave along the length of the chain. This would allow patterns for fish swimming to be evolved.

Initially the total number of neurons in the CPG was set to the suggested value in the literature [Grillner 1985][Golubitsky 1998], which was four neurons. The neurons are of the type described in section 6.3.2. The same tonic input was applied to all neurons. The output neurons were encoded as part of the chromosome, and were therefore chosen through evolution. The synapses (which were fully recurrent) had the weights initialised between ±0.5, the delay was not used (permanently set to zero) and the type of pulse (either short or long) induced in the post-synaptic neuron was set randomly, with a probability for each type of 0.5. The short neuron pulses were 60ms and the long pulses were 2000ms. The refractory period of the neurons (the duration which the neuron cannot produce any activity after producing an output pulse) was set to 100ms.

A (μ+λ) ES (as described in section 2.7) was used to evolve the CPGs, as it was the most successful in the previous experiments with the reflex. The population size was set to 15, with the number of offspring created at each generation set to 105, giving a ratio for μ:λ of 1:7. The mutation rate for the synapses was set to 5% (meaning that for each offspring created, on average five out of a hundred genes are mutated). When a mutation takes place, only one component of a gene (synapse weight, delay or length of pulse) is mutated with probabilities of 0.6, 0.39 and 0.01 respectively. When mutated, both the weight and delay have a small random number added or subtracted and the type is inverted. The outputs are mutated with a probability of 0.033, so that for every thousand mutations, the outputs are mutated on average 33 times. A single mutation involves changing an output from its current neuron to a neuron from which no output is already being drawn from. Crossover is used to create two offspring from two unique parent chromosomes randomly selected from the elite section of the population. The probability of each gene to be swapped between chromosomes is 0.04. These values came from initial tests where the evolutionary progress was evaluated and a good set of parameters were found.

The outputs of the network are integrated and are used as inputs to two reflexes (one for each leg) as evolved previously. The outputs from the reflexes are used to control the legs of a simulated biped robot.

The fitness score for each chromosome is how far the robot moved from its initial position; therefore, higher fitness scores were better. The CPGs were evolved and it was found that the gait produced was that of a jumping gait. Both legs were synchronised in their movements and the robot was able to move as far as possible (the legs being moved as quickly as possible and maximising the distance moved in each stride). However, it was more interesting to investigate whether the CPG could produce alternating movements, since this type of movement is widely used in nature [Stewart 1998]. An additional reward was added to the fitness function for stability, so that fitness was increased if at least one leg was on the ground at all times. The results of this evolution are shown in Figure 7.1.

As described in section 6.4, the robot simulation used a leg position of 0.2 as the position at which the leg moving backwards would be lifted from the ground, and 0.8 was the position at which a leg would come into contact with the ground after being moved forward past that point. As can be seen in Figure 7.1a, the legs are alternating as they would do in a normal walking type of gait. They are moved backwards until the leg can be lifted from the ground and are then moved forward and extended past the point of contact, after which it is moved backwards again. The other leg is performing the alternate movement, 180° out of phase, for the duration of the test.

The path of the robot is shown in Figure 7.1b and is as expected for this type of biped gait. With each step, the robot turns slightly, which is why the path appears to oscillate. The reason it is directed at an angle and not straight along the x-axis is because each step causes the robot to twist and move forward. Each odd step will angle the robot away from the horizontal and every even step will angle it towards the horizontal (but never past it as long as the steps are balanced, which they are).

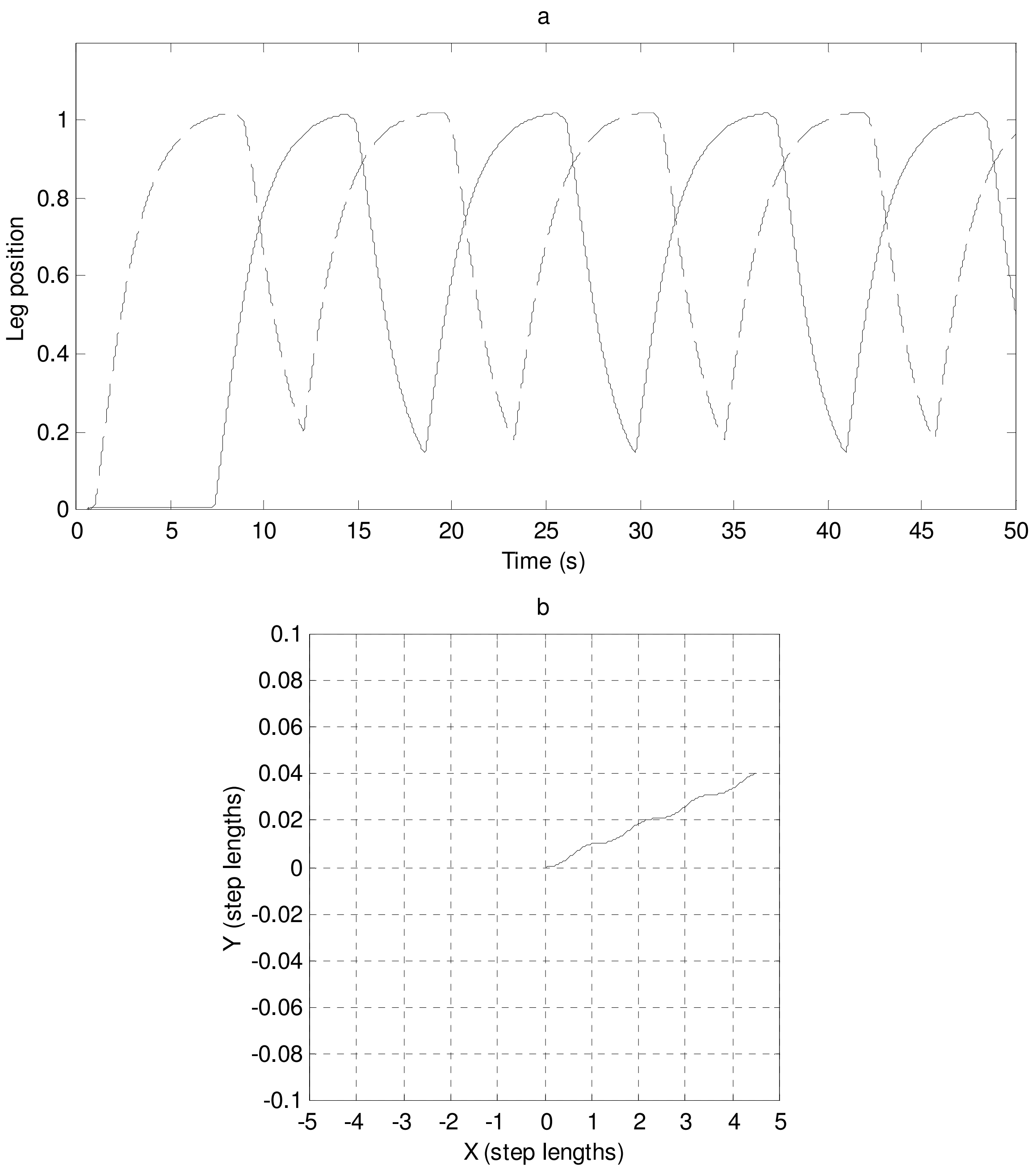


Figure 7.1: Results of best evolved CPG for biped locomotion with reward for one leg on the ground at all times. a) leg positions as controlled by best evolved CPG (solid line is left leg, dashed line is right leg). b) plan view of path of robot

7.3. Quadruped Gaits

The general application of the ANS was investigated by evolving another group of CPGs using the same methods as detailed in the previous section. In these, the focus was changed from biped to quadruped gait patterns. The gaits which were targeted for evolution in this section were gallop, trot, pace, pronk and walk. The phases of each gait were shown previously in Figure 6.8.

The network set-up is shown in Figure 7.2. Initially the number of neurons was set to eight (following the 2×n guide, where n is the number of legs). The input to the network was a tonic signal. Four outputs were taken from unique neurons.

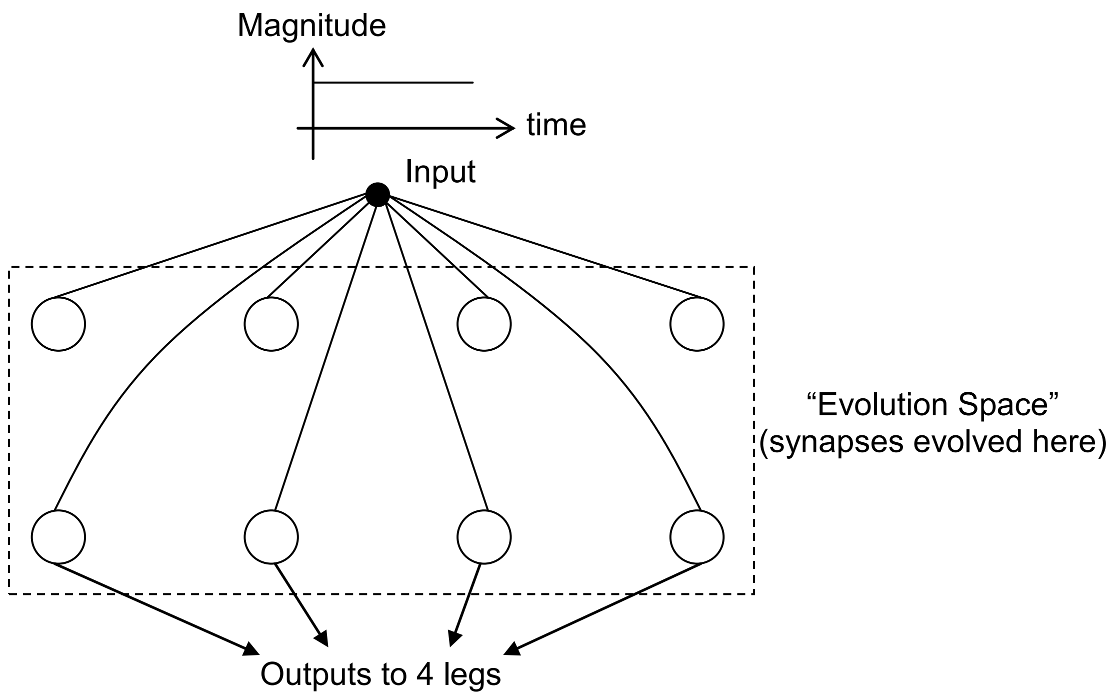


Figure 7.2: ANN setup for evolving CPGs

The encoding of each chromosome was extended to include both the synapses and outputs, as shown in Figure 7.3. Table 7.1 shows the times used in the network.

Weight 1	Delay	Type of induced pulse	Synapse encoding genes, for n^2 synapses (where n is the number of neurons in the CPG)
…	…	…	
…	…	…	
⋮			
Weight n^2	Delay	Type of induced pulse	
Output 0 (unique neuron number)			Output genes. Each is a unique neuron number.
Output 1 (unique neuron number)			
Output 2 (unique neuron number)			
Output 3 (unique neuron number)			

Figure 7.3: Chromosome structure used for evolving the quadruped CPGs

Table 7.1: Times used in CPG networks

Description	Time (ms)
Short pulse (as set by synapse type)	60
Long pulse (as set by synapse type)	2000
Refractory period of neurons	100

The same (μ+λ) ES which was implemented for evolving the biped CPG was also used here. Population sizes were the same as before (15 parent chromosomes, 105 offspring). The chromosomes were evolved for 500 generations. Crossover and mutation operated in the same way as with the biped, although mutation was changed to add or subtract normally distributed random numbers instead of uniformly distributed numbers. This was done to give a finer control over the changes, but still allow large mutations, only less often.

It was found that when the CPG was evolving, the best fitness in the population would suddenly drop. It was realised that this was caused by the legs being extended to their furthest forward position and then remaining there. Another problem discovered was that sudden mutations in the genes encoding the output neurons would significantly alter the output leg patterns to a degree where they were not performing a useful gait. To correct this and make the task easier for the evolutionary process, the inputs and outputs were fixed to a limited (and separate) set of neurons. The inputs were limited to reduce the amount of stimulus to the network (so the legs would not become paralysed in the forward positions). Fixing the outputs solved the problems of sudden losses in performance, and made the target of the evolutionary task much more stable. The choice of these neurons were totally arbitrary, since the network was fully recurrent and any neuron could connect in any way to another neuron. From that point of view, the neurons which were used for inputs and outputs did not make any difference.

The optimal number of neurons for this CPG was found to be 16 (rather than the initial setting of 8) which allowed all four legs to be controlled and contribute to the appropriate output patterns. The inputs and outputs however, remained connected to the same neurons as before (since the fully recurrent connections would mean there would be no difference if the neurons were changed).

Instead of using a fixed mutation rate, which usually provides a sub-optimal evolutionary performance, a variable mutation rate was employed for these more complex gaits. The mutation rate was decreased with progress of evolution, and increased if the fitness had not improved after a large number of generations.

From inspection of the activity of the CPG neurons (see Appendix D), it was decided that there needed to be more control over the pulses produced by the neurons and also that the short pulse time was too short and was not having any useful effect on the leg patterns being produced. This time was increased to 300ms and an additional type of synapse was created which induced medium length pulses from neurons. The time of pulse produced was set half way between the short and long pulses, at 1150ms.

With this final change the evolutionary progress was improved enough so that gaits were visibly starting to be formed, although they were not that good within the initial 500 generations. The evolution was continued for another 500 generations if the fitness was not good or the gait was not visibly well formed. The final results for a CPG producing a quadruped gallop gait are shown in Figure 7.4, below.

In all the figures showing the results of evolving CPGs, the following key is used:

Leg Positions		**Fitness graphs**	
Left Front ——	Right Front-----	Best ——	Average-----
Left Back ·········	Right Back -·-·-·	Worst ·········	

Figure 7.4a shows the leg positions of the robot while under control of the fittest CPG network. As can be seen, the positions of the legs are consistent with that of a gallop. The two fore legs (solid and dashed lines) are moving as one pair, and are anti-phase to the two hind legs (dotted and dash-dotted lines). The pattern of legs positions is quite stable, with the only anomaly being in the first and second steps.

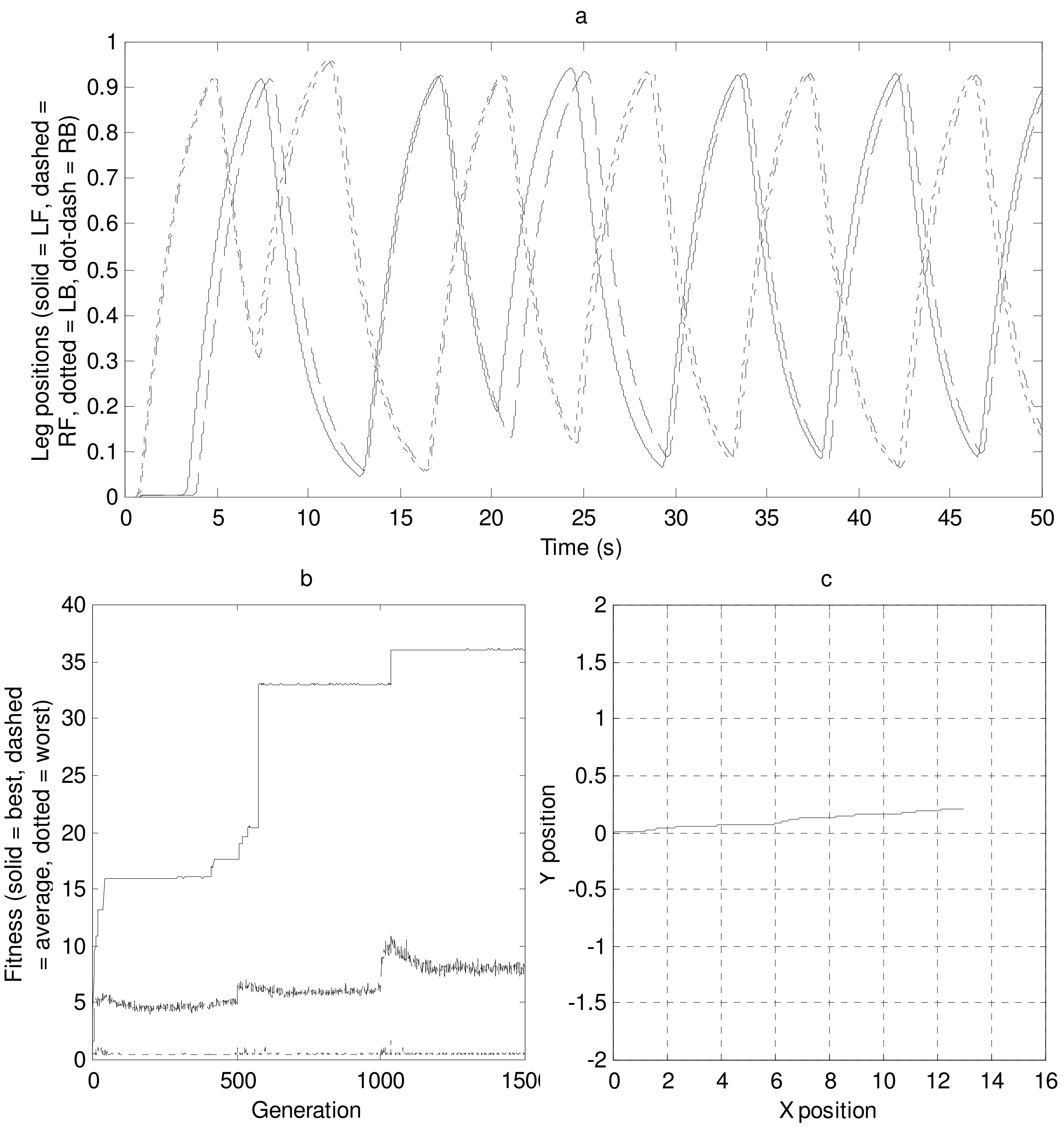


Figure 7.4: Quadruped gallop. a) Leg positions during gallop. b) Fitness during evolution. c) Plan view of path of robot movement

Figure 7.4b shows the fitnesses (best, average and worst) during evolution. The strange appearance of the average fitness is due to the way evolution was carried out. Initially, 500 generations was the threshold for stopping evolution but was twice continued from where it had stopped at the generation limit. This meant that the mutation rate (which changed depending on the evolutionary progress) was reset to the initial value. However, at least this highlighted any problems with the way in which the mutation rate was being changed. The long periods of no increase in best fitness also indicate a problem with the way the offspring were being generated (since no fitter chromosomes were being discovered for large numbers of generations). Further results in this section used the same method, but with more suitable values for the various parameters to achieve better results.

Figure 7.4c shows the path the robot takes while moving under the control of the CPG (with the leg positions from Figure 7.4a). The robot travels far further in the X direction than the Y direction, making the path quite close to moving forwards in a straight line (and the straighter the path, the better the gait has been formed in terms of balance between steps, equal step lengths). This was because the robot was initialised so that it is facing along the X axis.

The next gait to be evolved was the trot. In this gait, the diagonal legs move in pairs, with each diagonal pair moving in anti-phase from each other (as shown in Figure 6.8). The results of the evolution for this type of gait are shown in Figure 7.5, below.

The leg positions produced by the CPG for a trotting gait were almost perfect, as seen in Figure 7.5a. The only problem occurred in the final step where one leg (the right hind leg) was moved forward before the other leg in that pair. Compared to the gallop, the visibly better leg patterns are reinforced by the higher final fitness score. However, the fitnesses shown in Figure 7.5b also show the previously mentioned attribute of no improvement over a long number of generations. The same affects of restarting evolution are observed in Figure 7.5b.

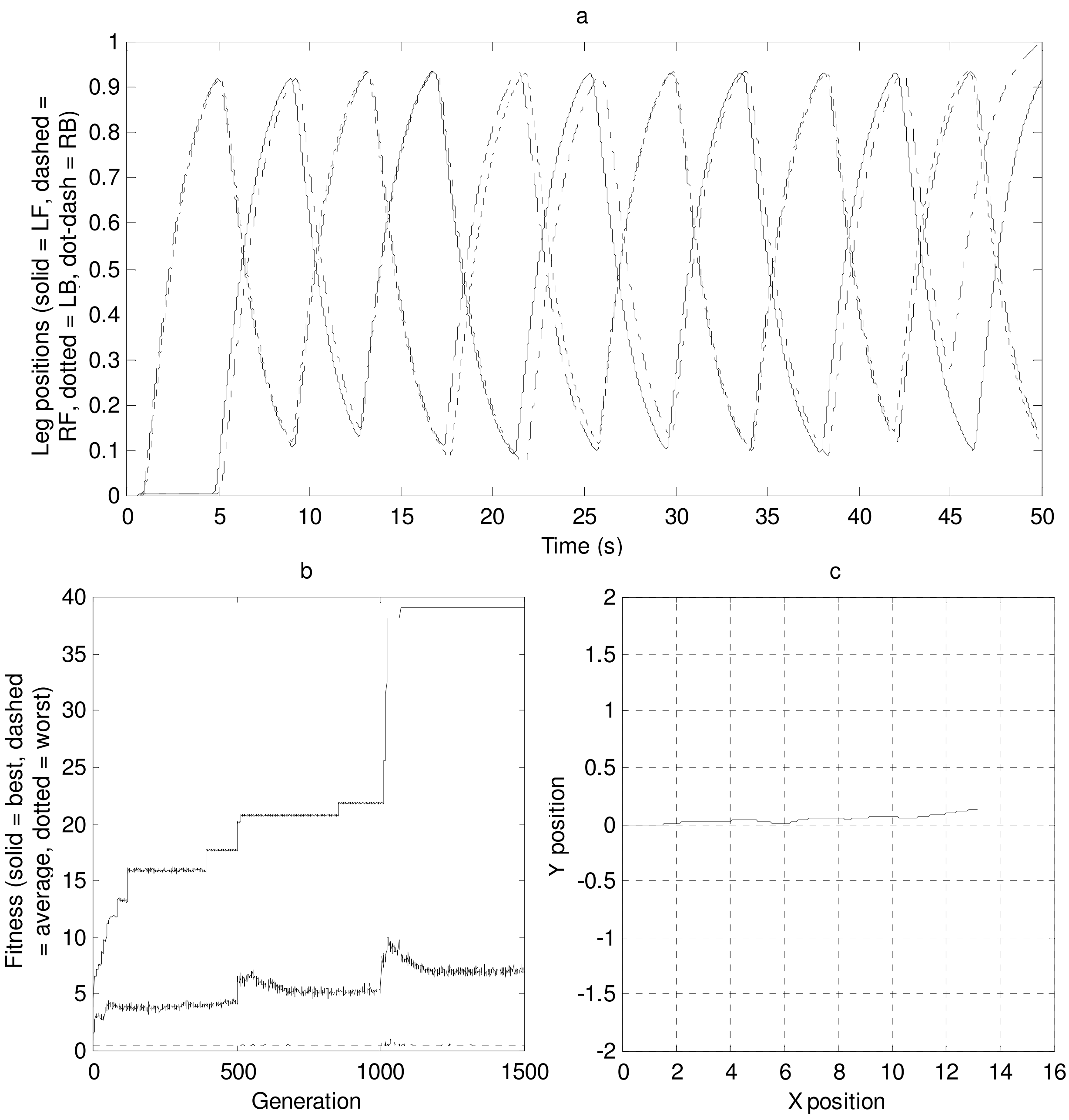


Figure 7.5: CPG to produce quadruped trotting gait. a) Leg positions while CPG controls robot. b) Fitness scores during evolution. c) Plan view of robot path during movement.

The next gait is the pace, where the two legs at one side of the body move as a pair, with the legs on the opposite side moving as anti-phase pair. The results are shown below in Figure 7.6.

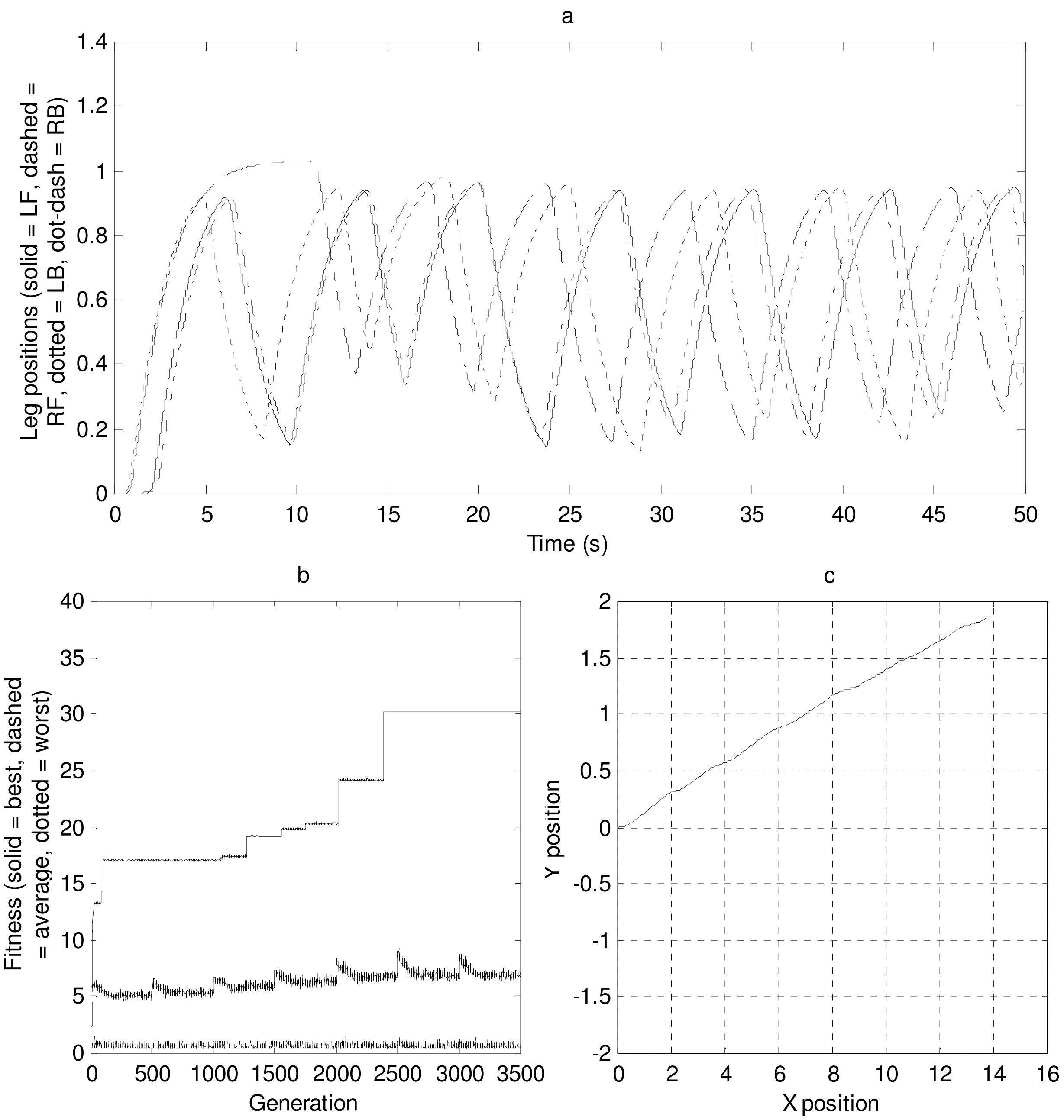


Figure 7.6: Pace quadruped gait. a) Leg positions. b) Fitnesses during evolution. c) Plan view of robot path during movement

The final fitness is lower than the previous two gaits, and took more generations to reach that level. This can be seen in the leg positions, which do not always move in two separate anti-phase groups at each step. However, the majority of steps taken are as expected for a pace gait, with the correct legs moving in pairs. The lack of improvements in fitness score were for much longer periods of time in this evolution. The robot covers almost the same distance as in the previous two gaits, but this is simply because it takes the same total number of steps and is not related to whether the legs are moved in pairs or not.

The gait of the final CPG to be evolved was the pronk. This gait involves all legs moving together, and can be considered as jumping. The results of this evolution are show in Figure 7.7, below.

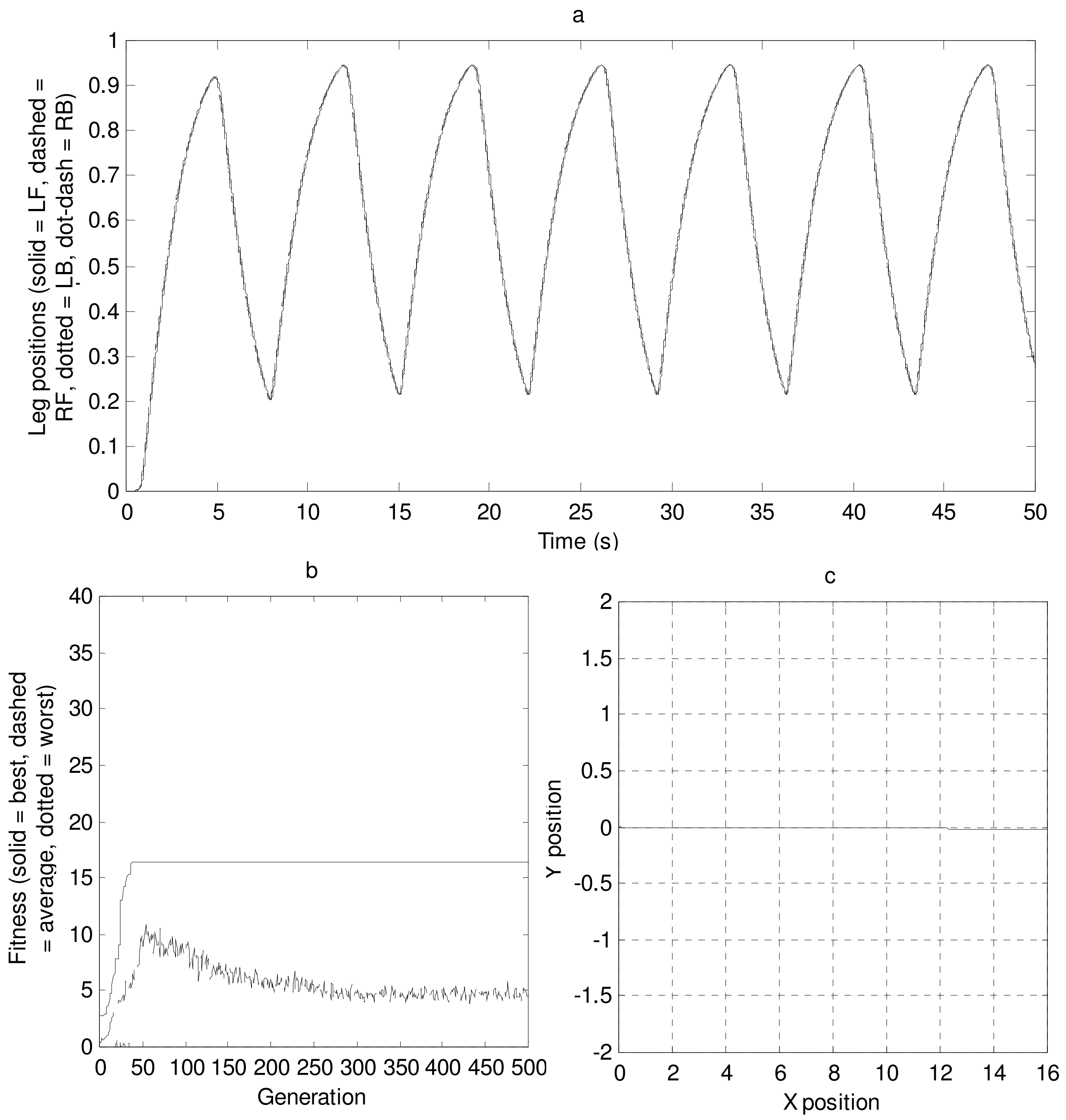


Figure 7.7: Pronk (jumping) gait. a) Leg positions. b) Fitnesses during evolution. c) Path of robot movement.

Although it is not possible to see from Figure 7.7a, all four legs are moving together which is the appropriate leg pattern for the pronk gait. The fitness score is lower than with the other gaits because it is measured only by the distance the robot moved, with no inclusion of fitness measures for producing stable, efficient or fast gait patterns (as with the other gaits). The fitness peaked very early during evolution, showing that this gait was very simple to evolve. The robot path shown in Figure 7.7 confirms the leg patterns and fitness graphs, showing that the robot moves in a very straight line as expected with all four legs moving in synchronicity.

A walking gait was never fully successfully generated using this method of CPG structure. The tendency was for the legs to oscillate but never diverge into separate stepping phases.

7.4. Alternate CPG strategies: separate oscillator and pattern generator

At this point, an alternative strategy for the topology of the CPG networks was evaluated. The reason for this, as mentioned in section 6.4.4, was to try to improve the performance of the evolved CPGs while also making them smaller and quicker to evolve. The design of the CPG was made more modular to achieve this (as was shown in Figure 6.10).

ANNs are more capable of performing single rather than multiple tasks [MacLeod 2001]. This also has the side-effect that they can be smaller and will take less time to train or evolve. To prove this, the CPGs have been split into two functional units (as proposed for a model of a CPG in [Prentice 1995]). The task of the first unit is to oscillate. For this, the CPG evolved for the biped walking pattern will be used, as it produces alternating oscillations from each output. This will allow the work to concentrate on the second unit. This unit will be a pattern generator taking oscillating inputs from the first unit and producing the appropriate gait patterns as outputs.

The gait patterns to be produced by the CPG using the more modular approach are the same as the previous section (walk, trot, pace, gallop and pronk).

The networks in this section were reduced to eight neurons since the simpler task allows less neurons to be used. Inputs to and outputs from the CPG are the same as with the single unit CPGs.

Use of the synaptic delays was permitted, since there were only two spare neurons (i.e. no inputs or outputs connected), which could be used solely for producing the correct patterns.

The method of changing the mutation rate was changed from that used in the previous section (linear increases and decreases depending on progress of average fitness). This was done to solve the problems of long periods of generations without any increase in fitness. The new method used was to select the mutation rate using a sinusoidal waveform, increasing the angle when there was a lack of fitness improvement over a small number of generations. This is shown in Figure 7.8.

This is the reason for the oscillations observed in the fitness scores (graph b in Figures 7.9 to 7.12). When there is a lack of improvement in the best fitness of the population, the mutation rate changes. The peaks are due to low mutation rates, but does not necessarily mean that the solutions in the population are getting better. It could simply be that the mutation rate is so low that there is little variation in the population. The troughs in the graphs are caused by high mutation rates that cause too much variation in the offspring, with more of them having low fitnesses.

The fitness functions used remained largely unchanged in the way they were scored, although the actual fitness levels were modified. Each component (distance and reward for leg pattern) was normalised and weighted to allow for easier balancing of the fitness score between the components.

The first gait to be evolved for the separate oscillator CPG was the gallop. The results of the evolution of this CPG are shown in Figure 7.9.

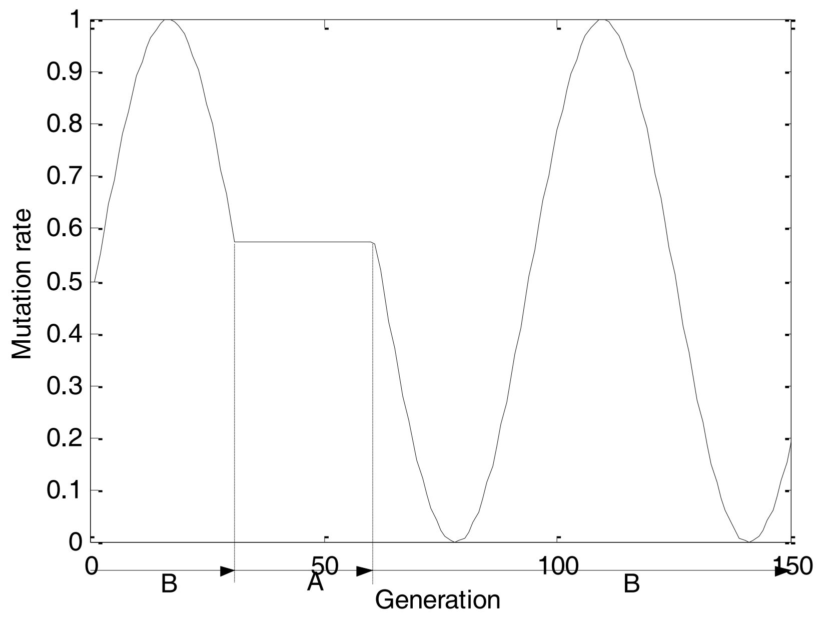


Figure 7.8: Oscillatory method for selecting mutation rate. Period A shows the mutation rate when the fitness is improving, B when there is no improvement. 1 represents the maximum mutation rate and 0 represents the minimum.

The leg patterns are well formed (Figure 7.9a) and although there is a slight gap between the front pair of legs (solid and dashed lines) throughout the time shown, the two legs basically move together when they are involved in the power stroke of the step cycle (when they are moving backwards and the foot is on the ground).

The best fitness reached a high score within the first 1000 generations, which is lower (i.e. better) than were required with the non-modular CPG. Although the components of the fitness score were normalised, the weights chosen for each component meant that the theoretical highest fitness score was 1.4. The oscillations on all fitness scores are caused by the sinusoidal method of altering the mutation rate. When the mutation rate is very low, new offspring will be very similar (depending on recombination rate) to the parents, thus reducing variation in the population and causing more fitness scores to be at similar levels (and therefore a higher average and worst fitness).

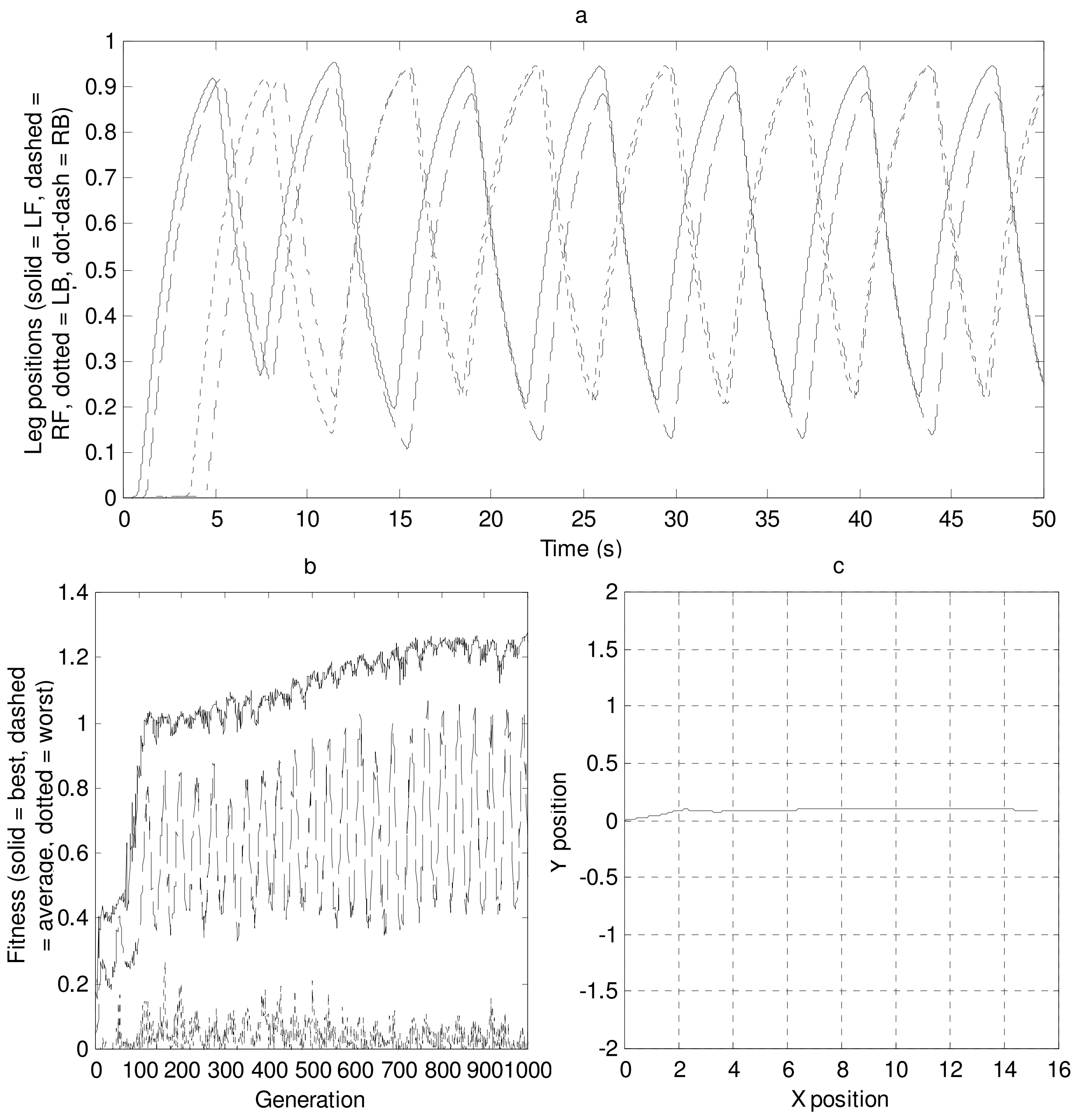


Figure 7.9: Quadruped gallop with split CPG. a) Robot leg positions. b) Best, average and worst fitnesses during evolution. c) Path of robot when controlled by CPG producing leg patterns from a).

The path which the robot takes is quite straight (as is expected) with the legs in each moving pair moving in close co-ordination. The path which the robot took is much straighter than when the gallop gait was produced by the single unit CPG. This is because the movements of the legs are more balanced than previously.

The results of evolving a CPG to perform a trotting gait are shown in Figure 7.10.

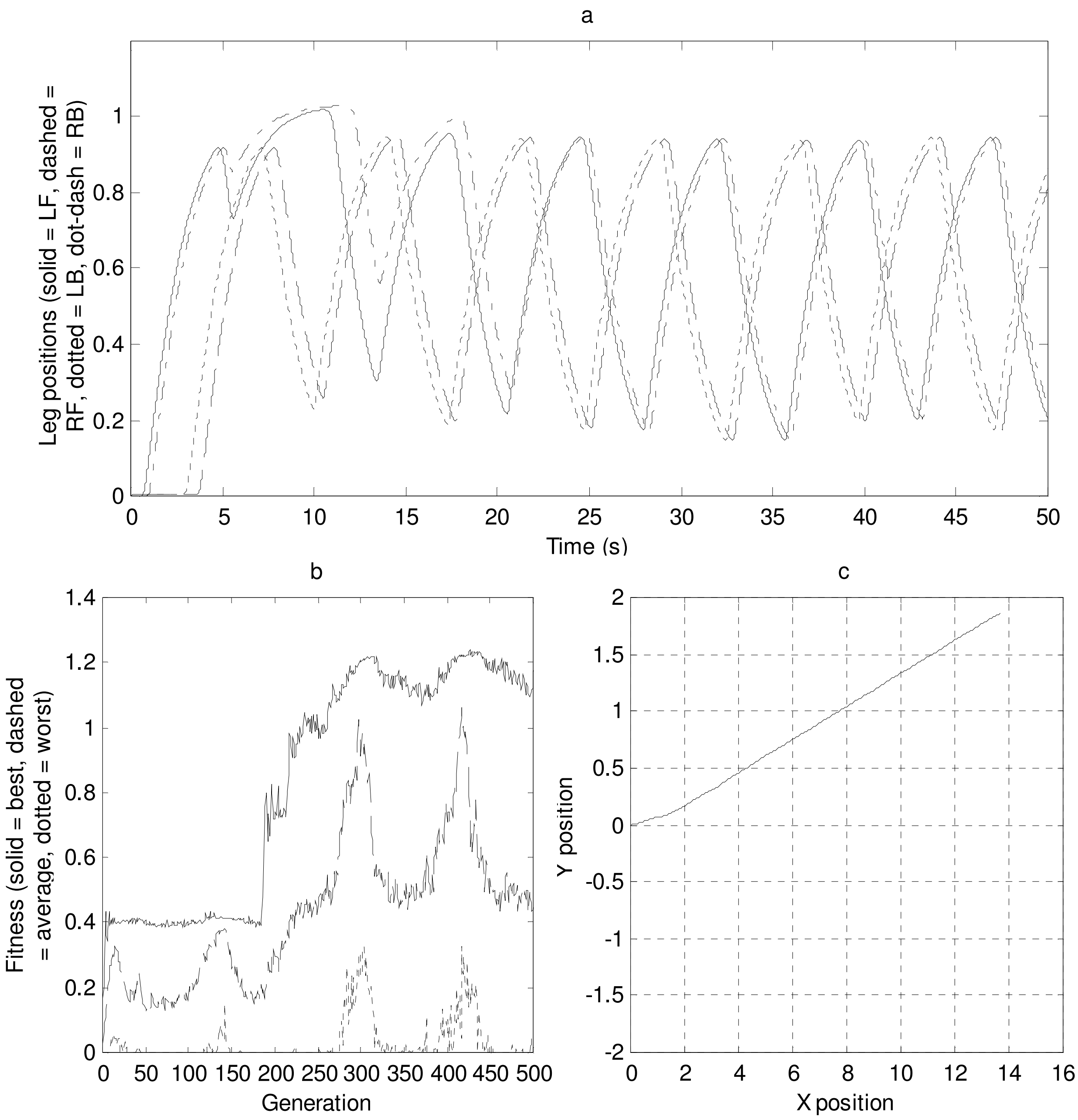


Figure 7.10: Results of trot gait CPG evolution. a) Leg patterns of robot when moving. b) Best, average and worst fitness during evolution. c) Path of robot motion for the leg patterns shown in a).

Apart from the first two steps of the front left and right hind legs (solid and dash-dot lines respectively) the leg positions are as expected for this gait. The fitness shows little progress past its initial value for slightly less than 200 generations. The fitness gradually increases until it reaches a high of just over 1.2. This is the same value that the fitness reached when evolving a CPG for the gallop gait (Figure 7.9b), except it was reached in around 300 generations for the trot as opposed to nearly 1000 (for the gallop). This was due to optimisations made to the sinusoidal mutation rate selection method, which also accounts for the lower number of oscillations visible in the average and minimum fitness values.

The direction of the path the robot took, as shown in Figure 7.10c, is straight, with the angle being caused by the first two irregular steps. The straightness of the robot path is caused by two legs from opposite sides of the robot moving as a pair and therefore the rotation caused by each leg cancels out.

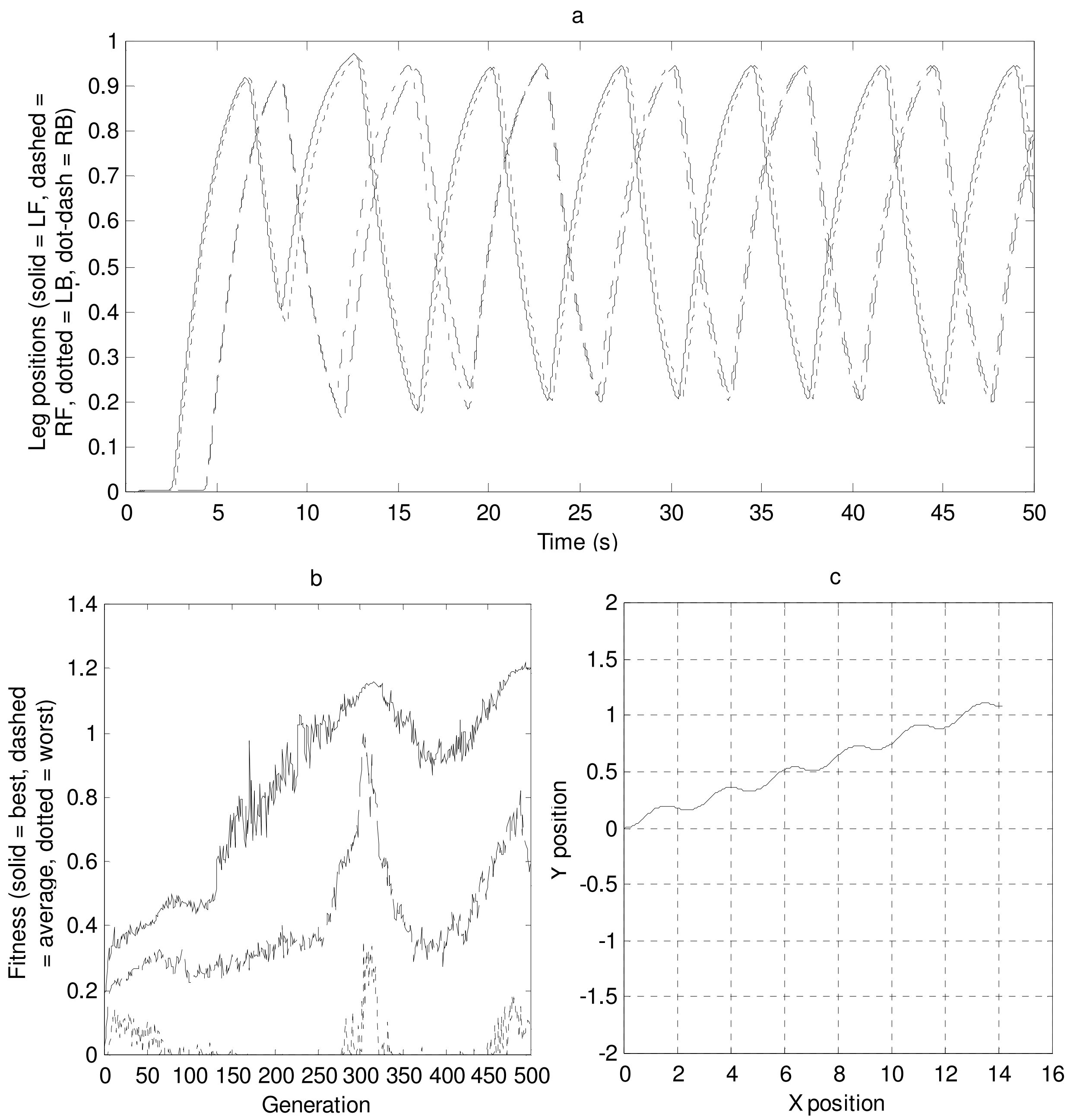


Figure 7.11: Results of evolution of a CPG for pace gait. a) Leg positions during robot motion. b) Best, average and worst fitness during evolution. c) Plan view of robot trajectory caused by leg patterns in a).

Figure 7.11 shows the results of evolving a CPG to perform a pace gait (the legs on either side working in pairs). The leg positions during the final testing phase are as expected for this gait, with the left front and hind legs (solid and dotted lines) moving as one pair and the other two legs as the anti-phase pair.

The fitness is similar to that found for the trot split CPG, except the progression of the best fitness is more continuous. It peaks about 1.2, slightly lower than for the previous two CPGs. This is due to the legs being stationary for longer at the start of the test phase.

The path of the robots movement is shorter than with the previous two gaits, supporting the lower fitness due to the delayed start of leg movement. The oscillations on the trajectory are due to the rotation of the robot body, caused by the legs on each side of the body entering their power stroke together.

Compared to the single unit CPG which performed a pace gait, the split CPG has better formed leg positions. The pairs of legs are moving as two distinct pairs and are almost exactly in unison. Although there is one malformed step for each CPG, that step is more in error in the single unit CPG than the split CPG. The path of the robot is straighter in the split CPG, although there are larger oscillations (which is caused by correctly formed steps).

Figure 7.12 shows the results of evolving a CPG to perform a pronk gait. The leg positions during testing are not perfect, since each leg is slightly separated from the others. However, the formation of the gait is good, as all four legs follow the appropriate pattern.

The fitness is lower than the other gaits, although this is for the same reason as the pronk gait when the oscillator was a single unit; the fitness was purely the distance moved by the robot. Therefore, the fitness levels reached for the pronk gait were as expected. Similarly, a high fitness level was reached in a low number of generations compared to the other gaits.

The path of the robot motion is slightly twisted, which is due to the separation between the legs during testing.

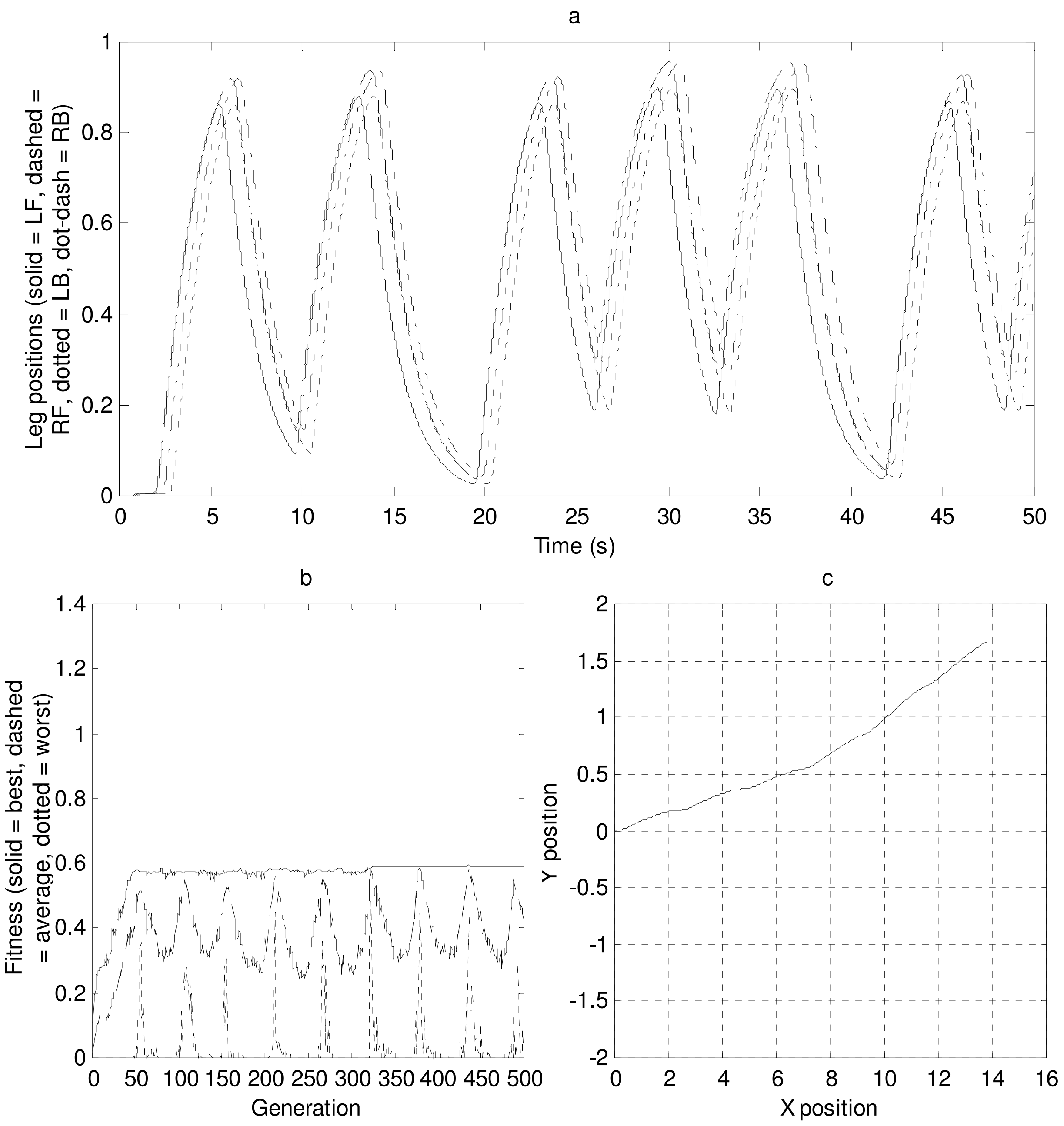


Figure 7.12: CPG evolved for pronk gait. a) Leg positions of robot during final test. b) Best, average and worst fitness during evolution. c) Path of robot during final test.

Overall, the single unit CPG for the pronk was better than the split CPG. The reason for this is that the EA has not optimised the synaptic delays properly (shown in section D.10). It was expected that very few of the synaptic delays would be anything other than 0, since everything should be co-ordinated to move together. Since the single unit CPG always had the delays set to 0, the correct co-ordination was easier to find.

Figure 7.13 shows the results of evolving a CPG, with separate oscillator and pattern generator units, to perform a walking gait. The results are encouraging, even though they are not perfect. As can be seen in (a), the legs are moving in three separate phases (two legs are moving together in one phase). Ideally there should be four separate phases, and while the CPG has not quite managed to achieve that, it is plain to see that it is close. This is reflected in the fitness graph which shows a low score (the optimum is at a level similar to the gallop, pace and trot scores).

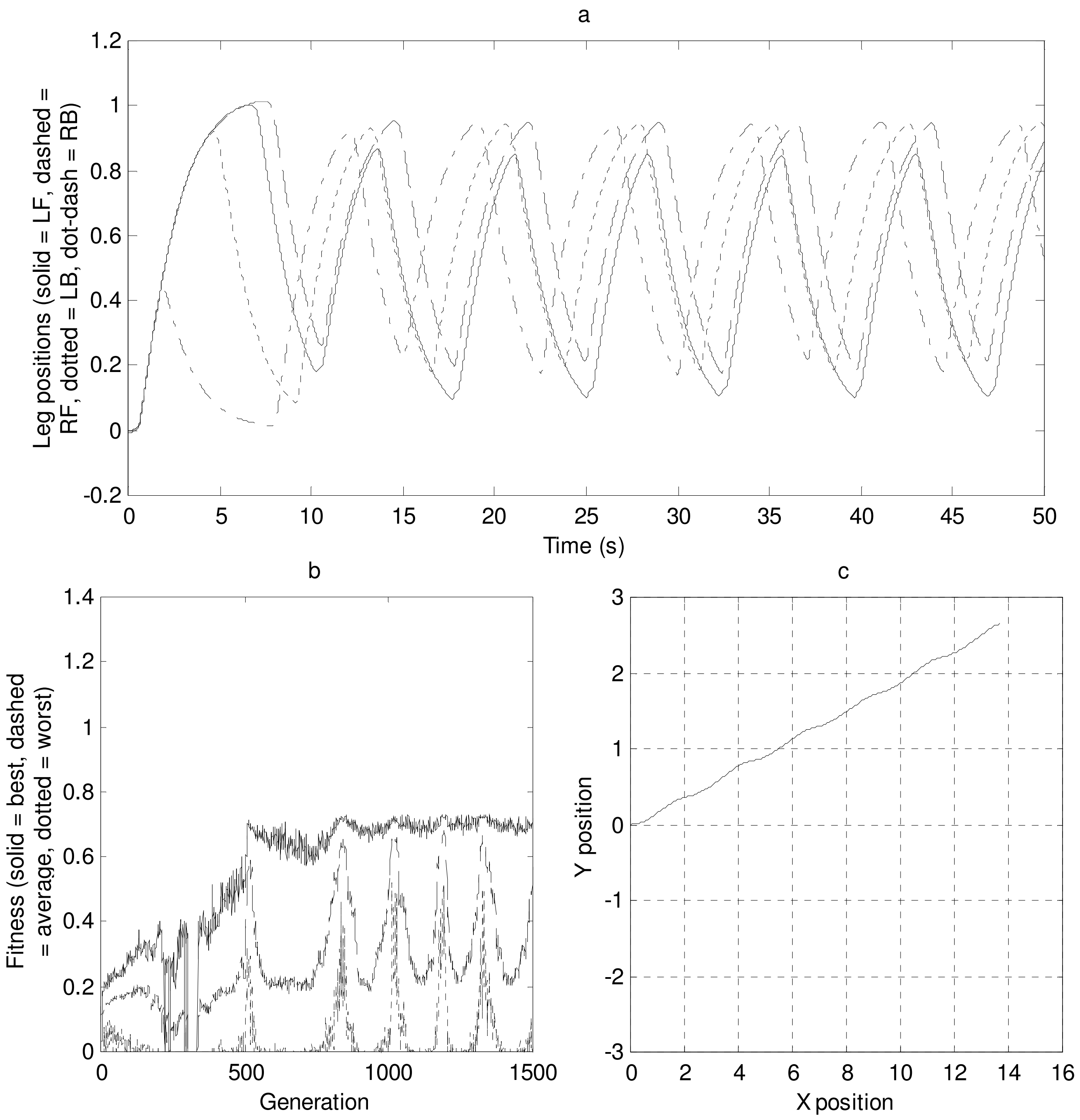


Figure 7.13: Walking patterns produced by separate unit CPG. a) Leg positions during final test. b) Best, average and worst fitness during evolution. c) Path of robot while walking.

The verification that the evolved CPGs could be transferred to the real robot to produce the same gaits as with the simulated robot is shown in Figure 7.14.

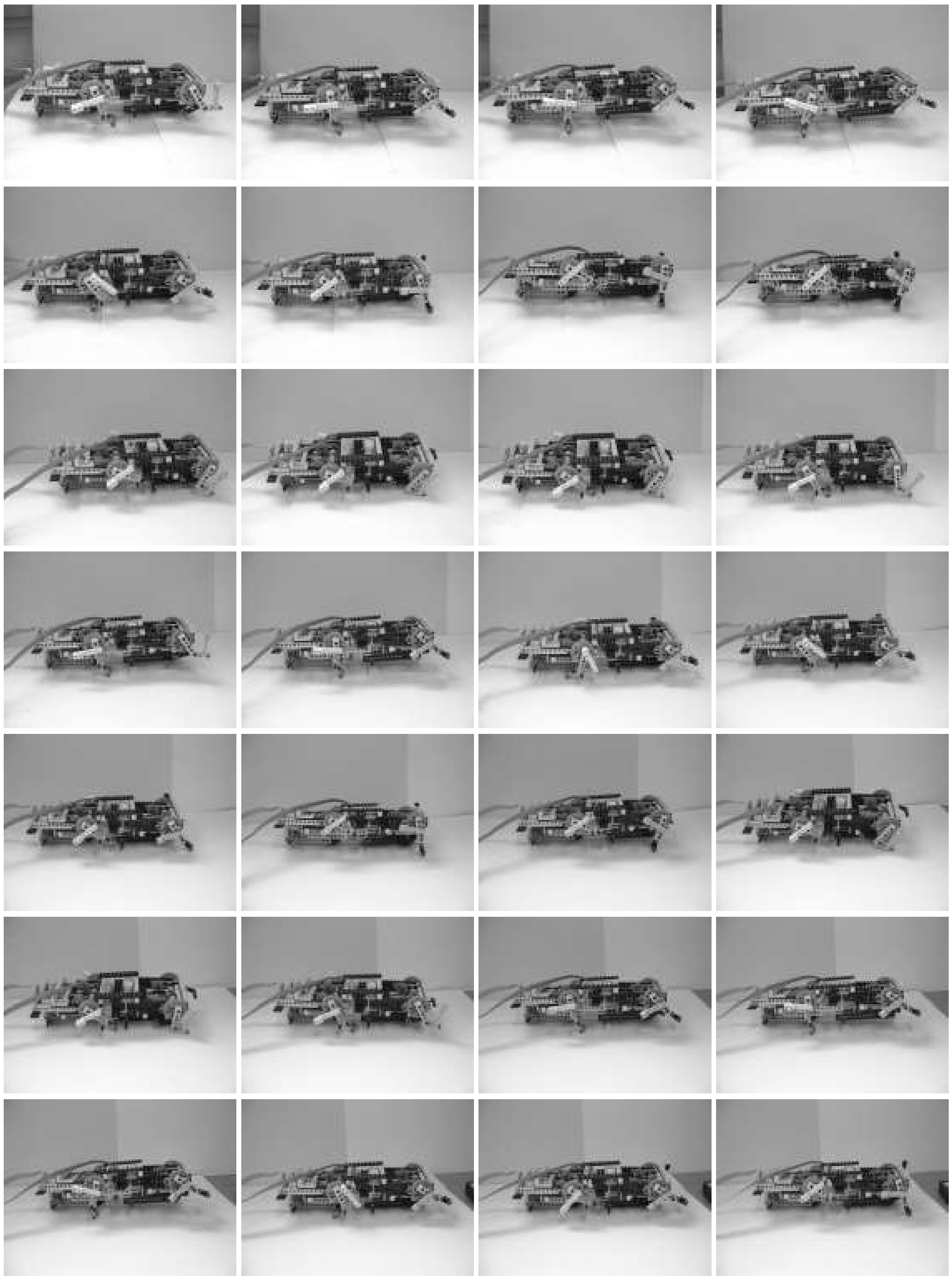


Figure 7.14: Series of pictures showing the real robot walking under control of the CPGs and reflexes (acting as an extra verification of the evolved networks).

7.5. Discussion

The main reason for the improvement in the number of generations taken to evolve to a near optimal fitness level is the separation of the CPG into two functional units. This makes the evolutionary task simpler, as the ANN only has to be evolved for a single task. The task of the oscillator unit is to produce rhythmic pulses of activity from tonic inputs, while the pattern generator unit must produce correctly patterned outputs from those rhythmic pulses of the oscillator. Compared to this, a CPG comprising of a single functional unit must convert tonic inputs into correctly patterned outputs. Although this is possible, the results presented in this chapter show that it is a more demanding task than using two separate units. Since the task is simpler it is possible to use lower numbers of neurons, which assist in lowering the number of generations required (by reducing the solution search space). We can therefore conclude that modular networks are more evolvable than homogenous ones.

The pronk gaits for the two methods of CPG creation have similar evolutionary performances, reaching a good fitness level in a very low number of generations. The gallop, trot and pace gaits (within each CPG strategy) can be evolved in similar numbers of generations, but in all cases take longer to evolve than the pronk gaits. The CPGs which produced walking gaits were the most difficult to evolve. This sequence of increasing evolutionary complexity is proportional to the number of phases of leg movement required by a gait. The pronk consisted of one phase (all legs moved together), gallop, trot and pace gaits have two phases (two pairs of legs moving separately) and the walk has 4 phases (each leg moves independently). Again, however, the superiority of the modular scheme is shown by its success in evolving (a reasonable) walking gait while the homogenous network of similar size could not.

The evolved CPGs had a low number of neurons. The suggested number of processing units for a CPG is $2 \times n$ where n is the number of legs (or joints if multiple degrees of freedom per leg) or segments on the animal or robot [Grillner 1985][Golubitsky 1998]. The biped CPGs were able to evolve with a network which matched this (four neurons) but the quadruped CPGs required more than the suggested number (twelve or sixteen neurons). However, the number of processing units per joint specified by [Grillner 1985] is not the complete number, as it is mentioned that there are unidentified neurons participating in the CPG (since the processing units are more like groups of neurons). Additionally, the processing units assumed in the 2×n suggestion of [Golubitsky 1998] are complex mathematical oscillators, rather than the simple types of neurons as used here.

Despite some neurons being inactive (see Appendix D), setting the total number of neurons in the CPG exactly to the suggested number of neurons makes the task of evolving the gait patterns difficult, to the extent that many times the number of generations must pass before the fitness approaches that of the CPG with more neurons. This is because with the exact number of neurons, the CPG must be evolved to its optimal size immediately (i.e. it cannot evolve gradually), and there is no room for errors or redundancy during evolution. It may also indicate inefficiencies in the EA implementation.

7.6. Conclusions

CPG networks can be evolved using small numbers of computationally simple neurons. The same method of evolution can be applied to the creation of CPGs for a number of different gait patterns on different configurations of simulated legged robots.

By making the structure of the CPGs as modular as possible, they can be constructed using less neurons and evolved in a lower number of evolutionary generations than CPGs which comprise a single functional unit. The extra modularity was introduced by separating the CPG into discrete oscillator and pattern generator units.

The complexity of the task of evolving a CPG to produce any given gait pattern increases with the number of phases of movement required in that gait.

7.7. Summary

This chapter has presented the results of experiments in evolving ANNs to act as CPG networks. These were demonstrated to produce various different gaits (walk, trot, pace, gallop and pronk where appropriate) for simulated biped and quadruped robots. Small networks of computationally simple neurons have been able to create the gait patterns, although making the CPGs more modular allows them to be smaller and evolve in less generations.

The next chapter will discuss all results presented in this and previous chapters and compare them to the theory presented here and from existing literature.

Chapter 8. Discussion

8.1. Introduction

This chapter will discuss the experimental work and results presented in the previous chapters of the thesis. A review of the work will be given, along with a summary of how well the implementations performed and suggestions of how they could be improved.

8.2. Discussion about ANS

The ANS model used here has been developed and refined throughout the duration of the project. The first changes to the ANS, since its proposition in [MacLeod 1998] and [MacLeod 1999], were made to the reflex layer. This was a change away from the previous use of the term "reflex" in animat design [Arkin 1995][Wikman 1996] to a lower level of control. In MacLeod's scheme, low level control was actually direct control over individual actuators on the animat, allowing for improved generality in the functioning of the reflex layer.

Another important area of the ANS model which was refined was the addition of a behavioural level, between the action and sensory layers. These behavioural modules would initially target those innate behaviours of animals, which can be added to the ANS through the use of EAs. Further behaviours would be added to this layer through learning. An example of a behaviour might be the avoidance of a predator. The behavioural module would need to produce sequences of reflex control signals and action activations to perform the appropriate movements (turning and running). Animat control systems which are based on behavioural systems have been studied by many other groups (for example, [Brooks 1986], [Bartolini 1995], [Sousa 1996], [Maurer 1997], [Gat 1998] and [Herbert 1998]), so much can be learned from other work before implementing the behavioural layer in the ANS here. For example, SA (Subsumption Architecture) is a good example of a system that can be utilised here. Combined with the higher and lower layers of our ANS a more general and simpler system for control can be created. Layers above the behavioural level would provide the sensory processing, so the SA behaviours can be controlled by simpler signals. The lower layers (action and reflex) would remove either the duplication of control, or connections into the middle of SA modules, by being available (with well defined interfaces) to all behaviours.

The ANS presented here was initially developed for the control of simple animats, in the same way that lower life forms (invertebrates and lower vertebrates) have relatively simple nervous systems (as compared to higher vertebrates). The ANS is comparable to those simple nervous systems both in terms of complexity and tasks which it would be required to solve. The structure is designed so that efficient engineering solutions can be created and are more understandable than if a complex system was used.

If the ANS presented here was to be used on an extremely complex physical system (e.g. controlling a full humanoid robot), some aspects of the ANS may need to be improved. The most obvious additions to the ANS would be more parallel connections between (non-adjacent) layers, for example connections from the higher layers of the ANS directly to the reflexes. These types of connection would allow higher layers to control the actuators of the animat without having to use the intermediate layers of the ANS. This would be required for skilled control (as with the primary motor cortex [Kandel 1991] of higher vertebrates) assuming the movements were not possible by using existing modules from the ANS. The behavioural modules would also require connections spanning layers, so that behaviours could be comprised of sequences of actions and simpler movements. Direct parallel connections between layers would also assist in unsupervised learning, as the results of descending control signals could be passed back to higher layers (for example, sensory data or efference copies of the resulting control movements). The advantages of these extra connections would be the wider range of activities which the robot could perform, as they would no longer be fixed to passing linearly through the layers of the ANS. However, this would come at the expense of added complexity. These additional connections are shown in Figure 8.1.

Since this project has focused on the lower layers and general structure of the ANS, learning has not been addressed to the same degree that creating structure has. The lower layers in any nervous system are generally less plastic (i.e. less capable of learning and more genetically defined) than the higher layers. This is the same in the ANS presented here, since the hardware of the animat will generally be fixed and pre-defined lower layer modules allow it to be functional immediately. However, to perform more intelligent tasks, the higher layers of the ANS will need to be implemented, and will also require learning capabilities.

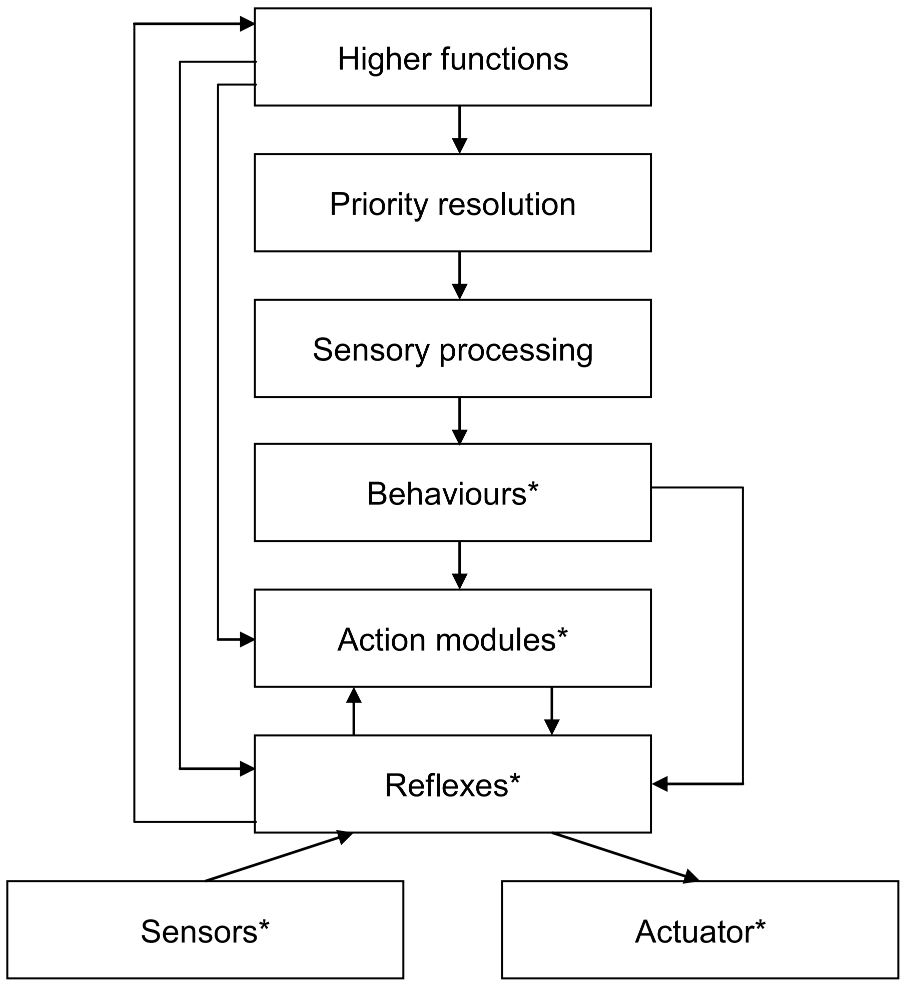


Figure 8.1: Additional connections in ANS model hierarchy for controlling highly complex systems

Learning within the layers studied during this project will be discussed in the relevant section, below.

8.3. Reflexes and the Reflex Layer

The reflex layer of the ANS contains multiple modules, operating in parallel, which provide direct low level control over the actuators of the animat.

This is different from most artificial reflexes in the existing literature, as those perform a more compound action, for example withdraw or extend an entire leg when slipping or tripping occurs during locomotion [Espenchied 1996]. Other common “high-level” reflexes found in the literature are used to alter the course of an animat performing obstacle avoidance or seeking tasks [Arkin 1995][Wikman 1996]. The implementation of these types of reflexes into the ANS was discussed in Chapter 4.

The work in this project concentrated on one type of reflex which mimicked the function of the muscle stretch reflex in biological systems, and were used to control the position of an actuator on an animat. This type of reflex was chosen since it is the most useful, from the point of view that it can be used and controlled by any layers built on top of it. If you consider the example of a high level reflex which causes an animat to dodge an obstacle, it can only ever be used in one situation – to avoid an obstacle, and are mainly triggered by external events rather than control signals from higher up in the nervous system. By creating the equivalent of the muscle stretch reflex, a more generally useful system can be made which is truly hierarchical.

Evolutionary Algorithms (EAs) were used to create the reflex, which was implemented as an Artificial Neural Network (ANN). As mentioned in section 8.2, this was primarily because the reflexes should be pre-defined so as to be immediately usable (as in nature). After consideration and experimentation, it was found that an ES (Evolution Strategy) consistently evolved the reflexes to a more optimal fitness than an EP or GA. This would be expected, as the ES uses a more balanced mix of recombination and mutation rates (as opposed to standard GAs and EP which primarily use one genetic operator), allowing for a more thorough search of the solution space.

The fitness function was not difficult to develop (although it was perhaps the most important aspect of the evolutionary system), since scoring the fitness was comprised of measurements which could be applied to traditional controllers. These included rise time (time to move actuator from a start position to a target position), offset (difference between actuator position and target position at end of test time), overshoot (maximum distance actuator is moved past target position). These measurements were used to evolve reflexes which performed the task of moving the actuator to a specified target position (and therefore controlling the position of the actuator). By altering the weightings of each fitness measurement, the actuator response could be tuned for different behaviour, such as faster rise times (but higher overshoot) or vice versa, or less oscillation of actuator position.

The system was used to evolve ANNs to act as the reflexes and was successful in this task. The performance of the best ANNs evolved was similar to that of traditional PID controllers. The main advantage of the artificial reflexes over the traditional controllers is that they are better at controlling actuators which they have not been designed for (without making any changes to the reflex or controller). Another possibility for implementing the reflexes would be to evolve PID controllers, where the search space could be reduced to three real numbers rather than one real number per weight in an ANN. This would have an advantage of being able to be evolved more quickly, although it would still have the disadvantage of lack of portability across actuators.

The ANNs evolved as reflexes used McCulloch-Pitts neurons (with sigmoid transfer function) and required a small number of them (the smallest having three inputs, three hidden layer neurons and one output) to be capable of controlling the actuator with a similar response to networks with more neurons. Connections within the ANNs were fully recurrent (as shown in Appendix C), although when only feedforward connections were used the networks only had to be slightly larger (the best being three inputs, seven hidden layer neurons and one output). The inputs to the networks were the target position for the actuator plus one or more feedback inputs from the actuator (for example, speed, position or acceleration of the actuator). The error value (difference between target position and current position of actuator) as an input was also investigated and found to perform similarly to a traditional controller which used only the proportional component of control. This suggests that the neurons were not capable of calculating the integral or derivative values to the extent that a proportional + derivative controller is, even when using fully recurrent connections. This is most likely due to the activation function of the neuron (sigmoid) as it would warp the calculated values. For example, if a linear activation was used, a neuron feeding back into itself should be able to calculate the integral of its input values.

One way in which the neuron could have been improved would be if an extra input was added for actively controlling the speed of actuator movement. Currently the reflex moves the actuator as fast as it can (assuming rise time is to be minimised in the fitness), so slower speeds could only be achieved by the layer above the reflex actively controlling the target position (and moving it in small steps).

The reflexes were tested in their ability to control two different types of actuator (an idealised linear actuator and a simulated D.C. motor) and actuators with different parameters within each type. The reflexes were capable of controlling all instances of the actuators to a satisfactory degree, although in the cases of the D.C. motors there was always a trade-off between the different components (e.g. rise time, overshoot, offset, etc.) of the actuator response.

A suitable learning algorithm could be used here to make another improvement to the reflexes. In biological systems the lower layers of the nervous system are mainly genetically defined. However, limited learning could be beneficial in the performance of the reflexes presented here. The main use of learning would be to create a universal reflex. Currently, the reflex had to be re-evolved for each actuator to achieve the optimal performance. Without changing the reflex there were slight deficiencies in the actuator response, such as longer rise times or more overshoot or oscillations. Learning could be implemented into the reflex so that a single general reflex could be evolved and then tuned on-line to the optimal configuration for any actuator.

8.4. Action Layer and CPGs

An ES was used to evolve locomotion CPGs in order to create modules for the action layer of the ANS. Initially, each CPG was responsible for creating a single gait pattern, taking tonic inputs (for stimulus) and sending outputs to one reflex per leg (of a simulated robot). Two different robots were experimented on; a biped and a quadruped, both being completely stable. This approach was used so that only the basic gait patterns would be evolved, since balance and stability are generally functions of the higher levels of the ANS. This is similar to experiments which sought to prove the existence of CPGs by suspending animals above treadmills [Grillner 1981].

The fitness functions proved difficult to formulate to achieve the desired gait patterns, which were evolved for stability (walk), simplicity (pronk), efficiency (trot and pace) and distance (gallop). All fitness scores were calculated by using the CPGs to control one reflex per leg and taking measurements from the simulated robot. This was a good way to test the integration of the system, but added to the difficulty in evolving the gaits. Had the measurements been taken from the outputs of the CPGs, and the fitness function calculated from the resulting waveforms, a more direct calculation of fitness could have been made. This would have removed the complexity of the parts of the system not directly connected with the CPGs, allowing the fitness scores to have a closer influence on the evolution of the CPG.

The CPGs were implemented as ANNs. Connections within the network were fully recurrent and inputs and outputs were applied to pre-determined (but non-specific) neurons.

CPGs were successfully evolved to produce gait patterns for walking and jumping in the bipedal robot. The ANNs for these had 4 neurons and were fully connected. Galloping, trotting, pacing and jumping gaits were successfully created for the quadruped robot, using 16 neurons; these ANNs were also fully connected. An alternative method for creating the quadruped gaits was investigated whereby the CPG was split into two functional units. The first part was the oscillator which used the previously evolved biped walk CPG. The second part was the pattern generator which converted the biped walk pattern into the quadruped gaits. Splitting the CPG in this manner allowed it to be evolved more easily than when a single network was used for the entire part.

The neurons used in the CPGs had a pulsing action. The parameters of the pulse were designed to enhance the ability of the ANN to produce appropriately timed patterns of activity. When the time parameters of the neurons are longer, they are more useful for producing the general phase patterns of the CPG waveforms. Shorter times would allow for a more finely tuned pattern of activity.

One obvious problem with the CPGs was that a walking quadruped gait was not evolved particularly successfully. Although the walk at first seems a simpler gait, it is actually more complex than, for example, a gallop (especially in this case where the robot was assumed to be stable and balanced at all times). Reducing the gait to the basic patterns of leg movement means that the walk has four distinct phases of leg movement, while all the other gaits have two. The gaits which have two phases required the legs to move back and forth at the same speed, and each phase to be evenly timed between the swing and stance. In the case of the walk, there are four phases as each leg moves separately. Each leg must also be in the stance phase longer than the swing phase, which was an added complication due to the reflex only having active control over the position of the actuator and not the speed.

Learning could probably be used at the action layer in a limited way. Taking the specific example of locomotion CPGs, learning could be used to reconfigure the CPG in the case where the gait must be permanently altered. Biological systems seem to be capable of this kind of reconfiguration instantly (which is probably due to the CPG circuits being larger than has been identified for locomotion and allowing extra neurons to contribute to the movements) [Grillner 1985]. Additionally, the cerebellum (when present) is a structure which is widely used in the learning of motor acts [Haines 1997]. Of course, this functionality may be evolved into the CPG, but that may require larger ANNs, and longer and more complicated evolution. It may be easier to evolve the CPG to perform the gait, and then reconfigure it through on-line learning if required. The other obvious use of learning would be similar to that in the reflex – to create a universal CPG. In that case, the gait produced by a standard CPG would be fine tuned through learning for whatever device it was applied to.

8.5. Summary

This chapter has discussed the investigations and experiments carried out during the project. The methods used for implementing the work have been described, and the main results have been summarised. Improvements and other issues have been described where required.

The next chapter will draw conclusions about the project as a whole with reference to the original objectives.

Chapter 9. Conclusions

9.1. Introduction

This final chapter presents the conclusions of the project. The original objectives, as specified in Chapter 1, are reviewed with reference to the work presented in previous chapters. A brief discussion of original contributions follows the conclusions, and suggestions for further work are made. Finally, some concluding remarks about the project as a whole are given.

9.2. Project Objectives Revisited

The project objectives as defined in Chapter 1 (and on the original Research Degree Application form) are listed below.

1. Literature survey of previous work in the area
2. Study of biological nervous systems and refinement of the ANS model
3. Implementation of a basic single reflex system
4. Comparison against other published work
5. Implementation of a multiple reflex system
6. Construction of an evolutionary co-ordinating network
7. Consideration of the neural functionality aspect of the network
8. System testing and comparison with published results

The following sections look at each of these objectives and considers how well they have been achieved.

9.2.1. Literature survey of previous work in the area

The study of relevant literature was performed throughout the duration of the research. A wide range of topics (nervous systems - biological and artificial - biological physiology, evolutionary algorithms and artificial neural networks to name four of the most important) are encompassed by it. It was the sole activity for the first four months, and continued at a lower level for the remainder of the project. Most of the literature is reviewed in Chapter 3, although references are made to appropriate material throughout the thesis.

9.2.2. Study of biological nervous systems and refinement of the ANS model

The study of biological nervous systems was performed as part of the study of relevant literature, and was accompanied in parts by study of biological physiological systems. Further ideas were gained about the structure of nervous systems of varying complexity (from lower invertebrates to higher vertebrates) and areas of the ANS model which required refining were identified.

The first change made to the original model was the identification of reflexes as primarily being general purpose and lower level control systems, rather than reactive actions (such as obstacle avoiding movements or trip correction). Those reactive or reflex actions could be implemented as reflexes or, more likely, as actions.

Another refinement of the model was the inclusion of a behavioural layer. This consists of modules which would represent innate behaviours in biological nervous systems (and are therefore available without the need to learn them).

The final refinement was the realisation that for more complex nervous systems, or when the higher layers were implemented, there would be more connections between the layers and not simply connections between adjacent layers as originally proposed. The higher layers would require connections to, for example, the reflex layer to be capable of performing skilled movements that were not already built into other layers of the ANS. The behavioural layer would also require connections directly to the reflexes, allowing the behaviours to be sequences of reflexes and actions.

The fully updated ANS model is shown in Chapter 2, in Figure 2.4.

9.2.3. Implementation of a basic single reflex system

A basic single reflex system was created using EAs to evolve ANNs. The single reflex had the task of controlling the position of an actuator on an animat. Chapter 4 describes the ideas and biological background to the reflex system and chapter 5 presents the implementation details and results of the basic single reflex system.

It was found that using an ES consistently produced reflexes with better fitness scores. The use of an elitist rather than a non-elitist ES produced more optimal reflexes.

It can be concluded that the fitness score was the most important part of the EA for producing reflexes which could successfully control actuators. The fitness measurements were based on the actuator response, and the way the reflex affected the response. By weighting these measurements it was possible to control which of them were optimised.

We can also conclude that an ANN with recurrent connections requires the least number of neurons to be able to control an actuator satisfactorily. ANNs which had only forward connections required a slightly higher number of neurons, while ANNs with connections only between adjacent layers required at least twice as many hidden layer neurons than the fully recurrent network.

Finally, it has been found that the reflexes can be applied to different actuators (from the actuators they were designed for) with less of a degradation in performance than traditional controllers.

9.2.4. Comparison against other published work

This was obviously tied to the objective described in section 9.2.3 and was presented along with the results of that objective in Chapter 5. The work was mainly compared to traditional methods of implementing low level control over actuators, as these can be considered the standard approach. The reflexes performed at least on a par with the traditional controllers, although had the advantage that they were better at controlling actuators which they had not been optimised for. Comparison was also made to other appropriate work which used AI techniques to control actuators.

9.2.5. Implementation of a multiple reflex system

This objective turned out to be trivial due to the nature of the reflexes: they worked independently and in parallel and were controlled by the layer above through a well defined set of inputs. The multiple reflex system was created by simply replicating the single reflex system a number of times and processing each reflex on its own. However, the correct functioning of the multiple reflex system became an important basis for the next objective.

9.2.6. Construction of an evolutionary co-ordinating network

This objective became vastly more important after it was realised that the multiple reflex system was trivial to implement and that the main focus would be the co-ordination between the reflexes. The co-ordinating network was implemented as a module from the action layer which acted as a CPG for controlling locomotion. Further investigation was carried out across a number of different locomotion patterns for different robots. Chapters 6 and 7 contain background ideas, implementation and results of the co-ordinating networks.

Low numbers of computationally simple neurons, with evolved connections between them, can be used as CPGs to produce various gait patterns. The same methods can be applied to the evolution of different gaits for different robot configurations.

The number of neurons and evolutionary generations required to evolve a gait can be reduced by keeping the CPGs modular. In the case of the results presented in Chapter 7, this modularity was achieved by separating the CPGs into two functional units (an oscillator and a pattern generator).

The complexity of the evolutionary task depends on the number of phases in the gait (with one phase in a pronk, two in the gallop, trot and pace and four phases in the walk).

9.2.7. Consideration of the neural functionality aspect of the network

The functionality of the network components (neurons and synapses) were reviewed before implementing the action network. An original model for the network components was designed so that CPGs could be easily created.

This objective was not only limited to the neural functionality in the action layer networks. The discussion presented in chapter 8 included ideas about the functioning of the neurons used in the reflexes.

9.2.8. System testing and comparison with published results

The system which was implemented (CPGs producing locomotion patterns and controlling reflexes) was tested to see if it could produce the desired gait patterns correctly.

The ideas behind the ANS in general (flexible, modular, extensible) and the structure and function of it were compared to alternative robot control systems in Chapter 3.

9.3. Original Contributions

Although the use of EAs to create ANNs is not new, the ANS structure which is being populated with these networks is original. The aim is for a flexible, modular and extensible system inspired by simple biological nervous systems which has so far been implemented through the use of simple, efficient neural networks with low numbers of neurons. The ANNs used have well defined inputs and outputs, making it easy to keep the ANS structure modular. Since the ANNs developed in this work have no coupling to each other (apart from through the hierarchy of the ANS), the system can be extended easily.

Consideration of the functionality of the proposed ANS has been made and the proposals for refining and modifying it to improve its future abilities have been given in this thesis. Those proposed refinements are unique to this thesis. The first change was the functionality of the reflex layer. Previously the reflexes represented movements which were initiated by events external to the body (e.g. pain withdrawal), but are now completely controlled by higher layers of the ANS (equivalent to e.g. muscle stretch reflex). The exact specification of the interfaces between the reflex, action and higher layers has been specified in this thesis, which had only previously been a rough organisational overview. Additions to the ANS structure made in this thesis include the identification of a possible behavioural layer and the need for extra connections between different layers (rather than simply adjacent layers being connected).

This is the first project which has investigated and applied the original ANS model which makes it original work.

Investigations into the operation of the reflex layer have yielded sets of results and guidelines, which can be used as a basis for creation of other reflexes. In this work, different EAs have been compared as to how well they can evolve ANNs for the task of controlling an actuator. Consideration has been given to different methods of formulating the fitness function, and what affects changes to this would have on the final solution. Different network topologies have been studied, with recommendations for sizes given different connection strategies. The reflexes have also been tested on their ability to generalise across different actuators with proposals for future work to enhance their generalising ability.

The models used for the network components (neurons and synapses, presented in Chapter 6) of the CPG networks are also original. The parameters which were used to define the behaviour of the components and the models being optimised to produce arbitrary patterns of activity through time, in a simple yet elegant manner, make these models original. The research conducted into these CPGs demonstrates the improvements achievable by modularising the neural networks. These improvements can be seen in the optimisation ability of the EA and the final performance of the evolved systems.

9.4. Further Work

There are four main areas in which further work could be carried out to extend this project. The main topic would be automatic definition or evolution of the ANS structure. The next two areas would be further horizontal and vertical implementation of the layers. The final area would be to enhance the work presented in this thesis. These are explained in more detail below.

9.4.1. Automatic definition of ANS structure

The approach taken throughout this project was that the structure of the ANS was defined manually (and any constraints applied); the EAs only worked within single modules (for example a single reflex, or the CPG to produce biped walking gaits). The next logical step forward would be a system which could be evolved from scratch. The modularity, ANS and module function would be defined automatically (i.e. without any design intervention) in the final chromosomes. Another research student in the School of Engineering at The Robert Gordon University will be investigating this area of further work.

There are no current EAs which are completely suitable for this type of task, so further ideas have been taken from nature and how it creates modular systems. These are described in the paper presented in Appendix B. This paper also discusses the importance of different features in networks, such as neuron parameters and functions, network organisation, learning and evolution.

9.4.2. Extend ANS vertically

An obvious piece of work to be carried out in the future is to implement the ANS layers higher in the structure than those studied in this project. Another PhD candidate (in the School of Engineering at the Robert Gordon University) has already begun work which concentrates on the sensory processing layers. In time, the action layers and sensory layers will be connected to each other to allow testing of the ANS with a more complete structure. Even further work would include researching the layers above the sensory processing.

9.4.3. Expand ANS horizontally

Another suggestion, complementary to the previous one, is to expand the ANS horizontally. This would involve adding more reflexes (to enable control of more actuators) and action layer modules (not only CPGs for locomotion) to the lowest two layers. Extending the ANS horizontally would also require at least some behavioural layer modules to be implemented, in order to control the newly created actions and reflexes. This line of work could be used to investigate animats with multiple degrees of freedom in each limb.

9.4.4. Extend the implementation of work presented in this thesis

The final suggestion for further work is to extend the implementation of the work on the modules investigated during this project. A truly general purpose reflex, that learned to adapt to the actuator it was used with, and was able to control speed as well as position would be an interesting and challenging piece of further work. More research can also be carried out into the topology of the reflex networks. As shown in Chapter 5 and Appendix C one of the reflexes has more neurons than was required. This can lead to over-fitting of the network to the problem and lack of generalisation. Smaller networks should be investigated in order to try to avoid these problems.

In terms of extending the work that was carried out in the action layer, there are a few options. The first is to evolve the walking CPG to produce that gait correctly. The second is to investigate the possibility and feasibility of one CPG performing more than one or all gaits for locomotion. Initial work had begun on creating CPGs which would be able to perform two different gaits, but this was stopped due to lack of time. The final suggestion is to enable the CPGs to control multiple degrees of freedom in the actuators for locomotion.

9.5. Concluding Remarks

The project has been successful in that all the objectives have been met, and that the scheme has been implemented and worked as expected.

The ANS presents a well defined modular architecture for creating robot controllers. The lower layers were implemented in this project successfully and, due to the flexible and modular structure, minimised the amount of changes to the existing layers when using different robots. Apart from difficulties in formulating the fitness functions for the EAs when evolving the quadruped CPGs, the creation of modules worked well.

This thesis now joins a body of other research describing the implementation of systems based on AI techniques (in this case Evolutionary ANNs) for low level robotic control.

References

Chapter 1

MacLeod, C., 1999, The Synthesis of Artificial Neural Networks Using Single String Evolutionary Techniques. PhD Thesis, The Robert Gordon University, Aberdeen, UK.

Chapter 2

Arbib, M. A., 1989, The Metaphorical Brain 2: Neural Networks and Beyond. Wiley.

Baeck, T., 1995, Evolutionary Algorithms in Theory and Practice. Oxford University Press.

Buchsbaum, R., 1968, Animals without Backbones: 1. Penguin Books, pp. 93-96.

Darwin, C., 1859, The Origin of Species by Natural Selection or the Preservation of Favoured Races in the Struggle for Life. John Murray.

Eberhart, R. C., Dobbins, R. W., 1991, Designing Neural Network explanation facilities using Genetic Algorithms. IEEE Joint Conference on Neural Networks, vol. 2, pp. 1758-1763.

Eldredge, N., Gould, S. J., 1972, Punctuated equilibria: An alternative to phyletic gradualism. In Schopf, T. J. M. (ed), Models in Paleobiology, Freeman & Cooper, pp. 82-115.

Ganong, W. F., 1995, Review of Medical Physiology. Appleton & Lange, pp. 84-101.

Haupt, R. L., Haupt, S. E., 1998a, Practical Genetic Algorithms. Wiley & Sons Inc., pp. 32-34.

Haupt, R. L., Haupt, S. E., 1998b, Practical Genetic Algorithms. Wiley & Sons Inc., p. 52.

Hebb, D., 1949, The Organisation of Behaviour. Wiley.

Hodgkin, A. L., Huxley, A. F., 1952, A quantitative description of membrane current and its application to conduction and excitation in a nerve. Journal of Physiology, vol. 177, pp. 500-544.

Holland, J. H., Arber, A., 1975, Adaptation in natural and artificial systems. The University of Michigan Press.

Hopfield, J. J., 1982, Neural Networks and Physical Systems with Emergent Collective Computational Properties. Proc. of the National Academy of Sciences of the USA, vol. 79, pp. 2554-2588.

Jenkins, M., 1998, Teach Yourself Genetics. Hodder & Stoughton, pp. 18-30.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., 1991, Principles of Neural Science 3rd Edition. Appleton & Lange, p. 24.

Kelly, J. D., Davis, L., 1991, Hybridising the Genetic Algorithm and the K nearest neighbours classification algorithm. INCGA91, pp. 337-383.

Levitan, I. B., Kaczmarek, L. K., 1997, The Neuron: Cell and Molecular Biology 2nd Edition. Oxford University Press, pp. 149-152.

MacLeod, C., Maxwell, G. M., McMinn, D., 1998, A Framework for Evolution of an Animat Nervous System. EUREL European Advanced Robotics Systems Development: Mobile Robotics, Leiria, Portugal.

McCulloch, W., Pitts, W., 1943, A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Methematical Biophysics, vol. 5, pp. 115-133.

Miller, G., Todd, P., Hedge, S., 1989, Designing Neural Networks Using Genetic Algorithms. Proc. 3rd International Conference on Genetic Algorithms, pp. 379-384.

Minsky, M. L., Papert, S. A., 1969, Perceptrons. MIT Press.

Montana, D. J., Davis, L., 1989, Training Feedforward Neural Networks Using Genetic Algorithms. Proc. 11th International Joint Conference on Artificial Intelligence, pp. 762-767.

Rosenblatt, F., 1962, Principles of Neurodynamics. Spartan Books.

Sarnat, H. B., Netsky, M. G., 1981, Evolution of the Nervous System. Oxford University Press, pp. 40-41.

Schaffer, J. D., Caruna, R. A., Eshelman, L. J., 1990, Using Genetic Search to Exploit the Emergent Behaviour of Neural Networks. In Forest, S. (ed), Proc. of 1989 Conference on Emergent Behaviour, pp 244- 248.

Schwefel, H. P., 1995, Evolution and Optimum Searching. Wiley Interscience.

Stanley, S. M., 1975, A theory of evolution above the species level. Proceedings of the National Academy of Science USA, vol. 72, pp. 646-650.

Wasserman, P. D., 1989a, Neural Computing: Theory and Practice. Von Nostrand Reinhold.

Wasserman, P. D., 1989b, Neural Computing: Theory and Practice. Von Nostrand Reinhold, pp. 63-71.

Wasserman, P. D., 1989c, Neural Computing: Theory and Practice. Von Nostrand Reinhold, pp. 77-92.

Werbos, P., 1974, Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences. PhD thesis, Harvard University, USA.

Yao, X., 1999, Evolving Artificial Neural Networks. Proceedings of the IEEE, vol. 87, part 9, pp. 1423-1447.

Chapter 3

Akiyama, S., Kimura, H., 1995, Dynamic Quadruped Walk Using Neural Oscillators – Realization of Pace and Trot. Proc. 13th Annual Conference of RSJ, pp. 227-228.

Albus, J. S., 1975, A New Approach to Manipulator Control: the Cerebellar Model Articulation Controller (CMAC). ASME Journal of Dynamic Systems, Measurements and Control, September, pp. 220-227.

Arkin, R. C., 1995, Reactive Robotic Systems. In Arbib, M. (ed.), Handbook of Brain Theory and Neural Networks, MIT Press, pp. 793-796.

Bartolini, G., Cannata, G., Casalino, G., Ferrera, A., 1995, A Hierarchical Control Architecture for the Control of Underwater Robots. Proc. IFAC Workshop on Control Applications in Marine Systems, vol. 3, pp. 60-65.

Beer, R. D., Chiel, H. J., Sterling, L. S., 1991, An Artificial Insect. American Scientist, vol. 79, pp. 444-452.

Benbrahim, H., Franklin, J. A., 1997, Biped Dynamic Walking Using Reinforcement Learning. Robotics and Autonomous Systems, vol. 22, pp. 283-
302.

Berkemeier, M. D., 1995, Coupled Oscillators in a Platform With Four Springy Legs and Local Feedback. Proc. 34th IEEE Conference on Decision and Control, vol. 3, pp. 2847-2852.

Brooks, R. A., 1986, A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, vol. 2, part 1, pp. 14-23.

Buchanan, J. T., Grillner, S., 1987, Newly Identified 'Glutamate Interneurons' and Their Role in Locomotion in the Lamprey Spinal Cord. Science, vol. 236, pp. 312-314.

Buchanan, J. T., 1992, Neural Network Simulations of Coupled Locomotor Oscillators in the Lamprey Spinal Cord. Biological Cybernetics, vol. 66, pp. 367-374.

Bullock, D., Contras-Vidal, J. L., 1993, How Spinal Neural Networks Reduce Discrepancies Between Motor Intention and Motor Realization. In Newell, K. M., Corcos, D. M. (eds.), Variability and Motor Control, pp. 183-221.

Cao, M., Kawamura, A., 1999, A design scheme of neural oscillatory networks by hierarchical evolutionary calculation for generation of humanoid biped walking patterns. Advanced Robotics, vol. 12, part 7/8, pp. 697-710.

Connell, J. H., 1998, A Behaviour Based Arm Controller. MIT AI Laboratory Memo 1025, http://www.ai.mit.edu/research/publications/publications.shtml.

Damper, R. I., Scutt, T. W., 1998, Biologically-Based Learning in the Arbib Autonomous Robot. IEEE International Joint Symposium on Intelligence and Systems, pp. 49-56.

Dellaert, F., Beer, R., 1996, A Developmental Model for the Evolution of Complete Autonomous Agents. Simulation of Adaptive Behaviour: From Animals to Animats, vol. 4, pp. 393-401.

Digney, B. L., 1997, Learning and Shaping of Hierarchical Control Structures. Proc. 7th Topical Meeting on Robotics and Remote Systems, vol. 1, pp. 30-37.

Ekeberg, O., Wallen, P., Lansner, A., Traven, H., Brodin, L., Grillner, S., 1991, A Computer Based Model for Realistic Simulations of Neural Networks. Biological Cybernetics, vol. 65, pp. 81-90.

Ekeberg, O., 1993, A Combined Neuronal and Mechanical Model of a Fish Swimming. Biological Cybernetics, vol. 69, pp. 363-374.

Espenchied, K. S., Quinn, R. D., Beer, R. D., Chiel, H. J., 1996, Biologically Based Distributed Control and Local Reflexes Improve Rough Terrain Locomotion in a

Hexapod Robot. Robotics and Autonomous Systems, vol. 18, pp. 59-64.

Ferrell, C., 1995, Comparison of Three Insect-Inspired Locomotion Controllers. Robotics and Autonomous Systems, vol. 16, pp. 135-159.

de Garis, H., 1991a, Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules. Parallel Problem Solving from Nature, Lecture Notes in Computer Science 496, Springer-Verlag.

de Garis, H., 1991b, The Lizzy Project: Genetically Programming an Artificial Nervous System. International Conference on Artificial Neural Networks, Espoo, Finland.

de Garis, H., 1994, An Artificial Brain : ATR's cam-brain project aims to build/evolve an artificial brain with a million neural net modules inside a trillion cell cellular automata machine. New Generation Computing Journal, vol. 12, part 2.

de Garis, H., 1995, CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 which Grows/Evolves at Electronic Speeds inside a Cellular Automata Machine (CAM). Proc. International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA95).

de Garis, H., Korkin, M., Gers, F., Nawa, E., Hough, M., 2000, Building an artificial brain using an FPGA based CAM-Brain Machine. Applied Mathematics and Computation, vol. 111, pp. 163-192.

Gat, E., 1998, Three Layer Architectures. In Kortenkamp, D., Bonasso, R. P., Murphy, R. (eds), Artificial Intelligence and Mobile Robots, pp. 195-210.

Gaudiano, P., Zalama, E., Chang, C., Coronado, J. L., 1996, A Model of Operant Conditioning for Adaptive Obstacle Avoidance. 4th International Conference on Simulation of Adaptive Behaviour: From Animals to Animats, vol. 4, pp. 373-381.

Golubitsky, M., Stewart, I., Buono, P-L., Collins, J. J., 1998, A modular network for legged locomotion. Physica D, vol. 115, pp. 56-72.

Grillner, S., 1981, Control of Locomotion in Bipeds, Tetrapods and Fish. In Brooks, V., B. (ed.), Handbook of Physiology: The Nervous System II, American Physiological Society, pp. 179-1236.

Grillner, S., Wallen, P., Viana di Prisco, G., 1990, Cellular Network Underlying Locomotion as Revealed in a Lower Vertebrate Model: Transmitters, Membrane Properties, Circuitry, and Simulation. Cold Spring Harbour Symposium on Quantitative Biology, vol. 55, pp. 779-789.

Hartley, R., Pipitone, F., 1991, Experiments with the Subsumption Architecture. Proc. IEEE International Conference on Robotics and Automation, pp. 1652-1658.

van Heijst, J. J., Vos, J. E., Bullock, D., 1998, Development in a Biologically Inspired Spinal Neural Network for Movement Control. Neural Networks, vol. 11, pp. 1305-1316.

Herbert, T., Valavanis, K., Kolluru, R., 1998, A Real-Time Hierarchical Sensor-Based Robotic System Architecture. Journal of Intelligent and Robotic Systems, vol. 21, pp. 1-27.

Hunt, K. J., Sbarbaro, D., 1995, Studies in Artificial Neural Network Based Control. In Irwin, G. W., Warwick, K., Hunt, K. J. (eds), Neural Network Applications in Control, The Institution of Electrical Engineers, London, pp. 109-139.

Ijspeert, A. J., Hallam, J., Willshaw, D., 1997, Artificial Lampreys: Comparing Naturally and Artificially Evolved Swimming Controllers. European Conference on Artificial Life, vol. 4, pp. 256-265.

Ijspeert, A. J., Hallam, J., Willshaw, D., 1998, From Lampreys to Salamanders: Evolving Neural Controllers for Swimming and Walking. Simulation of Adaptive Behaviour: From Animals to Animats, vol. 5, pp. 390-399.

Izumi, K., Watanabe, K., 1997, Fuzzy behaviour-based tracking control for a mobile robot. Proc. 2nd Asian Control Conference, vol. 1, pp. 685-688.

Izumi, K., Watanabe, K., 2000, Fuzzy behaviour-based control trained by module learning to acquire the adaptive behaviours of mobile robots. Mathematics and Computers in Seimulation, vol. 51, pp. 233-243.

Jacob, C., 1994, Evolutionary Programming of Artificial Nervous System. 39 Internales Wissenschatliches Kolloquium, vol. 2, pp. 31-38.

Jarandanan, E. G., Gajendran, F., 1998, Artificial Neural Network Based Control Scheme for a DC Motor. Institution of Engineers (India) Journal - Part EL, vol. 79, pp 72-76.

Kimura, H., Sakurama, K., Akiyama, S., 1998, Dynamic walking and running of the quadruped using neural oscillator. Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 50-57.

Kodjabachian, J., Meyer, J.-A., 1995, Evolution and Development of Control Architectures in Animats. Robotics and Autonomous Systems, vol. 16, pp. 161-182.

Lewis, M. A., Fagg, A. H., Solidum, A., 1992, Genetic Programming Approach to the Construction of a Neural Network for Control of a Walking Robot. Proc. IEEE International Conference on Robotics and Automation, pp. 2618-2623.

Maes, P, 1992, Learning behaviour networks from experience. Proc. 1st European Conference on Artificial Life (ECAL91), pp. 48-57.

Mataric, M. J., 1992, Integration of Representation Into Goal-Driven Behaviour-Based Robots. IEEE Transactions on Robotics and Automation, vol. 8, part 3, pp. 304-312.

Maurer, M., Dickmanns, E. D., 1997, An Advanced Control Architecture for Autonomous Vehicles. Proc. SPIE , vol. 3087, pp. 94-105.

Millan, J. del R., 1996, Rapid, Safe, and Incremental Learning of Navigation Strategies. IEEE Transactions on Systems, Man, and Cybernetics – Part B:

Cybernetics, vol. 26, part 3, pp. 408-420.

Miyakoshi, S., Taga, G., Kuniyoshi, Y., Nagakubo, A., 1998, Three dimensional bipedal stepping motion using neural oscillators – towards humanoid motion in the real world. Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 84-89.

Nakashima, H., Noda, I., 1998, Dynamic Subsumption Architecture for Programming Intelligent Agents. Proc. International Conference on Multiagent Systems, pp. 190-197.

Prentice, S. D., Patla, A. E., Stacey, D. A., 1995, Modelling the Timekeeping Function of the Central Pattern Generator for Locomotion Using Artificial Sequential Neural Network. Medical and Biological Engineering and Computing, vol. 33, pp. 317-322.

Prentice, S. D., Patla, A. E., Stacey, D. A., 1998, Simple Artificial Neural Network Models can Generate Basic Muscle Activity Patterns for Human Locomotion at Different Speeds. Experimental Brain Research, vol. 123, part 4, pp. 474-480.

Reeve, R., 1999, Generating Walking Behaviours in Legged Robots. PhD thesis, University of Edinburgh.

Rosenblatt, J. K., Payton, D. W., 1989, A fine-grained alternative to the Subsumption Architecture for Mobile Robot Control. Proc. IEEE/INNS International Joint Conference on Neural Networks, vol. 2, pp. 317-324.

Rovithakis, G. A., Christodoulou, M. A. 2000, Adaptive control with recurrent high-order neural networks: Theory and industrial applications. Springer-Verlag, pp. 1-6.

Saridis, G. N., 1992, Architectures for Intelligent Control. Proc. International Symposium on Implicit and Non-linear Systems (SINS92).

Scutt, T. W., Damper, R. I., 1997, Biologically Motivated Learning in Adaptive Mobile Robots. IEEE International Conference on Systems, Man, and Cybernetics - Computational Cybernetics and Simulation, vol. 1, pp. 475-480.

Shik, M. L., Orlovsky, G. N., 1976, Neurophysiology of Locomotor Automatism. Physiological Reviews, vol. 56, part 3, pp. 465-501.

Smith, R.L., Coghill, G. C., 1996, An Autonomous Robot Controller with Learned Behaviour. The Australian Journal of Intelligent Information Processing Systems, vol. 3, number 2.

Sousa, J. B., Perera, F. L., da Silva, E. P., Martins, A., Matos, A., Almeida, J., Cruz, N., Tunes, R., Cunha, S., 1996, On the Design and Implementation of a Control Architecture for a Mobile Robotic System. Proc. IEEE International Conference on Robotics and Automation, vol. 3, pp. 2822-2827.

Stewart, I., 1999, Designer Differential Equations for Animal Locomotion. Complexity, vol. 5, part 2, pp. 12-22.

Taga, G., Yamaguchi, Y., Shimizu, H., 1991, Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. Biological Cybernetics, vol. 65, pp. 147-159.

Todd, D. J., 1986, Fundamentals of Robot Technology. Anchor Brendon Ltd, p. 10.

Venkataraman, S. T., 1997, A simple legged locomotion gait model. Robotics and Autonomous Systems, vol. 22, pp. 75-85.

Wadden, T., Grillner, S., Matsushima, T., Lansner, A., 1993, Undulatory Locomotion - Simulations with Realistic Segmental Oscillator. In Eeckman, F. M., Bower, J. M. (eds), Computations & Neural Systems.

Wikman, T. S., Newman, W. S., 1996, Reflex Control for Safe Autonomous Robot Operation. Reliability Engineering and System Safety, vol. 53, pp. 339-347.

Wu, C., Hwang, K.-S., Chang, S.-L., 1997, Analysis and Implementation of a Neuromuscular-Like Control for Robotic Compliance. IEEE Transactions on Control Systems Technology, vol. 5, part 6, pp. 586-597.

Zeldman, M. I., 1984, What every Engineer should know about robots. Marcel Dekker Inc, pp. 1-12.

Chapter 4

Baeck, T., Schwefel, H.-P., 1996, Evolutionary Computation: An Overview. Proc. 3rd IEEE Conference on Evolutionary Computation, pp. 20-29.

Brodal, P., 1992, The Central Nervous System: Structure and Function. Oxford University Press, p. 229.

Buchsbaum, R., 1968, Animals without Backbones: 1. Penguin Books, p. 92.

Ganong, W. F., 1995a, Review of Mediacal Physiology 17th Edition. Appleton & Lange, p. 59.

Ganong, W. F., 1995b, Review of Mediacal Physiology 17th Edition. Appleton & Lange, p. 61.

Haines, D. E. (ed), 1997a, Fundamental Neuroscience. Churchill Livingstone Inc, pp. 135-139.

Haines, D. E. (ed), 1997b, Fundamental Neuroscience. Churchill Livingstone Inc, pp. 339-340.

Holland, J. H., Arber, A., 1975, Adaptation in natural and artificial systems. The University of Michigan Press.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., 1991a, Principles of Neural Science 3rd Edition. Appleton & Lange, p. 556.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., 1991b, Principles of Neural Science 3rd Edition. Appleton & Lange, pp. 560-561.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., 1991c, Principles of Neural Science 3rd Edition. Appleton & Lange, pp. 565-570.

Chapter 6

Baeck, T., Schwefel, H.-P., 1995, Evolution Strategies I: Variants and their computational implementation. In Winter, G., Périeaux, J., Galá, M., Cuesta, P. (eds), Genetic Algorithms in Engineering and Computer Science, Wiley, pp. 111-126.

Brodal, P., 1992, The Central Nervous System: Structure And Function. Oxford University Press, pp 21-34.

Carpenter, R. H. S., 1990, Neurophysiology 2nd Edition. Edward Arnold, pp. 242-244.

Friesen, W. O., Stent, G. S., 1978, Neural Circuits for Generating Rhythmic Movements. Annual Reviews of Biophysiology and Bioengineering, vol. 7, pp. 37-61.

Getting, P. A., 1988, Analysis of Central Pattern Generators. In Cohen, A., Rossignol, S., Grillner, S. (eds), Neural Control of Rhythmic Movements in Vertebrates, Wiley & Sons, pp. 101-127.

Golubitsky, M., Stewart, I., Buono, P-L., Collins, J. J., 1998, A Modular Network For Legged Locomotion. Physica D, vol. 115, pp. 56-72.

Gray, J., 1968a, Animal Locomotion. Weidenfeld and Nicolson, pp. 267-276.

Gray, J., 1968b, Animal Locomotion. Weidenfeld and Nicolson, pp. 282-286.

Grillner, S., 1974, On the Generation of Locomotion in the Spinal Dogfish. Experimental Brain Research, vol. 20, pp. 459-470.

Grillner, S., 1981, Control of Locomotion in Bipeds, Tetrapods, and Fish. In Brooks, V. B. (ed), Handbook of Physiology: The Nervous System II, Williams & Wilkins, pp. 1179-1236.

Grillner, S., Wallen, P., 1995, Central Pattern Generators For Locomotion, With Special Reference To Vertebrates. Annual Review of Neuroscience, vol. 8, pp 231-261.

Ijspeert, A., Hallam, J., Willshaw, D., 1998, From lampreys To Salamanders: Evolving Neural Controllers For Swimming and Walking: From Animals to Animats. Proc. 5th International Conference of The Society for Adaptive Behavior (SAB98), pp. 390-399.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., 1991a, Principles of Neural Science 3rd Edition. Appleton & Lange, p. 627.

Levitan, I. B., Kaczmarek, L. K., 1997a, The Neuron 2nd Edition. Oxford University Press, pp. 464-466.

Levitan, I. B., Kaczmarek, L. K., 1997b, The Neuron 2nd Edition. Oxford University Press, pp. 454-456.

MacLeod, C., McMinn, D., Reddipogu, A. B., Capanni, N. F., Maxwell, G. M., 2001, Evolution and Devolved Action. See Appendix B.

Prentice, S. D., Patla, A. E., Stacey, D. A., 1995, Modelling The Time-Keeping Function of the Central Pattern Generator for Locomotion Using Artificial Sequential Neural Network. Medical & Biological Engineering & Computing, vol. 33, pp. 317-322.

Schwefel, H.-P., 1995, Evolution and Optimum Searching. Wiley.

Shik, M. L., Severin, F. V., Orlovsky, G. N., 1966, Control Of Walking And Running By Means Of Electrical Stimulation Of The Mid-Brain. Biophysics vol. 11, pp. 756-765.

Shik, M. L., Orlovsky, G. N., 1976, Neurophysiology of Locomotor Automatism. Physiological Reviews, vol. 56, part 3, pp. 465-501.

Stewart, I., 1998, Life's Other Secret. Penguin, pp. 173-193.

Wallen, P., Grillner, S., 1987, N-methyl-D-aspartate Receptor-Induced Inherent Oscillatory Activity in Neurons Active During Fictive Locomotion in the Lamprey. Journal of Neuroscience, vol. 7, pp. 2745-2755.

Chapter 7

Carpenter, R. H. S., 1990, Neurophysiology 2nd Edition. Edward Arnold, pp. 242-244.

Gray, J., 1953, How Animals Move. Cambridge University Press, pp. 25-27.

Grillner, S., 1981, Control of Locomotion in Bipeds, Tetrapods, and Fish. In Brooks, V. B. (ed), Handbook of Physiology: The Nervous System II, Williams & Wilkins, pp. 1179-1236.

Grillner, S., Wallen, P., 1985, Central Pattern Generators For Locomotion, With Special Reference To Vertebrates, Annual Review of Neuroscience, vol. 8, pp 231-261.

Golubitsky, M., Stewart, I., Buono, P-L., Collins, J. J., 1998, A Modular Network For Legged Locomotion. Physica D, vol. 115, pp. 56-72.

MacLeod, C., McMinn, D., Reddipogu, A. B., Capanni, N. F., Maxwell, G. M., 2001, Evolution and Devolved Action. See Appendix B.

Prentice, S. D., Patla, A. E., Stacey, D. A., 1995, Modelling The Time-Keeping Function of the Central Pattern Generator for Locomotion Using Artificial Sequential Neural Network. Medical & Biological Engineering & Computing, vol. 33, pp. 317-322.

Stewart, I., 1998, Life's Other Secret. Penguin, pp. 173-193.

Stewart, I., 1999, Designer Differential Equations for Animal Locomotion. Complexity, vol. 5, pp. 12-22.

Chapter 8

Arkin, R. C., 1995, Reactive Robotic Systems. In Arbib, M. (ed), Handbook of Brain Theory and Neural Networks, MIT Press, pp. 793-796.

Brooks, R. A., 1986, A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, vol. 2, part 1, pp. 14-23.

Bartolini, G., Cannata, G., Casalino, G., Ferrera, A., 1995, A Hierarchical Control Architecture for the Control of Underwater Robots. Proc. IFAC Workshop on Control Applications in Marine Systems, vol. 3, pp. 60-65.

Espenchied, K. S., Quinn, R. D., Beer, R. D., Chiel, H. J., 1996, Biologically Based Distributed Control and Local Reflexes Improve Rough Terrain Locomotion in a Hexapod Robot. Robotics and Autonomous Systems, vol. 18, pp. 59-64.

Gat, E., 1998, Three Layer Architectures. In Kortenkamp, D., Bonasso, R. P., Murphy, R. (eds), Artificial Intelligence and Mobile Robots, pp. 195-210.

Grillner, S., 1981, Control of Locomotion in Bipeds, Tetrapods, and Fish. In Brooks, V. B. (ed), Handbook of Physiology: The Nervous System II, Williams & Wilkins, pp. 1179-1236.

Grillner, S., 1985, Neurobiological Bases of Rhythmic Motor Acts in Vertebrates. Science, vol. 228, pp. 143-149.

Haines, D. E. (ed), 1997, Fundamental Neuroscience. Churchill Livingstone Inc, pp. 397-398.

Herbert, T., Valavanis, K., Kolluru, R., 1998, A Real-Time Hierarchical Sensor-Based Robotic System Architecture. Journal of Intelligent and Robotic Systems, vol. 21, pp. 1-27.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., 1991, Principles of Neural Science 3rd Edition. Appleton & Lange, pp. 537-539.

MacLeod, C., Maxwell, G. M., McMinn, D., 1998, A Framework for Evolution of an Animat Nervous System. EUREL European Advanced Robotics Systems Development: Mobile Robotics, Leiria, Portugal.

MacLeod, C., 1999, The Synthesis of Artificial Neural Networks Using Single String Evolutionary Techniques. PhD Thesis, The Robert Gordon University, Aberdeen, UK.

Maurer, M., Dickmanns, E. D., 1997, An Advanced Control Architecture for Autonomous Vehicles. Proc. SPIE, vol. 3087, pp. 94-105.

Sousa, J. B., Perera, F. L., da Silva, E. P., Martins, A., Matos, A., Almeida, J., Cruz, N., Tunes, R., Cunha, S., 1996, On the Design and Implementation of a Control Architecture for a Mobile Robotic System. Proc. IEEE International Conference on Robotics and Automation, vol. 3, pp. 2822-2827.

Wikman, T. S., Newman, W. S., 1996, Reflex Control for Safe Autonomous Robot Operation. Reliability Engineering and System Safety, vol. 53, pp. 339-347.