The Evolution of Modular Artificial Neural Networks

A thesis submitted to
The Robert Gordon University
in partial fulfilment of the requirements for
the degree of Doctor of Philosophy

Sethuraman Muthuraman

School of Engineering
The Robert Gordon University
Aberdeen, Scotland, 2005

Acknowledgements

Firstly, I would like to thank my mother, Mrs Ramayee Muthuraman, for the love and encouragement she has given me. Her faith has been a great inspiration in compilation of this thesis. I would like to express gratitude to my younger brother Mr Valliappan Muthuraman who assists my mother in every respect while I am away from home.

Secondly, I am indebted to both my supervisors Mr Grant Maxwell and Dr Christopher MacLeod for their assistance and advice over the duration of the project. Without their supervision skills, the project would have been much more difficult and not nearly as enjoyable.

Thirdly, I am indebted to The Robert Gordon University for the award of a Research Studentship.

A special note of thanks is due to Mr Matthew G Crowley, who assisted in proof reading the thesis and made many valuable suggestions during the project and for his friendship.

Thanks to Dr Christopher MacLeod, Dr David McMinn and Mrs Ann B Reddipogu for their permission to include details of their work in the text.

Finally, I am also grateful for the encouragement during this project from The Robert Gordon University, School of Engineering staff, particularly Mr Kenneth S Gow and Mrs Ann B Reddipogu.

Sethuraman Muthuraman

ABSTRACT

This thesis describes a novel approach to the evolution of Modular Artificial Neural Networks. Standard Evolutionary Algorithms, used in this application include: Genetic Algorithms, Evolutionary Strategies, Evolutionary Programming and Genetic Programming; however, these often fail in the evolution of complex systems, particularly when such systems involve multi-domain sensory information which interacts in complex ways with system outputs. The aim in this work is to produce an evolutionary method that allows the structure of the network to evolve from simple to complex as it interacts with a dynamic environment. This new algorithm is therefore based on Incremental Evolution. A simulated model of a legged robot was used as a test-bed for the approach. The algorithm starts with a simple robotic body plan. This then grows incrementally in complexity along with its controlling neural network and the environment it reacts with. The network grows by adding modules to its structure – so the technique may also be termed a Growth Algorithm. Experiments are presented showing the successful evolution of multi-legged gaits and a simple vision system. These are then integrated together to form a complete robotic system. The possibility of the evolution of complex systems is one advantage of the algorithm and it is argued that it represents a possible path towards more advanced artificial intelligence. Applications in Electronics, Computer Science, Mechanical Engineering, and Aerospace are also discussed.

Contents

Chapter 1

Introduction

1.1 Introduction to Chapter

This chapter starts by describing the problems addressed by the project. The aims and objectives of the research are outlined and novel ideas discovered during the work are listed. A chapter by chapter breakdown of the thesis is also included.

1.2 The Nature of the Problem

The quest for Artificial Intelligence (AI) is one of the most exciting challenges that mankind has ever undertaken. The real promise of AI research is to study intelligent behaviour in humans and animals and attempt to engineer such behaviour in a computer or other machine. Biologically inspired Artificial Neural Networks (ANNs) are one of the tools used to achieve this.

At the present time, most of the research into ANNs which is not focused on Computational Neuroscience, is aimed at engineering applications. Examples of such applications include Pattern Recognition, Control Systems and Signal Processing. These usually involve fairly small networks with fixed topologies, unit functionality and training methods. This has led to the adoption of popular and simple "off the shelf" networks such as Back Propagation trained Multilayer Perceptrons, Radial Basis Networks and others.

This focus contrasts with the early expectations of connectionism, before the publication of "Perceptrons" [Minsky 1969]. Today, only a few researchers carry the flag for large general purpose networks as a route towards genuine intelligence in an unconstrained environment [de Garis 1995]. Most research towards this end has shifted away from neural nets and towards Robotic, Agent or Animat based routes such as Swarm Intelligence [Bonaneau 1999] and Interaction Based Systems [Warwick 1997].

The research presented in this thesis outlines a technique which draws on many of these strands of previous work.

The basis of this project is an evolutionary technique that allows an Artificial Neural Network to evolve in an unconstrained and open-ended manner. The method is demonstrated by using it to develop locomotive gaits for legged robots. The system works by starting with a mechanically simple robot, operating in a primitive environment. It then allows the environment and the robot's body plan, actuators and sensors to gradually become more sophisticated, while adding modules to the controlling neural network. In this way the controlling network grows in complexity along with the robot. As this development takes place, ANN modules (small networks) are added to the control system. During the process, previously evolved network structures are not retrained but retained. Since both the system and the network grow incrementally in complexity, this may be referred to as '*Incremental Evolution*'. The final intention of the research (beyond this thesis) is that, as the network develops, intelligence will eventually emerge.

A detailed explanation of the technique is given in Chapter 5. The method is based on computer modelling of an approach to biological evolution in an engineering context suggested by MacLeod et al in the PhD thesis of McMinn [McMinn 2002] - a previous researcher in the author's research group.

1.3 Modularity

The human brain has developed into a very complex structure through million of years of evolution. One of the great scientific challenges of this century will be to understand the code which lies behind its development. It is well known that the structure of the brain is modular [Arbib 1995]; that is, different parts specialize in different tasks (such as vision, taste, sound, touch, smell and language) and groups of neurons interact in complex ways. The modularity of the brain can also be illustrated by another example. When a person loses his vision as a result of brain damage, he is still able to smell, taste, or speak; if the brain were not modular, then all the processing capabilities would be affected when an area was damaged. Another advantage is that, in a modular system, individual functions are broken up into subprocesses that can be executed in separate modules without mutual interference

[Happ 1994]. One can even see this at a gross level in the human body, where different functions (for example, digestion and circulation) are carried out in different 'modules' (in this case the stomach and heart) in order to avoid interference between them.

1.4 Aim and Objectives:

The aim of this research was to develop an Evolutionary Algorithm (EA) to evolve ANNs in an open-ended way, without the need to artificially constrain them, so that they could automatically grow to an arbitrary level of complexity, without the need for human design or intervention. The EA should be able to automatically and naturally evolve a "system". A *system* in this context is defined as a group of fully interconnected ANN structures for multiple different, but related, functions; a good example of this is a robot where a "community" of ANNs may be associated with various sensory and motor functions. It is hoped that, by allowing ANN structures to evolve in this modular and incremental fashion, real "intelligence" would emerge.

To accomplish the aims, the following objectives were set out at the beginning of the project.

Background Reading and Appropriate Directed Study

Appropriate directed studies were undertaken at the beginning of the research. These included attending seminars and lectures in the field of study, understanding and reproducing work done by McMinn [McMinn 2002] and understanding the evolutionary method described in the paper "Evolution and Devolved Action" (EDA) [MacLeod 2002].

Literature Search in Field

A literature search into the development of ANN architectures was undertaken. The initial search concentrated on understanding the need for ANN architectures which can grow. Then, the concept of Modularity in ANNs was investigated. The search covered both the fixed and growing Modular ANNs (MANNs).

Later the concept of evolution of the Body-Brain system was studied. This type of evolution is applicable to robotic control systems. The growth of the robot's body plan and the ANNs controlling it was investigated. Finally, a search on Artificial Life was conducted to understand the effect of environment on the growth of ANNs.

Development of a Basic Central Pattern Generator (CPG) Network in a suitable format for Modular Evolution

The primary aim here was to investigate the development of a CPG which produces movement patterns for Legged Robots using the EA. This involved evolving both the body plan of the robot in terms of its actuators and sensors, and the environment it was interacting with. This was accomplished by allowing the robot's body plan and environment to start from a simple form and become more complex as it develops, while simultaneously adding ANNs to the structure of the controlling network.

Initial experiments were concerned with finding out whether it is possible to grow a modular neural network to control single functions, such as a simple leg. After evolving the control system for legs with a single degree of freedom, a second degree of mechanical freedom was added to the existing robot structure. In this case the previously evolved network structures are retained and new ANN structures were evolved as separate modules (but connected to existing modules by the EA) to control the new mechanical degree of freedom.

The EA under investigation was used to evolve CPGs for bipedal (walking and jumping) and quadrupedal (trotting) motions. The evolution of the ANNs, robot's body plan and environment (fitness function) was studied as the system evolved.

The Setting Up of an Experimental Framework for the Evolution of a Sensory System

The purpose of these experiments was to demonstrate the universality of the technique by applying it to a radically different type of network. The work outlined above was based on networks which mainly control outputs (producing walking patterns). On the other hand, a vision system processes inputs. Such a system allows investigations to be carried out to determine whether the technique can be applied more generally. To do this we allowed the sensor and the range of patterns to which it was exposed

started with a 1 by 1 grid (1 pixel) and evolved into a 5 by 5 (25 pixels) sensory system.

The application of the Previous Work to Such a Sensory System
The input sensor and the range of patterns to which it was exposed were allowed to grow from simple to complex as the environment changed and the ANNs controlling the behaviour were grown as described in the previous paragraph.

The Integration of these Techniques into an Overall Algorithm which Random capitalisation Deals with the Evolution of Systems
The issue of systems evolution, integrating both the locomotive and vision networks was considered. This included a consideration of the evolvability of networks in this domain and the neural functionality necessary to integrate these networks. Both the vision and locomotion networks were integrated by growing neural networks to map the different data sets into a single domain. Again, the ANNs have been grown using the method described previously.

Comparison with Previously Published Results from other Researchers
The results obtained in this research were compared with previously published results. Results were presented and discussed in detail to illustrate the technique in operation.

All the objectives mentioned in this section have been met.

1.5 Novel Aspects of this Research

Although researchers have used Evolutionary Algorithms (EAs) and Incremental Growth Algorithms (IGAs) for synthesising neural networks before, there are many unique aspects to the approach presented here. The most important of these are listed below.

- It was shown that, if the system is carefully set up (each module have a minimum number of neurons), the fitness can increase to a maximum as new ANN modules are added to previously evolved structures. This is an important result of the research.

- Experiments showed that the neuron model used was critical and should be as flexible as possible as it is required to perform many difficult mappings in both amplitude and time domains. This finding is core to the success of Incremental Growth of MANNs using EAs.
- Another significant finding was that the connections between modules as well as their weights have to be chosen by the EA. Fully connected networks are less successful in such Systems.
- Networks have been grown to integrate different networks to form a working system. This include the use of "Copy and Paste" methods, permissible connections for a particular module (especially in large networks; modules are added at the end or before of the previously evolved network) and finally network which produce several gaits and can switch between them.
- It was also shown that ANN modules can be added incrementally to the controlling network as the robot's body plan and the environment it interacts with evolves from simple to complex.
- Finally, in summary, the research has led to the discovery of a comprehensive method which allows the ANNs to grow incrementally to form a system.

1.6 Thesis Structure

Given below is an overview of each chapter.

Chapter 2: Review of Previous Work within the Department

This chapter describes the work undertaken by previous researchers within the research group and shows the development and context of the current work.

Chapter 3: Evolution by Devolved Action

In this chapter, the original proposal for the research is discussed and the five different practical approaches to the evolution of MANNs it contains are considered. A review of biological evolution and development which led to these approaches is presented.

Chapter 4: Literature Review

This chapter gives a review of other important work that relates to the research. In this chapter a separate section is devoted to describe the differences between the research work with other related investigations. It is hoped that this chapter will give a clear indication of the originality of this research.

Chapter 5: Growth Components for Evolution of Modular Artificial Neural Networks

The different types of simulated neurons and actuator models used in the research are discussed in this chapter. Both the robot's body plan and vision system framework are also presented. Finally, the growth algorithm is illustrated.

Chapter 6: Results Obtained from Application of Growth Strategies for a Single Function

The results obtained for fully and sparsely connected network modules to control single functions using two different types of neuron models for bipedal and quadrupedal locomotion are presented in this chapter. The result of localising the neural module's connections are also presented.

Chapter 7: Results Obtained from the Application of Growth Strategies to Multiple Related Functions

In this chapter, the results of network modules used to control further degrees of freedom for bipedal walking and quadruped trotting are presented. Results also illustrate the universality of the growth strategies for "copy and paste" and multiple gait networks.

Chapter 8: Results Obtained from the Application of Growth Strategies to Vision System and Integration of Vision and Locomotive Networks

The responses obtained from the sensory system are given in this section. The outcomes of systems integration the locomotive and vision networks are also demonstrated.

Chapter 9: Further Work

In this chapter suggestions are made for further work. Different application areas for the technique are described. Improvements that can be made with the growth technique are described. Methods to apply the growth technique to achieve the eventual goal of the research, beyond this thesis (emergence of complex and intelligent behaviours) are presented.

Chapter 10: Conclusions

The final chapter revisits the objectives outlined in the first chapter and critically assesses the success of the project.

Published papers and reports produced during the course of the research, and further results are included in appendices.

Chapter 2

Review of Previous work within the Research Group

2.1 Introduction to Chapter

The Artificial Neural Networks group in the School of Engineering at The Robert Gordon University was formed in 1994. Since then it has built up a considerable amount of knowledge and practical experience with Evolutionary Artificial Neural Networks. This work started with the PhD project of MacLeod [MacLeod 1999] and was continued by McMinn [McMinn 2002], Reddipogu [Reddipogu 2002] and others. The current research has evolved from work undertaken by researchers within this group. In this chapter, the previous research of the group and its development into the project work presented here is discussed.

2.2 Single String Evolutionary Techniques

During the early stages of research into Evolutionary Artificial Neural Networks (EANNs), the architecture of each network was predefined and fixed for a given task (the architecture of an EANN includes its topological structure and the connectivity of each node in the network). This has a significant impact on the network's information processing abilities. Unfortunately, the architectural design was heavily dependent on a human expert and involved much trial and error.

The group's first project [MacLeod 1999], concentrated on the optimisation of ANN topologies using Incremental Evolution (IE) - that is, allowing the network to expand by adding to its structure. This method allows the network to grow from a simple to a complex form, until it is capable of fulfilling its intended function. The approach is sometimes thought of as being somewhat analogous to the growth of an embryo and is therefore also called Incremental Growth or occasionally Embryology or an Embryological Algorithm (EA).

To illustrate the technique, let us first consider a fully connected, three layer standard network, as shown in Figure 1.

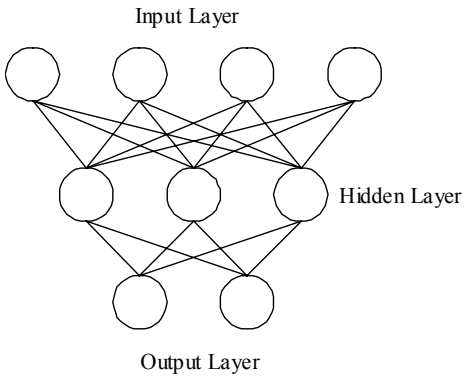


Figure 1 A fully connected network

This network will be used as a reference when describing the growth strategies. There are six different growth strategies which can be considered. These are:

1. Change the number of neurons
    - The number of neurons in a layer may be increased or decreased while maintaining a fully connected network.
2. Change the connectivity
    - The number of connections (active weights) in the network may be reduced or increased.
3. Asymmetry
    - Asymmetry may be introduced by providing more connectivity in part of the network
4. Horizontal connection
    - In synchronous networks (those which operate with a clock signal) horizontal connections may be introduced between neurons in the same layer.
5. Skipping layers
    - Rather than connecting to the layer directly below, a connection may skip a layer.

6. Feedback
    - Feedback may be added to the network. A connection is allowed to any previous layers.

To illustrate the operation of incremental growth, MacLeod applied the growth strategies to a simple two layer network designed for a character recognition problem. A basic example of the technique's operation is a network which adds neurons to its hidden layer, one by one, until the network is capable of fulfilling its intended functionality. The idea of the growth strategy is that the network changes in a predictable way and grows by adding incrementally to its structure [MacLeod 1999]. Figure 2 shows how the network's performance changes as neurons are added to its hidden layer.

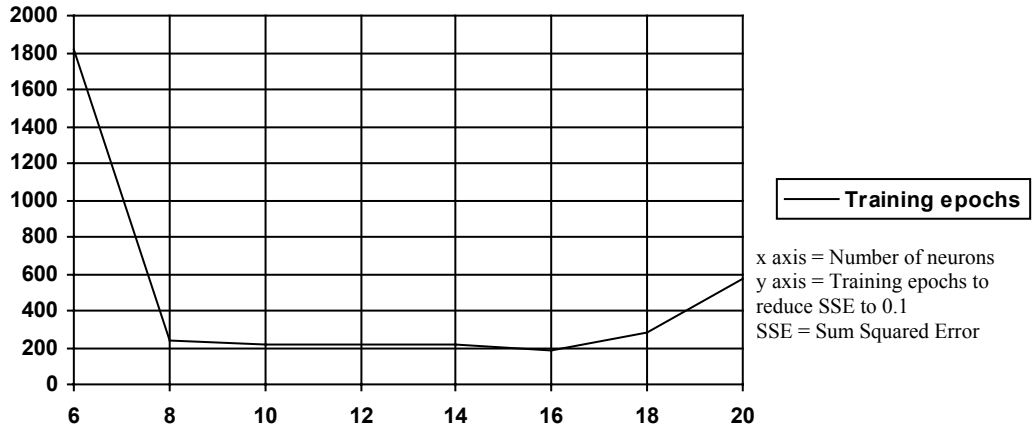


Figure 2 Network performance changes as hidden layer neurons are added to a pattern recognition network (Reproduced by permission of MacLeod)

The performance measure used was the number of training cycles required to train to a Sum Squared Error (SSE) of 0.1. Notice from Figure 2, that the network cannot solve the problem with fewer than six neurons but the performance increases as the number of neurons increases. 16 neurons is the optimal number for fastest training and by 20 neurons the network starts over-fitting.

MacLeod successfully used these growth strategies together with an encoding scheme, in a unified algorithmic framework to illustrate network growth for simple pattern recognition problems.

We may summarize MacLeod's work by noting that, although the network expands as the algorithm runs, the system is limited in that:

1) It is applied only to simple tasks.
2) It uses only the basic McCulloch-Pitts neuron model.
3) The whole network must be retrained after each alteration to its topology.
4) The network architectures used are essentially structured (layered) and simple.

At the end of this initial stage of research, a model of an Artificial Nervous System [MacLeod 1999] (ANS) was proposed by MacLeod as a suitable test-bed for further research into more complex network problems and, in particular, those involved in defining complex ANNs in a system context. It was suggested that this model could be used to construct a control system for an animal-like robot (an animat).

2.3 Evolution of Functions within the Animat Nervous System (ANS) – Lower Layers

The ANS model suggested by MacLeod is both hierarchical and modular; it consists of smaller individual networks operating together. The model allows us to understand the working principles of the nervous system's component modules, their interaction, connectivity and organisation. McMinn and Reddipogu implemented some aspects of the nervous system and insights into their work are described in the following sections. The ANS model enabled them to create a community of networks for a particular task. The networks were evolved based on a simulated robot.

It is necessary to first consider the ANS model as this forms the basis for the structure of later work and for a comparison of the results, as well as being an inspiration for the current research. The ANS is shown in Figure 3. Multiple modules can exist in certain layers marked with an asterisk.

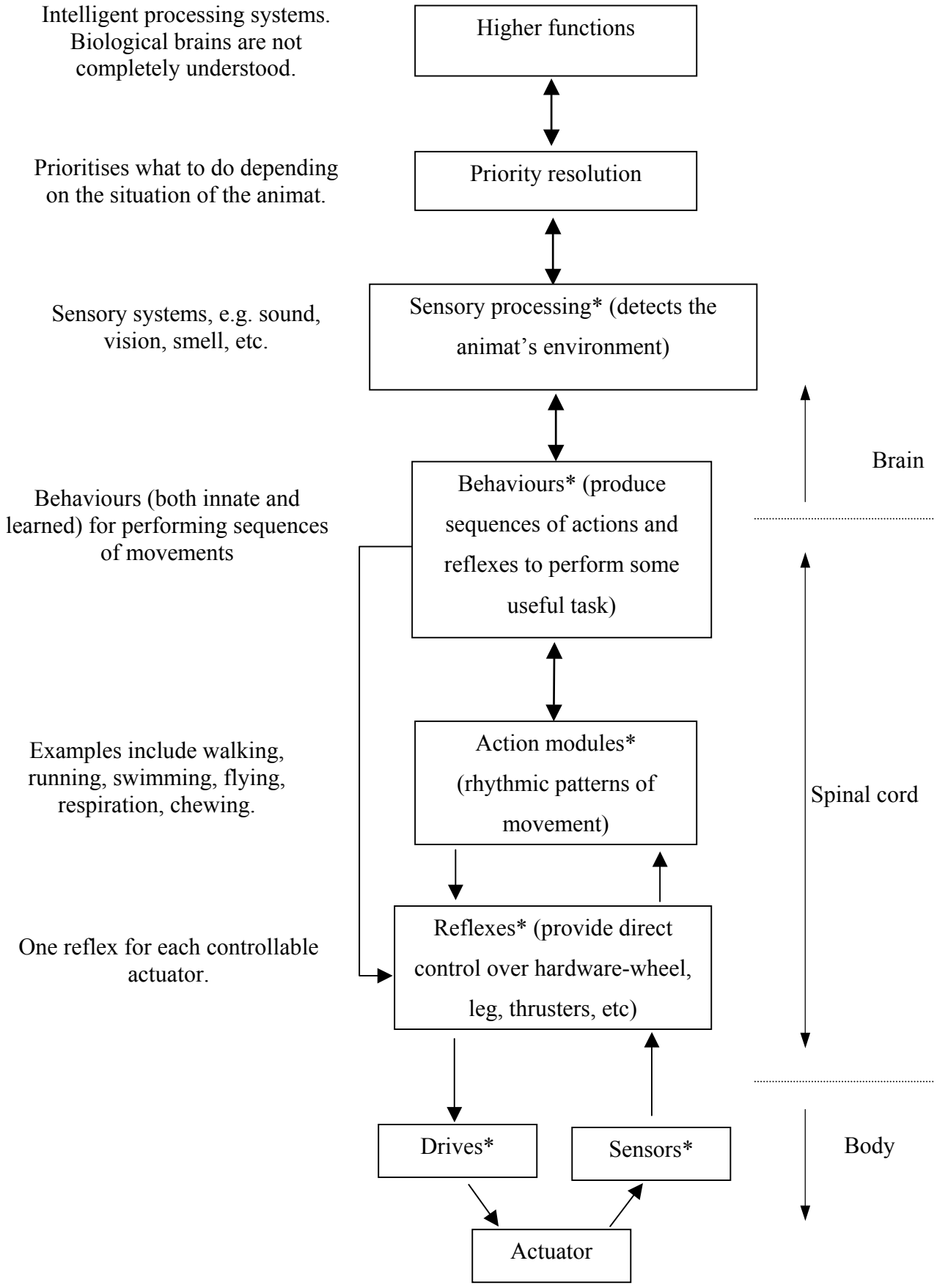


Figure 3 Animat Nervous System (ANS) (Reproduced by permission of McMinn)

The highest layer, labelled “higher functions”, in Figure 3, represents the intelligence layer, where higher levels of brain activity (like reasoned thought) reside. This is connected to the priority layer; here behaviours or actions are given a priority depending on the condition of the system. The sensory processing layer gathers information from the system environment using, for example, vision, sound and/or other sensors. This then triggers the appropriate behavioural modules for the current state. In turn, these initiate a sequence of actions from the action layer. The action layer uses the reflex layer to produce repetitive or rhythmic actions such as running or walking and corresponds to the Central Pattern Generator (CPG) in animals. Reflexes are used to control the physical movements of the system. Feedback from the actuators and sensors is fed to the reflex layer in order to make any movements precise and efficient in the form of a feedback control system.

The original ANS [MacLeod 1999] represented the flow of information in one direction, from the upper layer to the bottom layer. In later versions of the ANS structure [McMinn 2002], there were interactions among modules starting from the action module moving upwards on the ANS, as shown. If the system senses a change in its environment, it uses the higher functions to evaluate and prioritise the conditions before initiating any behaviour to produce a sequence of actions.

McMinn used this structure successfully as a basis to develop Evolutionary ANNs implementing Central Pattern Generators (Action Layer) and Reflexes (Reflex Layer) for robot locomotion [McMinn 2002]. Figure 4 shows the block diagram of the functionality of McMinn’s artificial reflex. The reflex ANN circuitry controls the position of the actuator. The actuator sensor in turn provides an additional input to the reflex on the status of the actuator. The artificial reflexes were created using a simulation of a DC electric motor as the system actuators.

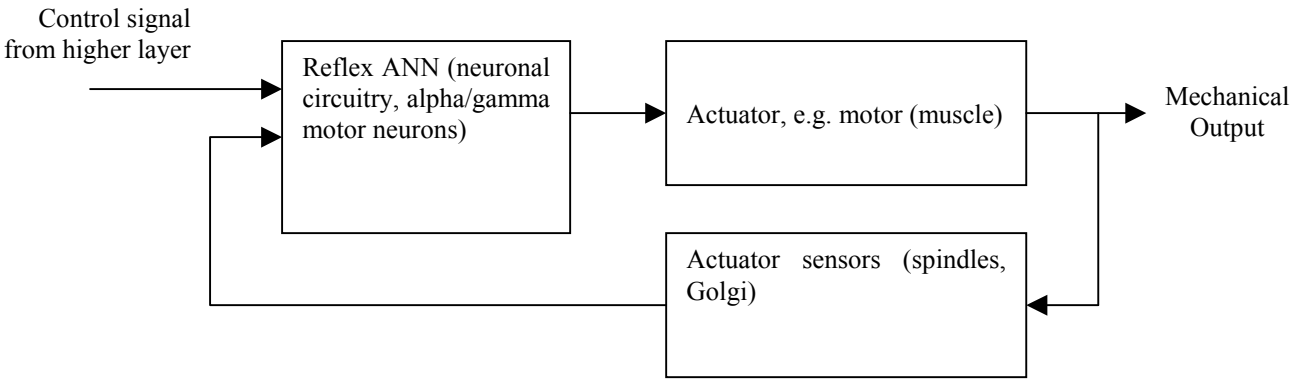


Figure 4 Functional block diagram of artificial stretch reflex, with biological equivalent parts marked (Reproduced by permission of McMinn)

Simple feed-forward and recurrent networks were used. The type of neuron was limited to a McCulloch-Pitts model with a sigmoid transfer function. The three main EAs (GA, EP, and ES) were used to train the reflex ANNs and their performance was compared. The ANN weights were trained until a good solution was found.

After creating the lowest layer of the ANS (the reflex), McMinn constructed the action layer. This layer was built on the functions provided by the modules in the reflex layer. The neural circuits responsible for generating rhythmic patterns (for locomotion) in the biological nervous system are called Central Pattern Generators (CPGs). McMinn successfully evolved CPGs for biped and quadruped gaits.

A new neuron model was developed specifically to simulate the timings required for the CPGs. The simple McCulloh-Pitts neuron does not produce time varying outputs and therefore the synapse model used in the artificial CPG networks was designed to include features which made it more suitable for simple implementation of time dependant parameters. More information about the neuron and synapse model can be found in [McMinn 2002].

The neurons in the network were randomly connected; there was no imposed layered structure in the network. The artificial CPG networks were created using an Evolutionary Strategy (ES). Again, the entire network's connections were retrained until a good solution was found.

Finally, McMinn combined the evolved CPGs with the reflexes as shown in Figure 5. Since the CPG neurons produce pulsed outputs in the time domain and the reflexes require a continuous input value, a "leaky integrator" was added to convert from discrete pulses to an average firing frequency. For further information on leaky integrators refer to [McMinn 2002].

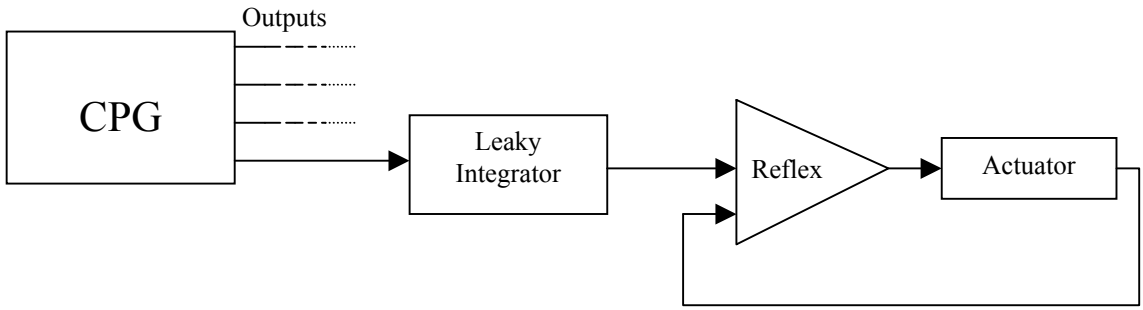


Figure 5 Chain of connections from CPG to robot actuator (Reproduced by permission of McMinn)

An alternate strategy for structuring the network was also investigated. The CPG evolved for the biped walking pattern was used as an oscillator. The pattern generator took the oscillating inputs from this and produced the appropriate gait patterns as outputs. The connection between the two units is shown in Figure 6.

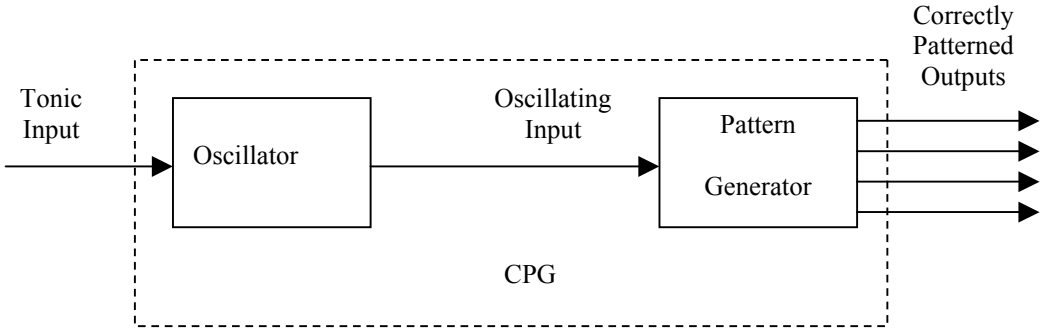


Figure 6 Connectivity of the functional units in alternate CPG strategy (Reproduced by permission of McMinn)

Quadruped Gallop, Trot, Pronk, and walking gaits were successfully evolved using this alternative method. An example result for a quadruped gallop is shown in Figure 7. The conclusion of these experiments was that by making the structure of the CPGs as modular as possible, they can be evolved more easily.

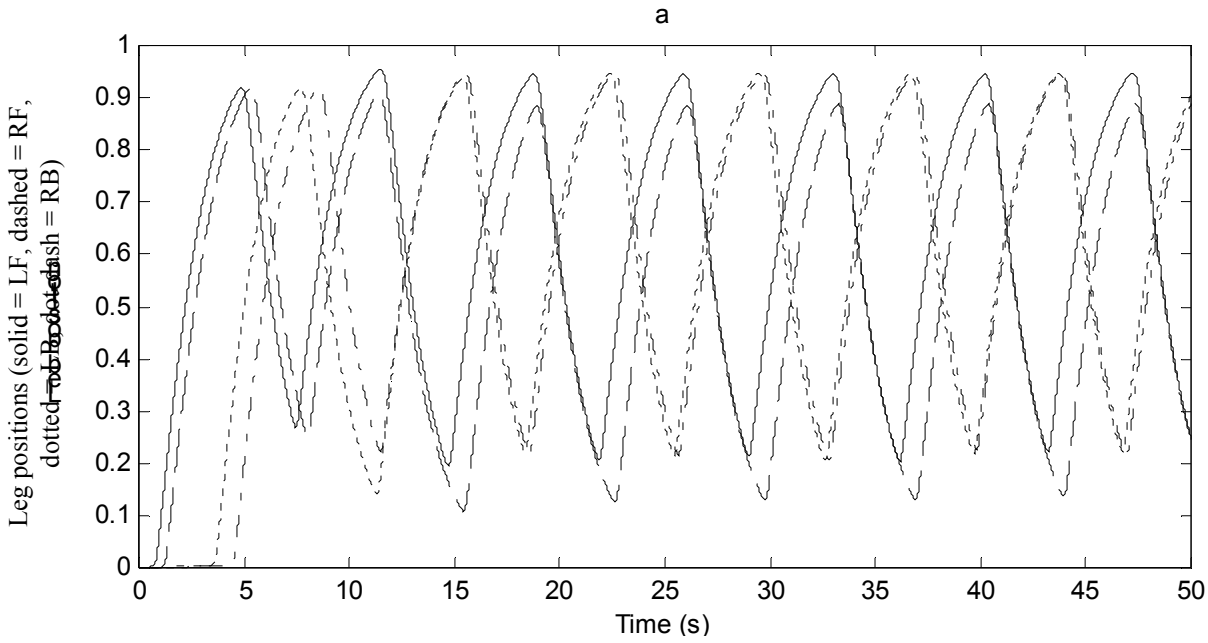


Figure 7 Robot leg positions for Quadruped gallop with split CPG (Reproduced by permission of McMinn)

2.4 Evolution of Functions within the Animat Nervous System (ANS) – Upper Layers

Reddipogu looked at the upper layers of the ANS. The work mainly concentrated on the sensory layer and particularly the processing of visual information. A careful search of the various options was undertaken to find a suitable neural network which combined simplicity and functionality. Eventually, it was found that the visual system of toads was interesting since their brains are structurally simpler then the human brain, and this offered a good model to build a novel visual system upon.

A biologically inspired vision system, based on the toad's ability to differentiate between prey and predator, was then developed. This work is described below.

Firstly, the visual field was spilt into a grid (for example, 10 x 10), which forms the front view of the toad, as shown in Figure 8. The various patterns that best represent the prey and predator configuration are presented within the visual field at various locations. For example, if a worm configuration (a long horizontal line) is presented in the snapping region, the expected behavior would to be for the toad to snap.

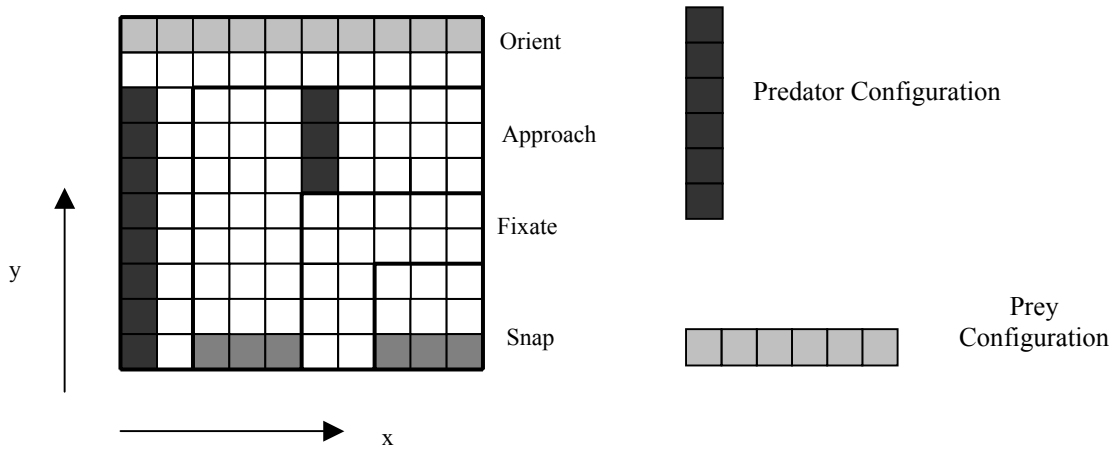


Figure 8 Toad's View (Reproduced by permission of Reddipogu)

A modified biological neural circuit based on a toad's vision system, proposed by Ewert [Ewert 1987], was used for testing the system's suitability for simple pattern recognition tasks, as shown in Figure 9 (the network has been reduced in size for simplicity). All the neurons in the network are McCulloch-Pitts type with a sigmoid logistic. An Evolutionary Algorithm, using Reinforcement Learning (EARL) was used to train the network. The network connection weights are trained until a good solution is found, incorporating all different input patterns.

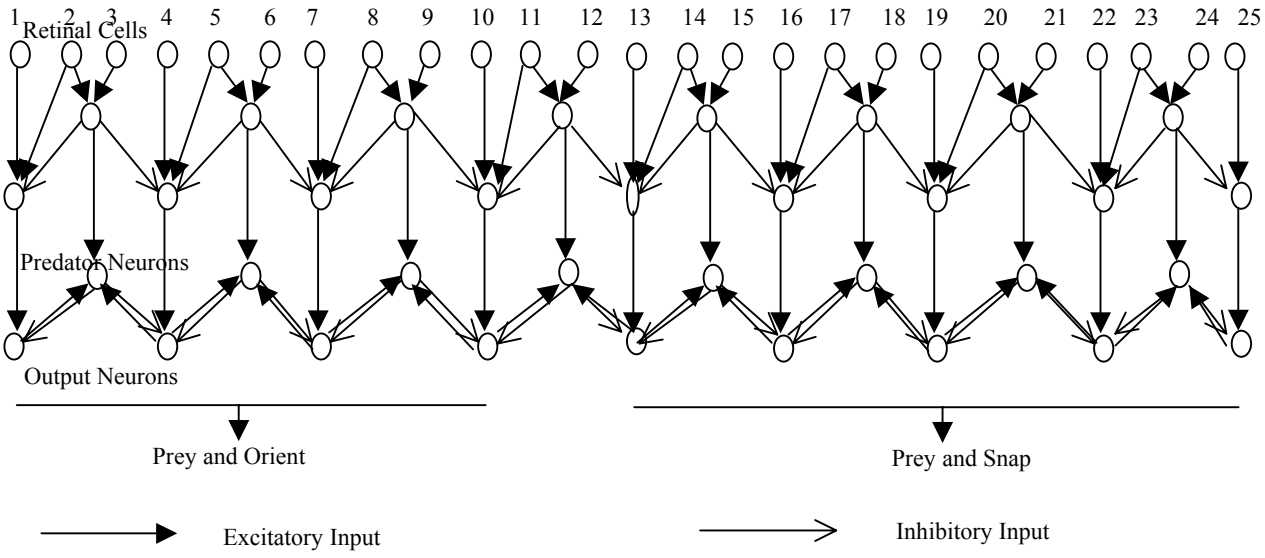


Figure 9 The network of the vision system based on the toad (Reproduced by permission of Reddipogu)

The network was then tested with new patterns to check its ability to generalize. A typical output of the network is shown in Figure 10. The horizontal axis represents the classes of outputs and the vertical axis corresponds to activation level of each predator and prey output neurons.

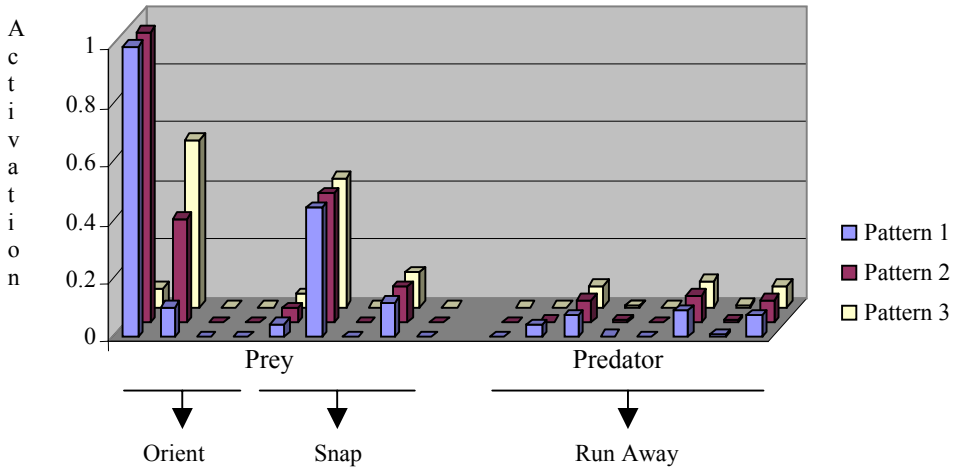


Figure 10 Output for Prey and Orient input pattern (Reproduced by permission of Reddipogu)

The artificial vision system was trained using inputs that best represented prey and predator patterns in various positions in space. Later, the network successfully recognised the combination of patterns which were not part of the training set and developed into a Robotic Vision System. The capabilities of the network are thought to arise from its modularity. Further detailed analysis of this network can be found in [Reddipogu 2002].

McMinn and Reddipogu's work was aimed at investigating the effect of modularity on the network and its evolution. However, it should be noted that the arrangements of the modules within the system is fixed and that the structured growth aspect introduced by MacLeod had been lost.

2.5 Conclusions Drawn from the Group's Previous Work

Although interesting conclusions were drawn from the work described in the previous sections, it became apparent, over the course of these projects, that a network which can evolve into a modular structure without the need for designed partitioning would be the next stage in the research. This would represent the most general Evolutionary Networks. The EA should allow the network to develop naturally and in an open-ended way without the need to artificially constrain or design it. Such an approach needed an EA that could automatically and naturally evolve a "system"- that is, a

modular network which could operate in different sensory domains rather than a fully interconnected homogenous structure. No existing Genetic Algorithms or EAs were available to do this. Therefore the group looked to nature to discover the reasons why natural systems allowed such modularity to evolve and how it might be exploited. This search for a more general and sophisticated algorithm resulted in the paper "Evolution and devolved action" which is discussed in Chapter 3. The paper concluded that the growth aspect of evolution in MacLeod's work needed to be integrated with the modular networks of McMinn and Reddipogu to produce a more general system.

2.6 Summary

Initial research within the RGU group focused on the growth of simple networks to fulfil relatively straightforward functions, using simple neurons. From this an interest in "Communities" of networks working together as a system developed. Research in this area was undertaken using an 'Artificial Nervous System' as an experimental framework with particular reference to robotics.

It became apparent, during this research, that the most general system would be a combination of the two techniques above (growth and modularity), resulting in a system which could evolve or grow modular neural networks. However, suitable theoretical frameworks and algorithms for this purpose were lacking and this forced the group to look back to biology for inspiration. This resulted in the paper "Evolution and devolved action" which is the foundation stone upon which this current research is built. The next chapter gives a review of the paper, its conclusions and developments into the current work.

Chapter 3

Evolution and Devolved Action

3.1 Introduction to the Chapter

As explained at the end of the previous chapter, the paper "Evolution and Devolved Action" formed the starting point of the research reported here; the paper is attached in Appendix A.

"Evolution and Devolved Action" examines the limitations of present Artificial Evolutionary Algorithms from a biological perspective and looks at how these limitations might be overcome. A central theme of the paper is a view of genetics as a system of Evolutionary Automata. The paper is wide ranging and contains several other important topics, including Evolutionary Cellular Automata and Learning and Functionality in Neural Networks. This thesis, however, only deals with the evolution of network topology (other researchers within the group are examining other issues).

This chapter describes how the reconsideration of evolutionary algorithms, mentioned above, led to five different suggested approaches to the evolution of network topology and how these were, in turn, amalgamated into one "universal" approach. The chapter is designed to provide a brief summary and commentary on the important points of the paper and for more details the reader is referred to the original in the appendix.

The previous work of the group, explained in Chapter 2, may be summarised as:

Initial work by MacLeod concentrated on growing simple ANN topologies using Incremental Growth. Later, McMinn and Reddipogu investigated the effect of modularity on the network and its evolution, using the ANS model. The conclusions of these research projects were:

1) Simple Evolutionary Algorithms were not flexible enough to allow the sophisticated development seen in biology.

2) An Evolutionary Algorithm was required to allow the network to combine the two previous approaches – that is, allow the network to grow, but also incorporate a modular aspect (which McMinn had shown was important) into its development. Such a modular approach should allow different sets of sensors and actuators to be integrated into the system - that is, it should allow a complete system to develop naturally.

None of the available algorithms allowed the network topology to evolve in this way. In the next sections, the approach of the paper to these problems will be examined, starting, as the paper does, with a review of biology.

3.2 Biological Evolution

Chemical analysis shows that the genetic information or blueprint of an organism is encoded by deoxyribonucleic acid (DNA). DNA is a very long molecule which encodes this information as a unique sequence of four chemicals called 'bases'. The bases are: Adenine (A), cytosine (C), guanine (G), and thymine (T). In humans the DNA is a linear arrangement of 3.1×10^9 bases.

The information stored in DNA is read and used by other molecules. Each short portion of the DNA string directs the synthesis of specific amino acid molecules. Chains of amino acids are joined together by peptide bonds to form a protein. There are twenty amino acids found in proteins and the number of different ways that they can be combined is very large. The process is summarised in Figure 3-1.

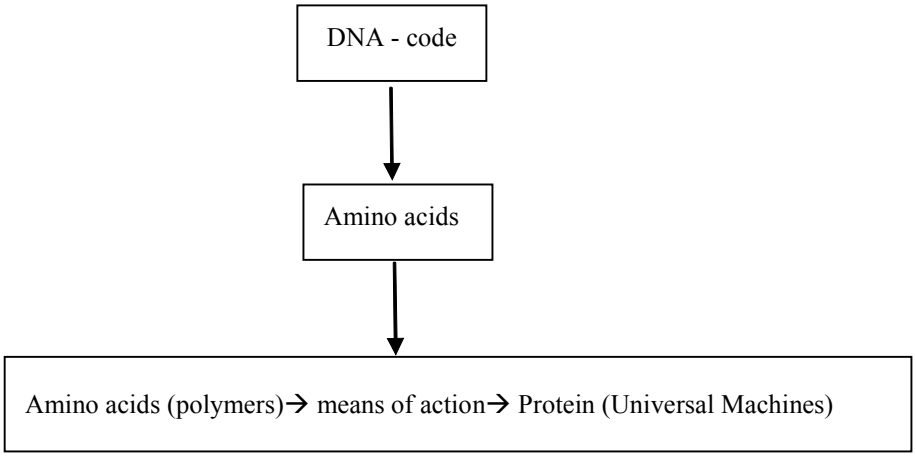


Figure 3-1 How DNA codes proteins

Proteins are the universal machines of biology. They play a predominant role in most biological processes. Proteins determine the shape and structure of cells and provide their functionality.

Biological engines like the brain or liver are manufactured by the assembly of large amounts of proteins. These protein machines can react chemically, form rigid structures, react mechanically or perform a multitude of other tasks. Critically, they can also self-organise like pieces of a jigsaw puzzle into a greater and more complex system.

Proteins can therefore perform an impressive array of tasks. In fact, it could be said that they are the 'Universal Machines' of the cell. Figure 3-2 shows a tentative classification.

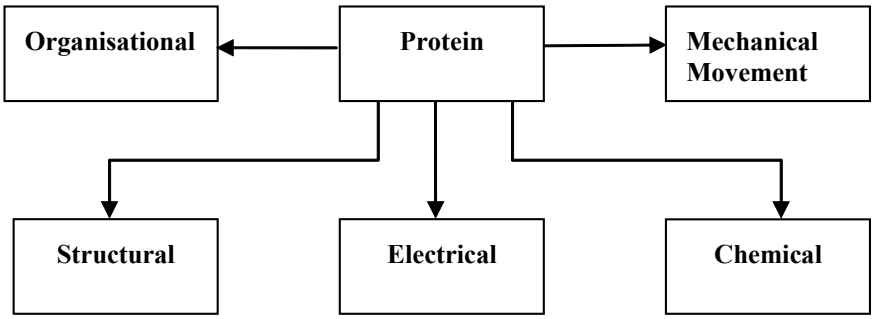


Figure 3-2 Proteins as Universal Machines

Proteins can also lock to each other or to the parent DNA and stop it producing more of the protein (or a different protein), so parts of the code can be switched on or off.

During to the foetal development of an organism, released proteins set up "gradients", which in turn inhibit or excite other proteins building up patterns of material. In this way smaller and smaller details can be built as one protein triggers another. One result of this activity is that the physical structure produced is not homogenous but modular, with delineated identifiable regions that perform specific tasks. This is important because structures like the brain have been shown to be modular and this modularity is essential to functionality.

The rules governing proteins and their structure are determined during evolution. So, over time, natural selection and mutation produces particular proteins which interact with others in a beneficial way.

We may summarize all this by saying that the biological system has two components, as shown in Figure 3-3.

- Firstly, a code (the DNA) which can be mutated and exchanged through breeding.
- Secondly, the universal machines (proteins) which the code specifies and which can assemble into complex structures and build biological engines.

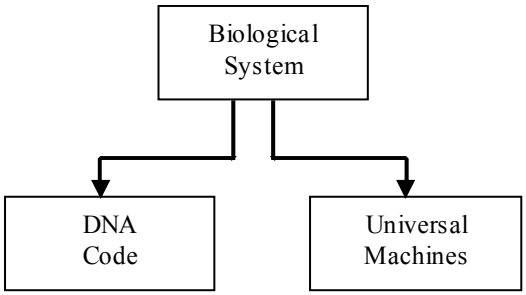


Figure 3-3 Biological components

The biological system is therefore not directly coded into the genome as in most of the current artificial EAs.

ANNs are usually directly coded into a Genetic Algorithm (GA) [Schaffer 1992], if such is to be used for topology evolution. Each node or connection will be a parameter of the chromosome. However, the entire human genome does not contain enough space to directly code even a small part of an actual biological brain.

The conclusion of the review from biology contained in 'devolved evolution' is that the biological system is encoded quite differently to the artificial techniques. An in-depth discussion on this aspect of biological evolution can be found in [McMinn 2002].

3.2.1 Organism at the Cellular Level

Having established some of the reasons why biology is different to artificial evolution, the paper concludes that the implementation of an artificial system mimicking biology at the molecular level would be very difficult (because of the difficulty of mimicking the wide ranging behaviour of proteins). It then goes on to discuss how the lessons learnt from molecular behaviour might be applied to structural evolution in neural networks by considering the process of structure formation at the cellular level.

At the cellular level there are four main processes in the development of an organism. These are cell differentiation, proliferation, migration and patterning. Consider these aspects.

A single fertilised cell produces many cells by means of cell division. Specialised cell types are created in a process known as differentiation. As the cells receive different protein combinations and concentrations from other cells in the environment according to their location, different genes are expressed within them. When they divide, their offspring are different from the parent cell, and cells become specialised for different tasks, for example, bone, muscle or neurons.

Differentiated cells have to generate many new offspring that will form the bulk of the brain and similar structures. This process is known as proliferation. The specialised cells divide until there are enough of them to build the structure of the organism.

For various reasons after differentiation and proliferation cells might not be at their final destination. Clusters of cells will then migrate to their ultimate home. Finally, in the case of neurons, connections are established within the clusters (locally) and between clusters of cells (globally). More information on biological pattern formation can be found in [Bentley 2001].

3.3 Organisation Methods

Although it is difficult to mimic and model the biological system exactly, an engineering standpoint can be taken to extract the essence of what is required to produce a working network then code this from a purely pragmatic point of view. However, there are certain obvious aspects that the algorithm will have to accommodate.

The four elements of network organisation as outlined above are position, quantity, function and connection of units. These are the key aspects of the network.

Although positional (migration) organisation plays an important role in the development of human and higher primates, as will be shown, it plays a lesser role compared to the other elements.

Outlined below are five different methods for creating networks as described in Evolution and Devolved Action [McMinn 2002],

1) Modelling Biology
2) Production Trees
3) Fractals
4) Revising Traditional Evolutionary Algorithms
5) Direct Growth.

Consider these.

3.3.1 Modelling Biology

The first method is to simulate biological development closely using a computer model, as shown below.

Firstly, an evolution space is defined as shown in Figure 3-4 a).

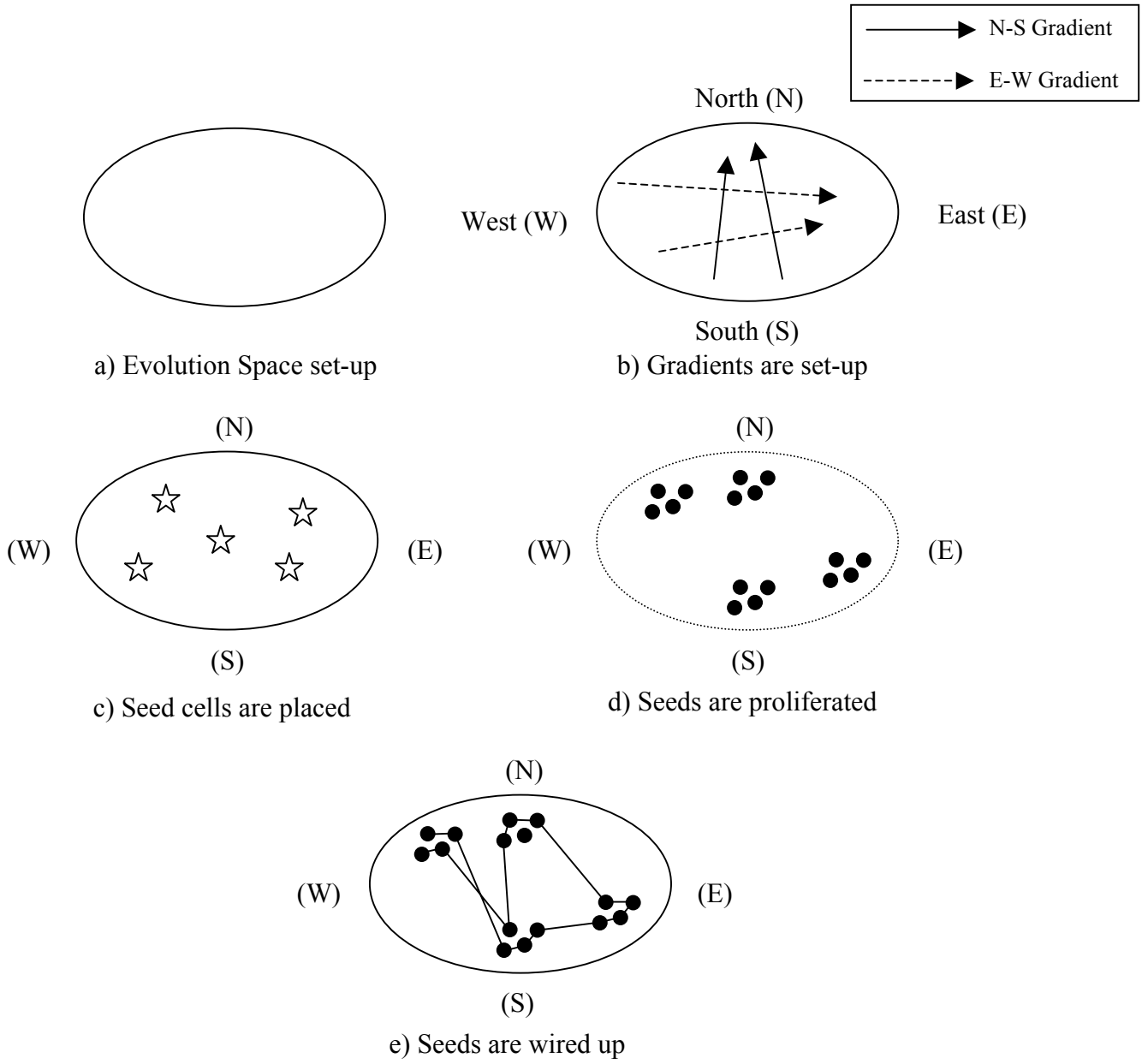


Figure 3-4 Modelling biology

The evolution space acts rather like the body of an organism and allows the set up the conditions necessary for development. The evolution space has a North-South and East-West gradient as shown in Figure 3-4 b) (in biology these gradients are set up by chemical diffusion of proteins within the organism). A number of seed cells are released into the evolution space, Figure 3-4 c). These are pre-programmed (by the EA) to migrate to fixed positions within the space defined by the gradient. Once the seed cells are in position they proliferate.

Again, this is controlled by an EA determined parameter pre-programmed into each seed. The result of this is that modules or clusters of cells now exist centred at the seed-cell positions as shown in Figure 3-4 d). Finally, these clusters are wired up, Figure 3-4 e). To follow the biological example through, this can be done using cellular adhesion markers (again chosen by the EA) which control which cells should be attached to which others.

3.3.2 Production Trees

Another approach that captures the essence of the biological approach but at simpler level is to use production trees to evolve ANNs. A typical tree for encoding a network is shown below in Figure 3-5.

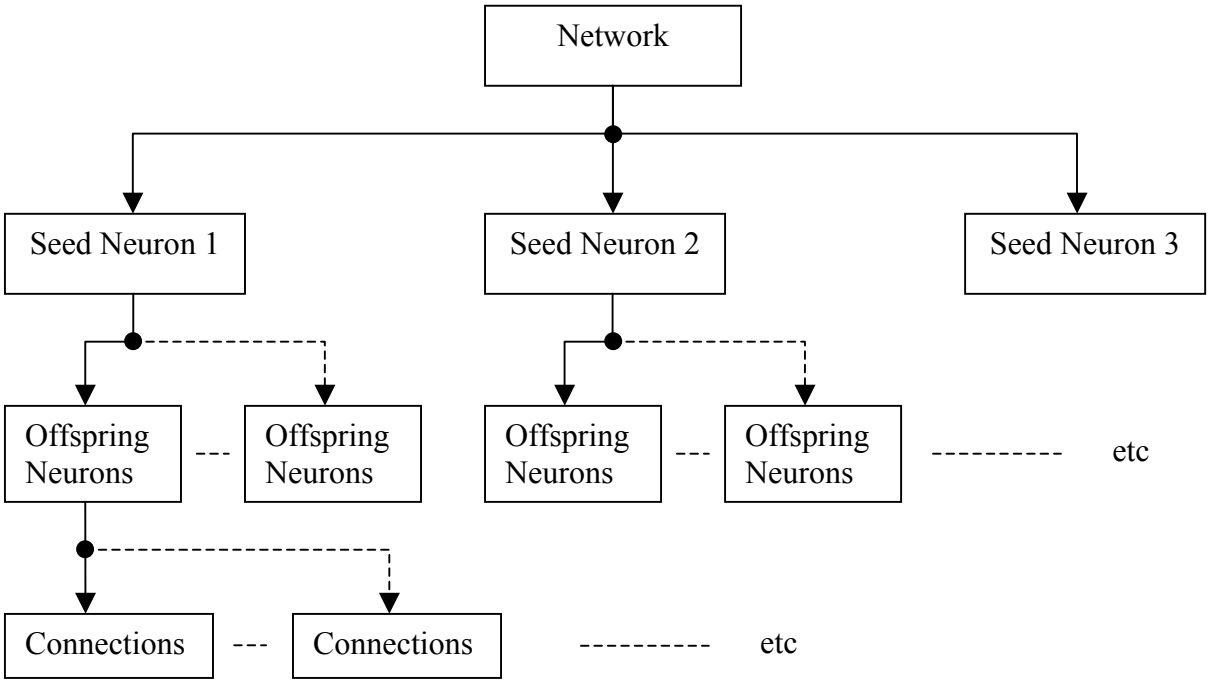


Figure 3-5 A production trees encoding method (Reproduced by permission of McMinn)

The rules for the encoding method are as follows:

I. Start with a network
II. Create multiple seeds
III. Create offspring for each of the seed cells
IV. Connect the offspring

The tree structure comes under the control of the EA. A system like this has the advantage of swapping or mutating the individual braches of the tree when crossover and mutation operators in GP are applied. In this way, important sections of the network can be re-used without the need to be re-evolved. The connections may be part of the production rules or evolved using genetic coding. The genetic coding should include the number of seeds, number of offspring and connection information. One can readily see that this produces a similar result to the previous biological method, but is simpler, more stylised and more suitable for a computational implementation because of its structure.

3.3.3 Fractals

The complex repeating patterns produced by plants, for example ferns, are known as 'fractals' and provide a means to evolve ANN topology. Biological systems in higher animals also display such symmetry (as, for example, does the biological nervous system). The idea that fractals could be used in defining ANN topologies has been suggested before [MacLeod 1999], but researchers have yet to take it seriously enough to produce a working system and therefore very little work has been done in this area. There are two obvious ways to use the fractals as described in [McMinn 2002]. Firstly, the nodes of the fractal could be used as placement points for neurons and the branches for their connections. This is illustrated in Figure 3-6 a) below. Alternately, the nodes could be placement points for network modules, Figure 3-6 b).

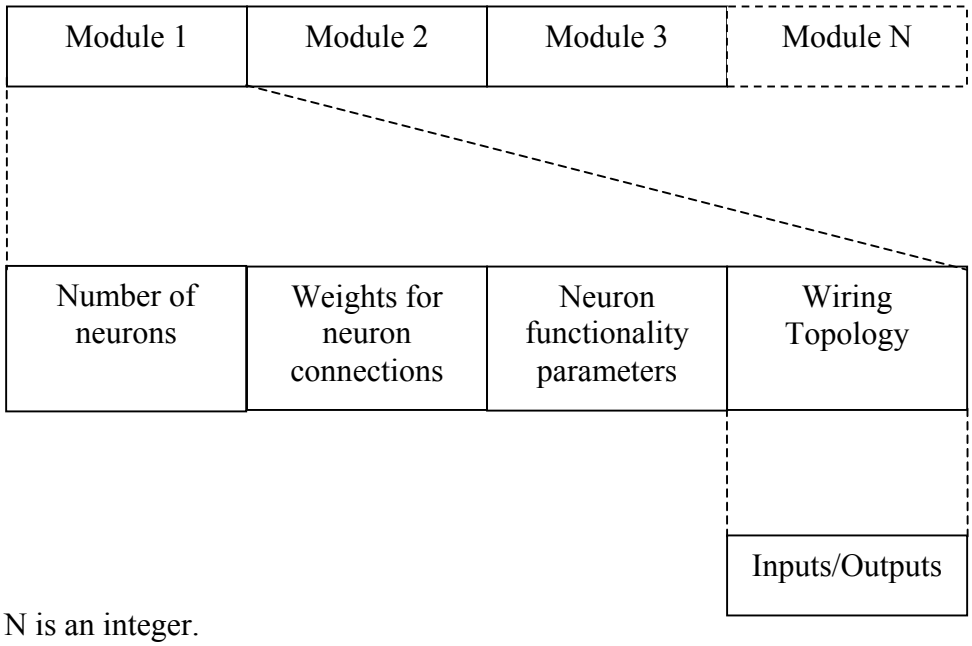
a) Black circles represent neurons b) Black circles represent modules

Figure 3-6 Fractal Method (Reproduced by permission of McMinn)

3.3.4 Altering Existing Evolutionary Algorithms

Another approach is to modify the standard EAs (GA, GP, EP, and ES) to produce a modular result in an ANN. There are several possible ways to do this – for example:

1) Define each module by a section of chromosome within the population of the GA, as shown in Figure 3-7. Each section of the module is divided to code the number of neurons in the module, the respective weight for every neuron connection in the module, the neuron functionality parameters and the information on the wiring topology. The wiring topology section could be further sub-divided to represent the information on which neurons act as inputs and outputs. These allow connections to be established to other modules. As modules are added, the string is allowed to grow.



N is an integer.

Figure 3-7 An internal representation of a chromosome

2) An extension to the method above is to have a fixed string length for each module. This is an attempt to get around the problem of strings having to grow if modules become bigger or, alternatively, have an independent GA for each module. A new module could be created when the GA string reaches a certain fixed size or when the network had fulfilled its function (once the fitness of the network is as high as possible). At this point the algorithm automatically creates a sub-network which is independent of the parent network.

3.3.5 Direct Growth

The final technique presented in the paper is termed Direct Modular Growth. The method works as follows: consider the concept of an “evolution space” where the network will develop as shown in Figure 3-8.

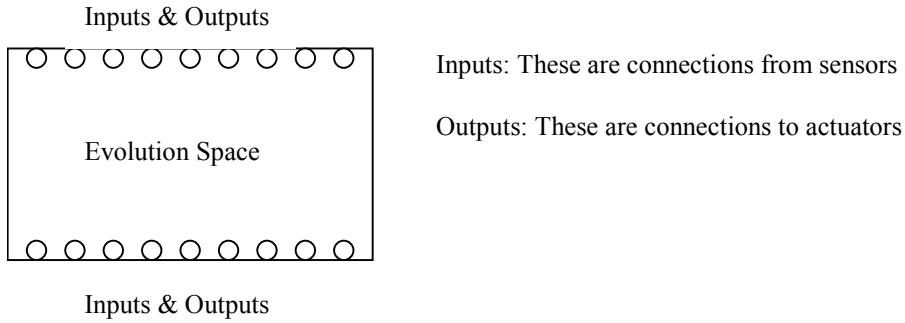


Figure 3-8 An "Evolution Space" (Reproduced by permission of McMinn)

In the traditional approach, a fixed network of neurons is placed in the evolution space and its connection weights are evolved as shown in Figure 3-9.

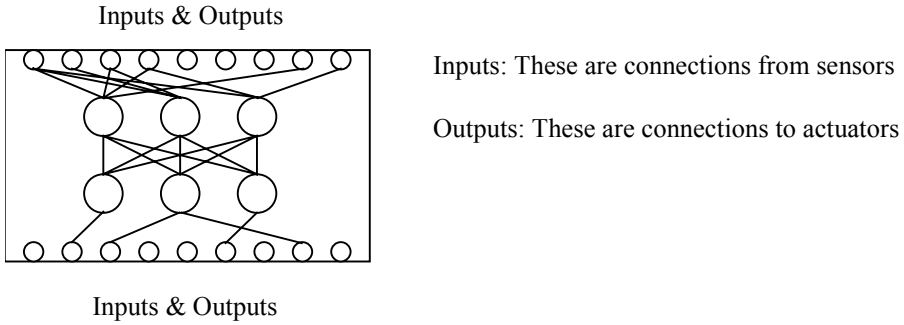
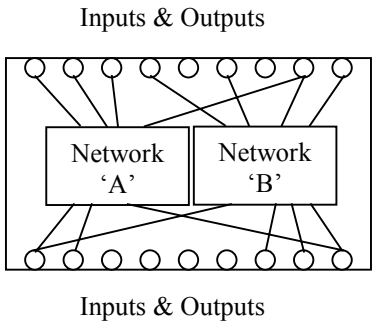


Figure 3-9 Evolution space for traditional ANN (Reproduced by permission of McMinn)

However, this concept can be easily adopted to serve modular neural networks. This is achieved by replacing individual neurons in the diagram above by networks, as in Figure 3-10.

An evolutionary algorithm determines the wiring between the networks and inputs/outputs. This evolutionary algorithm also decides which connections should be present within each network.



Inputs: These are connections from sensors

Outputs: These are connections to actuators

Figure 3-10 Evolution space for modular networks

3.3.6 The Role of Incremental Change

The approach outlined above has several unresolved issues. These includes how many modules should there be in the system, and how should they be placed with regard to the system sensors and actuators.

These considerations resulted in Incremental Evolution becoming a central part of the system. After all, if an animal had to go through a series of gradual changes from simple to complex as part of its evolution, why shouldn't a robot? Gradual change also offered a solution to two other problems:

a) Searching a large solution space is much easier if it can be broken down into a much smaller one that grows.

b) It also allows the gradual integration of sensors and actuators into the scheme by incrementally introducing them.

The theory behind the enlarged importance of Incremental Evolution is given below.

The complex organisms which surround us today are the result of over three billion years of evolutionary development, starting from simple initial life forms. The argument in the previous section is best illustrated by example. The first fossils evident in Precambrian rocks are those of simple, single-celled organisms.

Early multicellular animals, exemplified today by sponges, were amorphous creatures lacking the cellular specialization of later animals – for example, recognizable muscles, nervous system, gut and sensory organs. They lived in a simple environment leading a sessile existence, typically attached to a rock.

Jellyfish and their kin appear next in the fossil record. They can actively move and had simple sensory and nervous systems. Many also lived in a more complex environment (the open ocean), albeit simpler than later environments to come (with no need for even basic obstacle avoidance, for example).

One particular route of developments can be traced through various worms, echinoderms and simple chordates to fish, amphibians, reptiles and mammals as shown in simplified form in Figure 3-11. Four aspects of the organisms develop:

- Their body plan
- Their sensors and actuators
- The environment (at least as the organisms perceive it).
- The nervous system

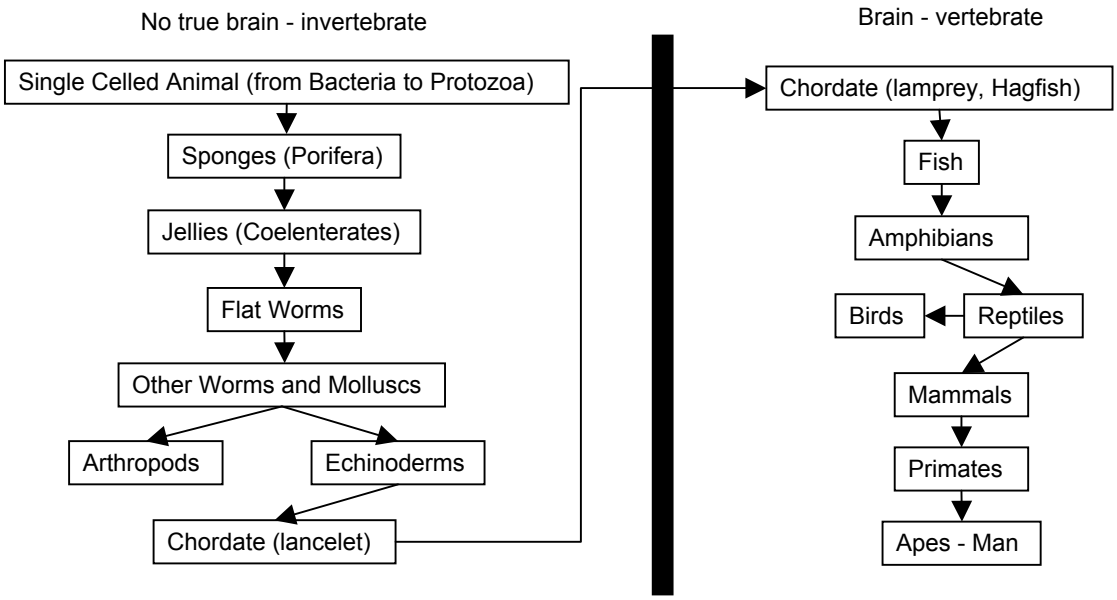


Figure 3-11 Evolutionary development

If one gives careful consideration to what is happening, one is drawn to the conclusion that, only through this process of gradual incremental change from one form to another (simple to complex), can the complexity inherent in biology build up. Otherwise the initial evolutionary search space would simply be too complex. It represents a march of progress, from simple forms to complex. It should be noted that this process is similar to the development of the human embryo in the womb leading to the term "embryology" which is sometimes applied to similar systems [MacLeod 1997]. However, although embryology is an interesting analogy to evolution, it is evolution itself which is important.

It is true that at each stage of this process, species have radiated and proliferated in form and function to fill available ecological niches; this happened most famously in the "Cambrian Explosion" [Gould 2000]. However, these early creatures, for all their variety and ingenuity of design, were simpler organisms than those which came later. Perhaps this is because, at any point of evolutionary time, organisms explore their genomic search space through mutation whereas the addition of truly new genes is a rarer occurrence, opening up new developmental possibilities.

One thing is clear. As an organism develops, it becomes impossibly complex to rearrange potentially billions neurons and trillions of connections in its network with each evolutionary step; the network must grow incrementally, building new layers upon old. This is the basis of Paul McLean's Triune theory of brain evolution [Restak 1979] and is illustrated in Figure 3-12. The deepest layers of the brain, located at its base, deal with the basic reflexes necessary for survival, such as breathing and heart beat. Higher functions, for example, basic intelligence are contained in upper layers. The top layer contains functions only found in humans and higher primates. This model is consistent with the neural network building up new structures upon old over aeons of evolutionary time following a path from simple to complex forms.

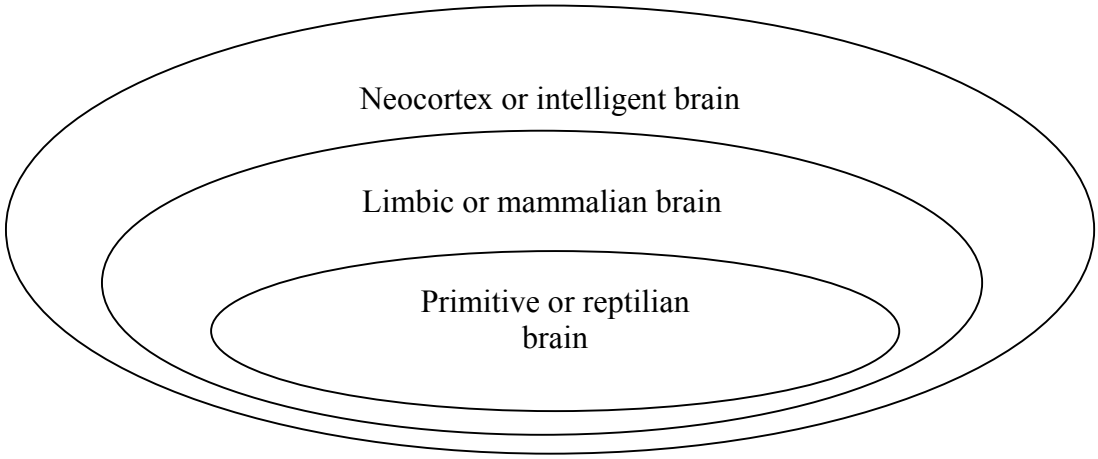


Figure 3-12 Triune theory of brain structure

Although this process has not been fully investigated from a biological point of view, it is clear that mutation does allow new and unassigned groups of neurons to appear from time to time. If these neurons are fortunate enough to be placed appropriately, they may become integrated into the network as a whole, so allowing it to grow and extending its capabilities. Fritzsch [Fritzsch 1998] discusses one such instance.

3.3.7 The Final System

Having covered the main biological arguments which are relevant to the approach adopted here, it is useful to briefly summarise them before continuing to consider how they are applied to the artificial system.

1. As an organism develops, its body plan, sensory system and interactive system (actuators) become progressively more complex.
2. This development is spurred by, and interacts with, an increase in environmental complexity, which in turn makes the evolution of intelligence and advanced behaviour more likely.
3. Both of these factors are facilitated by the gradual growth in the neural network due to small groups of new neurons becoming available from time to time through mutation. These new additions must add to the network without substantially changing previously evolved structures.

As discussed above, it was felt that a technique based on the 'direct growth method' was the best way to approach the project. Consider now how such a system might operate in a practical sense.

Starting with a simple evolution space (Figure 3-13) the system can grow by adding neural network modules.

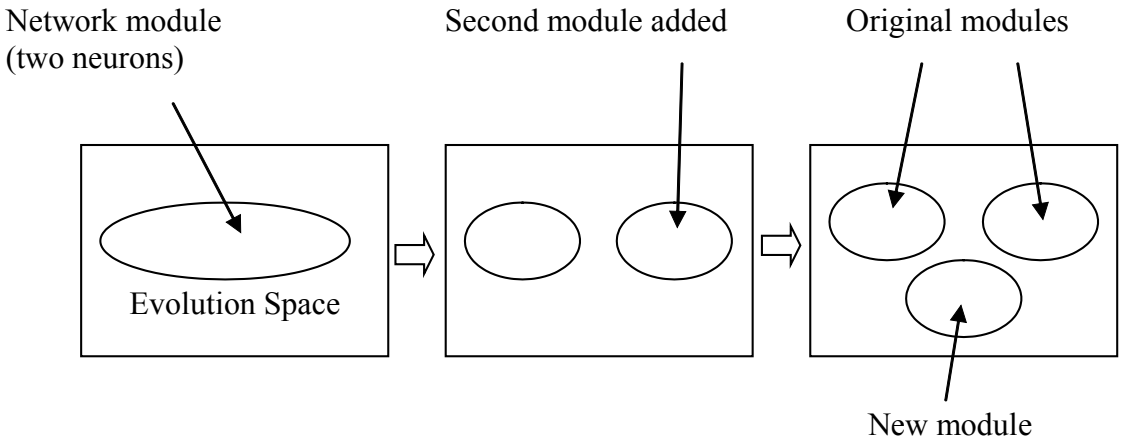


Figure 3-13 An evolutionary algorithm using direct modular growth (Reproduced by permission of McMinn)

At the start of the algorithm, for example, a minimum of two neurons could be used. These two neurons are considered a module (Figure 3-14). Each neuron in this module is connected to an actuator. As explained above, such an approach requires that the input sensors and actuators increase in complexity along with the network - in effect evolving the body plan of the robot. For example, a legged robot might start off with simple single active degree of freedom legs, each with a single sensor input perhaps measuring leg position and a single actuator output to move the leg as shown in Figure 3-15. Each neuron in the module is again connected to an actuator of the robot. The neuron connections and their respective weights are determined by an EA. Each module is trained until the maximum fitness is reached for that module, then another module is added. In this approach, previously trained modules are not retrained but retained (the weights of the connections and other neuron parameters are frozen). The fitness function used is a measure of the performance of the module or network based on the distance moved for a particular locomotion gait, within a specified time frame. Modules of neurons are added until the maximum possible fitness is reached for a particular function.

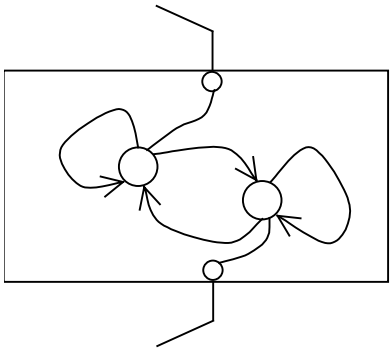


Figure 3-14 Initial module

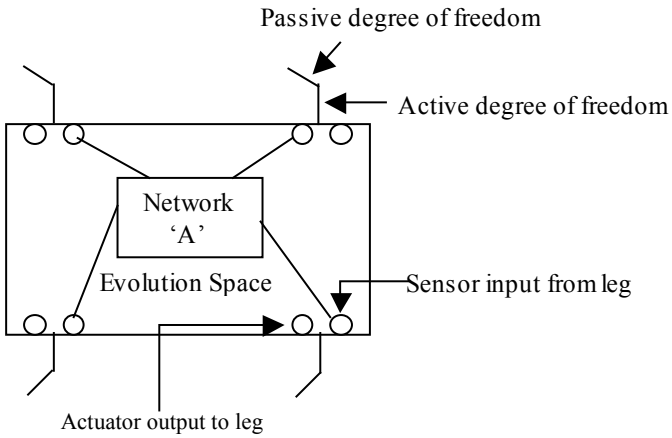


Figure 3-15 Robot's initial body plan

Once the system can control its simple legs, a new network (network 'B') is added incrementally (as shown in Figure 3-16) and evolved to control the extra degrees of freedom. The control system for a prosthetic limb might also proceed along similar lines, starting with gross movements and working down, finally, to digits. Likewise, a sensory system like vision would start, perhaps with a single detector cell (an eyespot) – only able to perceive light and dark and evolve in complexity from there. Obviously, any complex artificial organism would start life (as with both evolutionary and developmental biology) as a simple group of cells and then develop in a similar manner.

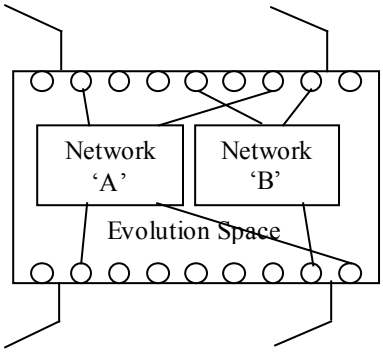


Figure 3-16 Evolution of more complex "body plan"

To further illustrate the technique, consider the situation shown below. The leg has two degrees of freedom A and B, Figure 3-17 a).

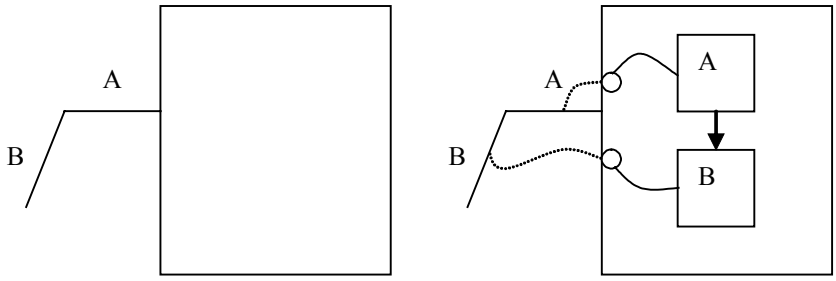


Figure 3-17 Interaction between modules

Assume segment B starts moving after A has fully moved backwards. To do this, assuming there are two networks (Figure 3-17 b), network B needs inputs from network A because it needs to know when Leg A has moved to one extreme. Again, in biology there is nothing stopping any neuron being connected to any other. It is not feasible to forecast which neurons in module A are needed by B. So, the connections and their respective weights from A to B need to evolve.

Therefore, in the system used here, the evolutionary algorithm chooses the connections and the weights for the connections. The evolutionary algorithm should be able to make the weight for a particular connection zero in order to improve the fitness. Once the fitness has reached its peak value for this configuration, the initial module weights are stored and other modules are added.

Having “wired the simple system up”, the algorithm next adds another module and more inputs and outputs to the outside world. The process is then repeated, except that the previously wired modules are retained and only the connection weights of the newly wired module are changed.

An absolutely critical aspect of this approach is that the algorithm starts with only a few inputs and outputs and builds up by adding to these as well as growing the modular network and so the whole robot develops as a system. As this happens, the robot’s environment may also become more complex and challenging. Therefore, not only does the network grow, but so does the robot’s body plan, its access to sensors and actuators and the environment in which it finds itself.

3.3.8 Amalgamating the Function Methods

It should be noted that the five methods described in the paper were simply suggestions for further research and had not been implemented in reality. As such, a detailed description of the operation of each was not presented. This was to be the purpose of this project.

At the start of the present project, all the strategies described above were considered and reviewed. The idea was to compare them. However, some, like the cellular coding and altering existing algorithms, had already been investigated by other researchers. The fractal method, although suggested by other workers was not thought practical - it was difficult to see how a working system could operate.

This left the ‘Direct Growth Method’ and the ‘Biologically Inspired’ method. Careful consideration indicated that both these methods achieved the same ends. They placed small clusters of neurons (modules) in an evolution space and then connected the clusters internally and externally using an EA. All the connections are trained when a new module is added. This is very similar to the existing methods (GA, Back Propagation, GP and others) used for training neural networks. The extended Direct Growth Method offers an alternative training scheme, in which previously trained networks are retained and not retrained. This method supports Triune’s theory on brain structure and evolutionary development of complex organism.

It was felt that investigating all of these methods separately was a waste of effort. Given this, it was decided to choose the 'Direct Growth' method, as this appeared to be the more realistic and simpler of the two to implement.

3.4 Summary

Initial work concentrated on growing ANNs and investigating the effect of modularity on the network and its evolution based on an ANS model. It was discovered the available EAs were not capable of evolving a system which mimicked some important points in the biology. Therefore, a review of biology was undertaken to discover the reasons why biological systems allow such complexity to evolve.

The conclusion of the review was that the biological evolutionary system is quite different from current artificial evolution. In biological evolution, DNA rather than coding the network, codes the building blocks which fit together rather like a jig-saw puzzle. These building blocks interact to form a system. There are no simple ways to simulate this process in a computer; therefore, a way of growing practical neural networks was needed. The other important point about biological evolution is the development of the organism itself. The biological justification in Section 3.3.6 shows that the organisms start from simple forms and become more complex as the environment becomes more challenging.

The conclusion from biology was that, as the organism evolves from simple to complex, previously evolved structures are retained and not retrained. This process is similar to adding layers on top of others like onions (as described in the Triune brain theory). Lessons from biology can be used in the artificial system.

Five different methods for creating modular networks were proposed in the "Evolution and Devolved Action" paper, but there were no technical details on how to implement these methods. Some of the suggested techniques are easier to implement than others. By looking closely at all five methods, one can see that, in essence, they are almost the same. All the algorithms concern the evolution of modular neural networks. The problem of evolving large ANNs in a modular fashion still remains difficult because of the huge search space involved. Incremental Evolution seems to offer a ready answer. The biological justification is described in the development of

animal kingdom and the Triune brain theory (Section 3.3.6). Development of the body plan (sensors and actuators), nervous system, and the environment the organism is interacting with are the key factors in determining the growth of the organism.

The extended Direct Modular Growth method was chosen to be implemented for the purposes of this research. The technical operation of the algorithm is discussed in more detail in Chapter 5.

Chapter 4

Literature Review

4.1 Introduction to the Chapter

In this chapter, previous work related to this research is reviewed. The chapter will begin with a brief review of the problem in context and then discuss related research in the field. Finally, a summary will put the research presented here into context with the reviewed work.

4.2 Multilayer Perceptrons

Artificial Neural Networks (ANNs) have been used widely in research and for practical applications since the early 80's. Most of the work used fixed ANN topologies and standard off the shelf learning algorithms like Back Propagation (BP) [Yao 1997]. The learning algorithm generally only trains the connection weights and unit bias. The problem of designing a near optimal ANN architecture for an application is still largely done on a trial and error basis. However, it is an interesting issue because there is strong biological evidence that the information processing capabilities of an ANN are determined by its architecture [Happel 1994].

4.3 Evolutionary Artificial Neural Network (EANNs)

The evolution of ANN connection weights and architecture using Evolutionary Algorithms (EAs) (Genetic Programming (GP) [Holland 1992], Genetic Algorithm (GA) [Goldberg 1989], Evolutionary Strategy (ES) [Back 1991], and Evolutionary Programming (EP) [Fogel 1966]) provides an alternative approach to a fixed size ANN and the drawbacks of the 'standard' training algorithms mainly due to their gradient descent nature [Sutton 1986].

The first suggestion that simulated evolution could be used to design and train ANNs was described by [Bremermann 1968]. The real potential of using an EA to enhance the design of ANNs was not revealed until late 1980's and early 1990's because of a lack of computer processing power [Fogel 1994].

EAs can be used to search for an optimal architecture in a topological search space. Because of the problems of searching a large space, much research has been carried out into this aspect [Koza 1991] [Miller 1989] [Kitano 1990] [Harp 1989], which concentrates on the evolution of ANN architecture (the number of nodes in the network, the number of layers and the connection topology).

A key issue in evolving an ANN is to decide how much information about the architecture should be encoded into the genetic representation. There are two broad types of encoding scheme.

Firstly, in the Direct Encoding Scheme, the entire neural network structure is directly represented by a string (chromosome). In this scheme, each connection of an EANN is specified directly by a binary representation [Oliker 1991] [Alba 1993]. For example: a '1' for the existence of a connection and '0' for no connection. The resulting string has a one-to-one mapping of the corresponding architecture. Direct encoding is often represented by a connectivity matrix [Vonk 1997]. This matrix has size N x N, where N is the maximum number of neurons in the network. Figure 4-1 shows the direct encoding scheme for a feedforward network. This method can also be applied to recurrent networks.

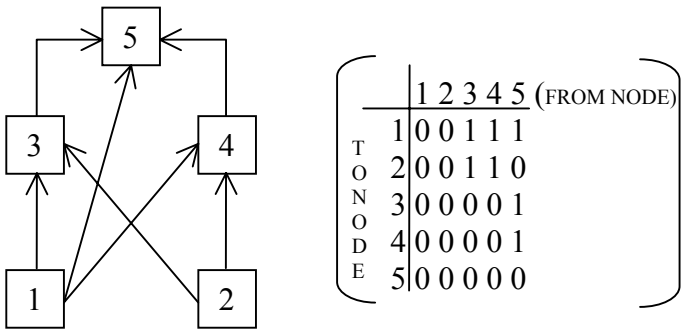


Figure 4-1 Direct encoding of a feed forward network, its connectivity matrix and its binary representation

In the Direct Encoding Scheme, the string grows longer with increasing size of the network. Therefore, direct encoding is only suitable for handling small networks. In order to reduce the length of the chromosome, the Indirect Encoding Scheme has also been used by many researchers [Kitano 1990] [Harp 1989] [Harp 1990]. Here only the important parameters of the ANN are encoded in the chromosome. In this method, detail about the architecture is either specified by a Parameter or a Grammar Encoding Scheme.

The Direct Encoding Method is sometimes called a low-level representation, because the entire architecture is encoded into a chromosome. When high level representations are used, the chromosomes do not contain a complete network mapping. This is often called a Parameterised Encoding Scheme. The information being encoded into the chromosome is more abstract; for example, the number of hidden layers, the number of neurons in each layer, number of connections between two layers, the type of node transfer function, etc. [Alba 1993] made a distinction between structure, connectivity and weight optimisation. The network structure is defined in terms of the number of layers and the number of neurons in each layer.

Grammar Based Encoding schemes are often used to encode large neural networks. Kitano [Kitano 1990] used a modified version of the graph generation system [Doi 1988], which includes a set of graph generation rules that construct connection matrices - each connection matrix corresponding to a directed graph. The graph generation rule consists of a left-hand side (LHS) and a right-hand side (RHS) element. Each rule on the LHS rewrites a character into a 2 x 2 matrix of characters on the RHS. The LHS can be presented implicitly by the rule's position in the chromosome. Each position in a chromosome can take one of many different values, depending on how many nonterminal elements (symbols) are used in the rule set. For example, the nonterminals may range from "A" to "Z" and "a" to "z". Since there are 26 different rules, whose LHS is "A," "B,"…,"Z" respectively, a chromosome encoding all of them would need 26 x 4 = 104 alleles, four per rule. Figure 4.2 summarizes this method.

Kitano demonstrated better results with this scheme than direct encoding when evolving simple ANNs (such as XOR and simple encoders). Given a set of developmental rules, an ANN architecture can be generated by applying the rules in three steps as shown in Figure 4-2. Other types of Indirect Encoding Scheme include Gruau's Cellular Encoding [Gruau 1992][Gruau 1994], Boers and Kuiper's L-systems [Boers 1992] and Merrill and Port's Fractal representation [Merrill 1991].

Left Hand Side
S = A B
    C D

Right Hand Side
A = a a B = a i C = i a D = a a …
    a a ,   i a ,   i c,   i e

a = 0 1 c = 1 0 e = 1 1 i = 1 0 …
    1 0 ,   1 0 ,   0 0,   0 1

Figure 4-2 Example of some development rules used to construct a connectivity matrix. S is the initial element.

4.4 Growing ANNs

Advanced Competitive Networks such as Adaptive Resonance Theory (ART) [Carpenter 1986; Carpenter 1987] and Grow and Learn (GAL) [Alpayin 1994] networks are interesting because they solve some of the fundamental ANN architecture problems and also represent and early attempt at network growth [MacLeod 2001].

The more general papers on ANNs which grow fall into three categories:

I. Network which grow by adding layers
II. Network which grow by adding neurons
III. Network which alter by changing their connections

Many papers in this area refer back to work by [Ash 1989] who outlines a network, with one hidden layer, which grows by adding another neuron to that layer when necessary.

[Chakraborthy 1995] takes this idea further. His network grows by adding units to its hidden layer. This is accomplished by starting with two networks and combining them.

[Vinod 1996] outlines an algorithm which grows one neuron at a time. A unique aspect of his approach is that the neuron is not added in an arbitrary position, but in the position calculated to give the maximum reduction of error.

[Ferran 1991] investigates different sizes of networks and their capabilities by adding layers and neurons to the network structure. His paper provides an extensive account of the performance of different network architectures. Although he does not demonstrate an actual growth algorithm, the paper presents many ideas on the subject.

[Kozma 1995] looked at the reduction of connectivity in the network by allowing weights, that are not being reinforced through BP weight changes, to decay to zero. This produces a skeleton network. Other similar work is by [Mozer 1989].

One, very interesting paper comes from [Anderle 1995]. The importance of his contribution is that he considers recurrent networks which are inherently stable and grow in such a way that their stability remains assured. Anderle's method starts with an unconnected network and grows connections, one by one, until the desired result is achieved.

4.5 Modular Neural Networks

When dealing with a complex problem, a monolithic neural network often becomes too large and complex to design and manage. One way around this problem is to design a Modular Artificial Neural Network (MANN) system consisting of multiple simple networks [Yao 1996]. According to Gruau's [Gruau 1992] definition, an encoding scheme is modular if the genotype can be decomposed into some parts that specify the organizations of sub-networks, and other parts that describe how to interconnect these sub-networks. Thus, this allows the same pattern of connectivity to be expressed several times within the network. Gruau demonstrated this by evolving a

sub-ANN for controlling one leg of an 8-legged robot and put together 7 copies of the module to control the other legs.

There are many ways to design Modular ANNs [Jacobs 1991a] [Jacobs 1991b], [Battiti 1994], [Hansen 1990]. Most of them follow a two-stage design process. Firstly, the individual modules are generated; secondly they are integrated. In most of the applications the modules are simple Multilayer Perceptrons. The number of modules and ANN architectures within each module is determined by the designer or by a trial and error process. There is no interaction between the modules until they are integrated together.

A system with a complex input/output relationship can be decomposed into simpler systems in several ways. There are four common methods of putting modules together to form a modular neural network.

Firstly, we will look at input decomposition. A system with multiple inputs can be decomposed into subset of modules and inputs. This is illustrated in Figure 4-3.

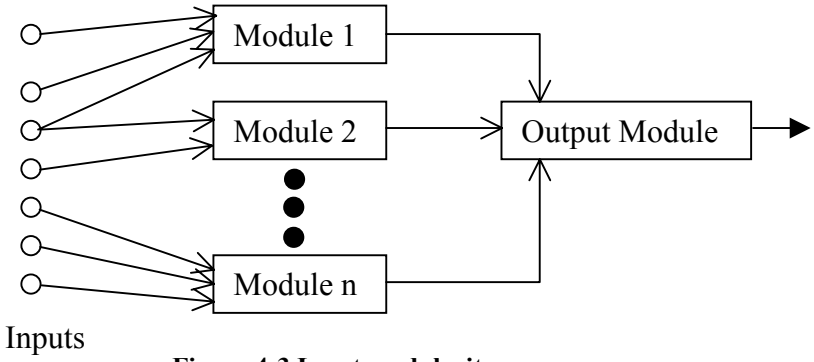


Figure 4-3 Input modularity

This approach is considered to be modular because a large input array is decomposed into several small arrays. Information from smaller arrays is easier to understand and to process. This is the essential feature of the Neocognition, developed by Fukushima for visual pattern recognition [Fukushima 1980, 1987, 1988, 1993].

The second approach is called output decomposition. A neural network can be designed for each subtask and the overall result is a collection of the results of smaller neural network modules. The basic idea is illustrated in Figure 4-4.

Rueckl [Rueckl 1989] found that training time was shorter when separate networks were used to identify the location of and provide recognition of an object in an image.

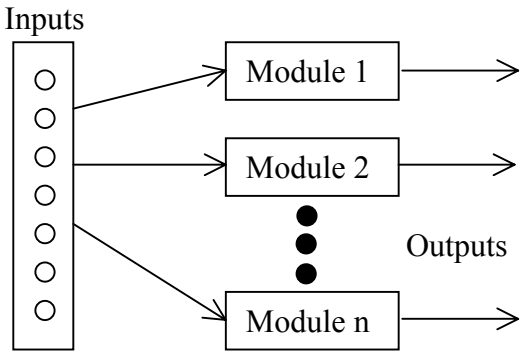
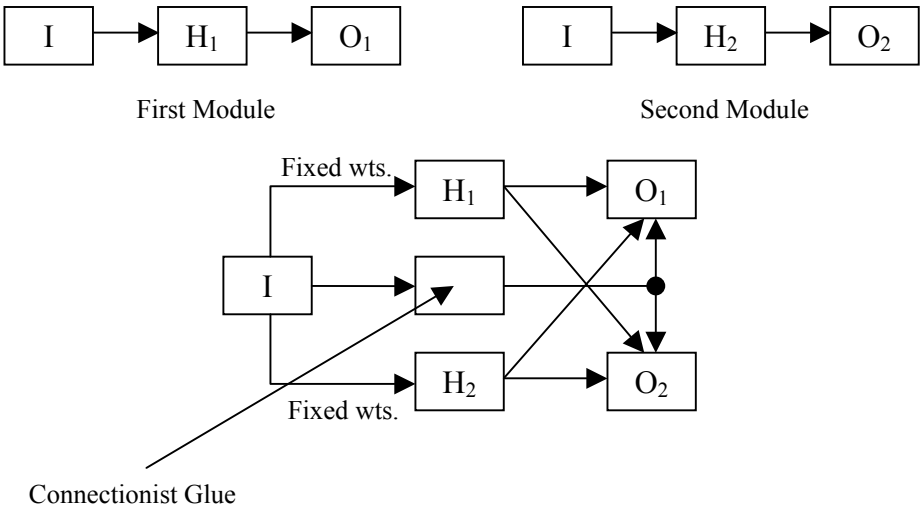


Figure 4-4 Output Modularity

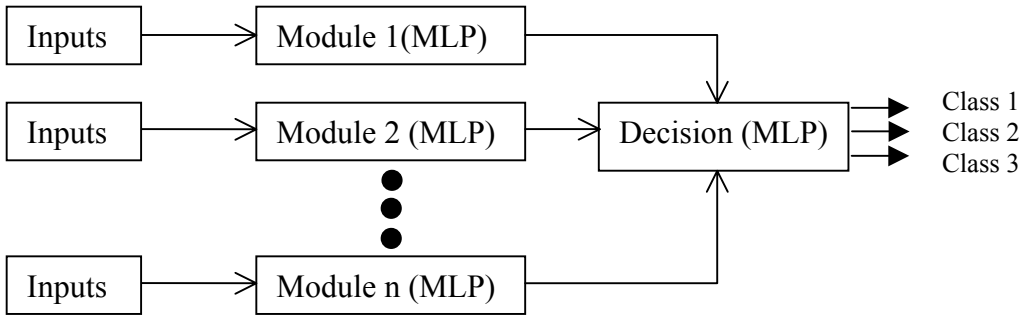
[Waibel 1989] has devised a technique called connectionist glue to train modules for different tasks and then combine them as shown in Figure 4-5. He found that performance improved in the network's capabilities using this approach.



I = Input, Hn = Hidden Layer, On = Output Layer, wts. = connection weights, n = integer

Figure 4-5 Waibel's connectionist modular network

The third approach is termed hierarchical decomposition. A system with multiple outputs and inputs can sometimes be decomposed into simpler multi-input and output modules arranged in a hierarchy, as illustrated in Figure 4-6. [Schmidt 1998] illustrates this concept.



MLP = Multi Layer Perceptron

Figure 4-6 Hierarchical organization

[Happel 1994] attempted to introduce modularity into the network based on a special neural network called CALM and is very similar to the third approach described above. CALM stands for Categorization And Learning Module. CALM has been especially developed as a building block for modular interactive neural networks. All the connections inside the CALM modules are non-modifiable (the architecture of the module itself remains fixed). A CALM module consists of a number of representation modules (R-nodes) which are fully connected to inputs through modifiable connections. The inputs to the CALM module are from another CALM module or from an activation pattern. When a number of CALM modules are used in a network, it is said to be modular.

Another simple kind of modularity involves pipelining, shown in Figure 4-7. This is useful when the task requires different types of neural network modules at various stages of processing. [Yang 1992] presents an illustrative example of the hierarchical approach. Outputs from one module are fed into the next module. The whole network's connections are retrained until a solution is found.

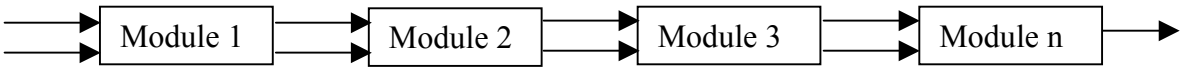


Figure 4-7 Pipelining architecture

n = an integer for module number

The fourth approach is called Combining Outputs of Expert Modules. Figure 4-8 illustrates the basic idea of this approach. Expert Networks are trained and combined using gating networks [Jordan 1994]. A variation of this approach is the growing multi-expert network by [Chu 2000]; here the network is added to incrementally. The local experts are added to the network strategically based on network error. [Perez 1998] evolved a modular neural network with an expert module for handwritten digit recognition.

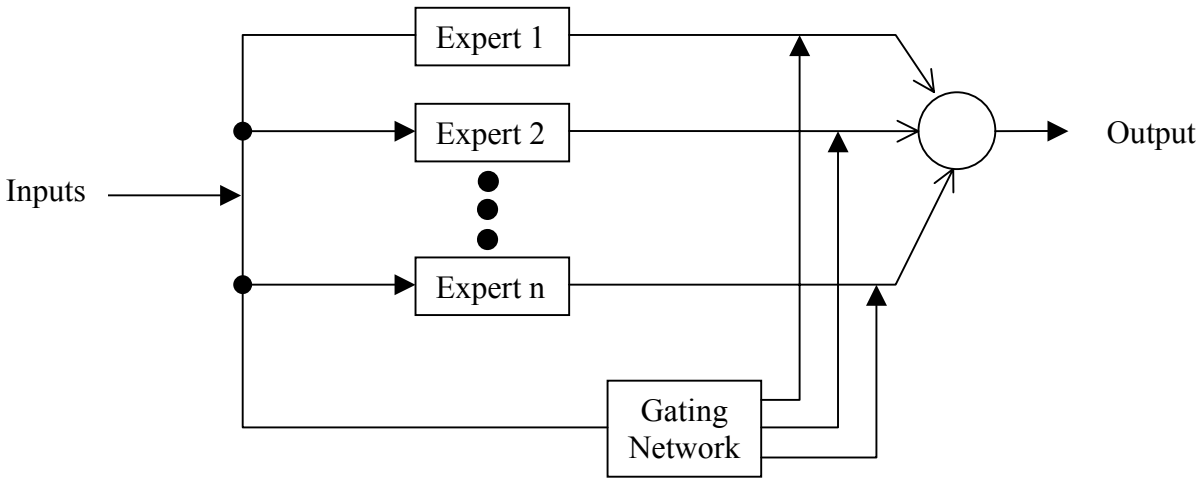


Figure 4-8 Basic structure of mixture of expert networks

All the modular neural networks described above are based on feed-foward layered networks. Described below are works by other researchers using non-structured neural networks.

[McMinn 2002] used an alternative strategy for the topology of a Central Pattern Generator (CPG) network. In his work, neurons in the modules are randomly connected. The network was said to be modular because the CPGs have been split into two functional units. The task of the first unit was to oscillate. For this, the CPG previously evolved for the biped walking pattern was used, as it produced alternating oscillations from each output. The second unit was a pattern generator taking the oscillating inputs from the first unit and producing the appropriate gait patterns outputs. The concept is illustrated in Figure 8 of Section 2.3.

[Gomi, T et al. 1998], [Takamura, S et al. 2000], [Hornby, G.S et al 1999], evolved gaits for legged robots. In all these works the gaits were generated from a base level using evolutionary techniques (the weights and connections of the ANN topology were trained until a successful leg movement is found).

4.6 Simple Incremental Learning of ANNs

A different approach to determining the architecture of a neural network is to modify the network topology as part of the learning process. This typically starts with an initial network topology and then adds new units in order to learn a set of examples. The final topology of the network is determined by the algorithm and the criteria for adding a new cell depends on the algorithm chosen. There are about six established incremental learning algorithms. They can be classified into those which operate on neural networks (whose input and output pattern space are of a continuous nature) and those which work with networks (whose input and output space are of a discrete nature).

The six algorithms are the Tiling Algorithm [Mezard 1989], the Tower Algorithm [Gallant 1990], and the Upstart Algorithm [Frean 1990] for discrete networks; the Cascade-Correlation Network (CasCor) [Fahlam 1990], the Restricted Coulomb Energy Network (RCE) [Reilly 1982], and the Resource-Allocation Network (RAN) [Platt 1991] are associated with continuous networks. These learning algorithms are applied on feed-forward layered networks.

Before discussing this further, it is necessary to consider the commonest modification to all the six algorithms: the Pocket Algorithm developed by [Gallant 1986]. The Pocket Algorithm is designed to deal with data sets which are not linearly separable. The simple Perceptron Learning Algorithm is guaranteed to find an exact classification of the training data set only if it is linearly separable. If the data set is not linearly separable, then the algorithm fails to converge. The Pocket Algorithm involves retaining a copy of the set of weights which has so far survived unchanged for the longest number of pattern presentations. Then a new neuron is added to the network and the connections to the new neuron only are trained. This process is repeated until convergence is reached.

Since the algorithms are rather similar, only one discrete and one continuous example are discussed in the following section.

The Tower Incremental Algorithm was devised by [Gallant 1990]. It starts by defining and training a simple Fully Connected Feed Forward Neural Network. The neurons in the network are of the Simple Sigmoid type. If the network results are not satisfactory, all the weights are frozen and a new output cell is connected to all the input cells and the previous output cells. The process is then repeated so that all the new weights are trained. If the results are still not satisfactory, the weights are frozen and another new cell is inserted. This process of adding new cells continues until the result is satisfactory. Figure 4-9 shows the operation.

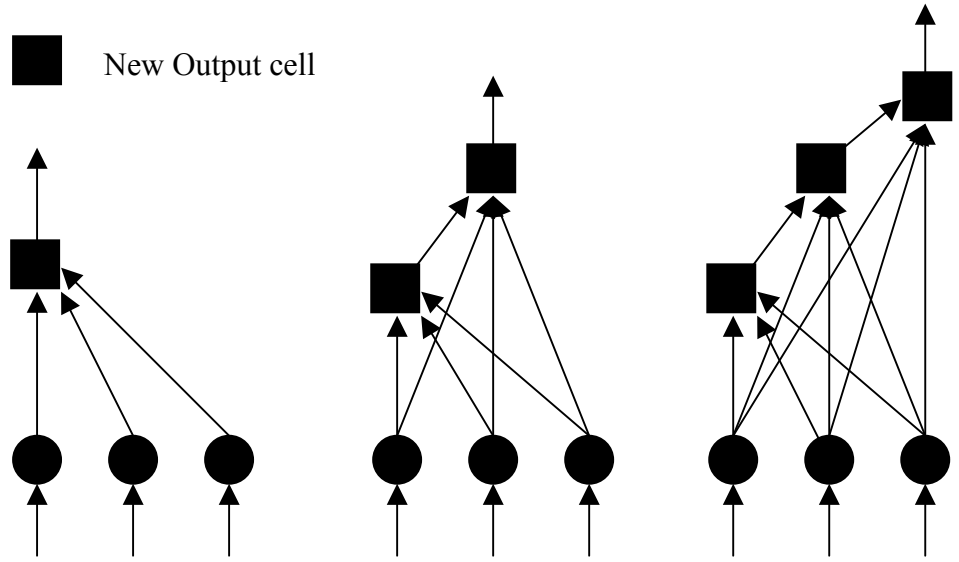


Figure 4-9 Development of Tower network topology

The Cascade Correlation Algorithm is a good example for continuous networks and was developed by [Fahlam 1990]. CasCor addresses the issues of evolving network architecture by adding new hidden neurons one by one.

The algorithm starts with a minimal topology, consisting only of the required input and output units plus a bias unit that is always equal to 1. Both layers are fully connected. The network is trained until no further improvement in error is obtained. Then, a collection of new candidate cells is generated. All the candidate units are connected to the every input unit and to the existing hidden cells, but not to the network output units. A number of training sets are applied to the candidate cells, and

the input weights are adjusted after each pass to maximize the magnitude of the correlation between the output of candidate cell and the network output neurons. When the correlation stops increasing, the candidate unit with the highest correlation is selected and the other candidate cells are discarded. This selected unit is installed in the network and its input weights are frozen. Again, all the connections leading to the network output cells are trained until the network error no longer decreases. Hidden units are added like this until the overall error of the network falls below a target value. Figure 4-10 shows the operation of the CasCor network.

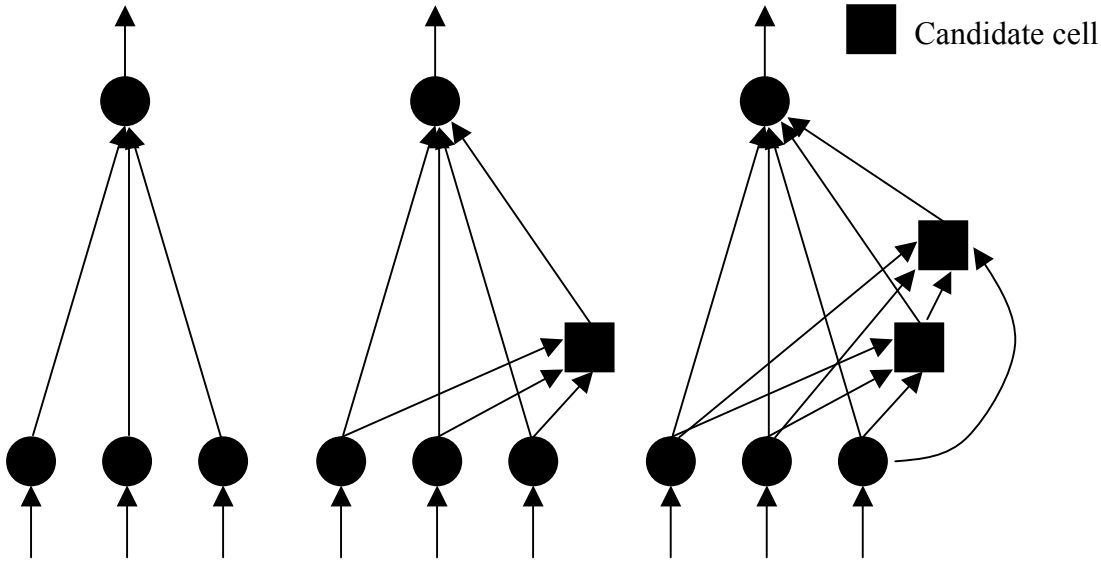


Figure 4-10 Development of CasCor network topology.

Another interesting piece of work comes from [de Garis 1993]. In his paper, he describes incremental evolution by inserting a small portion of an earlier chromosome (which results from a previous phase of evolution) into a later, larger chromosome for a second phase of evolution. He found that by doing this, the network evolved faster.

[Fritzke 1994] describes an incremental algorithm using Growing Neural Gas (GNG). Growing Neural Gas is an unsupervised network model, which learns topologies [Fritzke 1995]. A set of units connected by edges is distributed in the input space with an incremental mechanism which tends to minimize the mean distortion error.

4.7 Evolving More Complex Systems

So far, methods of determining ANN structure for simple applications have been examined. Even although some of the indirect representations, such as Kitano's methods, provide a solution suitable for evolving a large neural network, they are not designed to evolve a "*system*". The definition for *system* in this context can be found in Section 1.3. In this Section we shall look at the approaches to the evolution and development of control architectures in animats [MacLeod 1999] (animal-like robots which are the commonly used as test beds for 'systems').

The work of Gruau [Gruau 1992] [Gruau 1993] encodes grammar trees in a chromosome. The grammar tree represents nodes which are labelled with character symbols. These characters represent instructions for unit development that act on the cell. This encoding scheme is called cellular encoding. Gruau's chromosomes are subjected to genetic operators. This encoding scheme has been used by [Gruau 1994] to evolve a neural network capable of controlling the motion of a six-legged robot.

The work of Nolfi and Parisi [Nolfi 1991] used genes that describe the developmental fate of a given neuron to discover a neural architecture. This architecture enables an animat to move in an environment and to capture food. Results of the evolved architecture tend to be structured in functional sub-networks. The extension of this work [Nolfi 1994], considered both the genes and the environmental influence in the neural development.

The work of Vaario [Vaario 1993] [Vaario 1994] approach takes as its starting point environmental effects on the development of neural networks. This approach is inspired by Lindermayer's Systems [Lindermayer 1968]. In Vaario's work, each cell is characterised by a set of attributes and a set of production rules. The production rules are used to model various morphogenesis processes such as cell division, cell fate, axon and dendrite growth, etc. Vaario's approach has been used to develop the nervous system of an animat with two sensors (which allow the animat to receive stimuli) and four actuators which allow it to move.

The work of Cangelosi [Cangelosi 1995] is concerned with the evolution of animats possessing motivational sensory units, processing units and motor units. The sensory units inform some internal needs (hunger or thirst). This information is relayed to processing units, which are in turn used to control motor action. The control architecture for the animat is a bidirectional network that develops from an initial egg cell. The initial egg will go through five cell divisions and a migration cycle followed by five cycles of axonal growth.

The developmental process begins with the egg located in the centre of the evolution space. At the end of cell division and migration, 32 cells are created. The functionality of the cells is determined by the location and the cell type. Thus, neurons at the lower end will work as a sensory network. Neurons in the upper band will work as a motor unit and neurons which end up in the intermediate band will work as hidden units. At the end of cell division and migration, an axonal growth process begins. During five growth cycles, each neuron grows its branch axon according to the corresponding parameters (axon's angle and the length of branching, connections weight) specified. In order to evolve such a control architecture, genetic operators are applied to each parameter of the population. The approach simulates the process of axonal growth that determines the connectivity of a network.

4.8 Body-Brain Evolution

This section describes research on the simultaneous development and evolution of both an animat's control architecture and its morphology.

Dellaert [Dellaert 1994a] was concerned with the development of a whole artificial organism (including both the nervous system and body). His system worked by extracting some of the beneficial properties from biological developments. The genetic regulatory network is the principal component in his model. Each cell in the system will respond to the expression of some gene. The morphology of the animat is a two-dimensional square consisting of cells of various types (sensor, axon, and actuator). The cell types are subject to genetic operators. In order to evaluate the capabilities of their encoding scheme, Dellaert and Beer have evolved a simple animat that roughly reproduces the relative positioning of sensors, actuators and control

system in a simple artificial agent. This animat exhibits bilateral symmetry, with sensors (cell-type 2) placed sideways at the front, with actuators (cell-type 4) placed sideways at the back, and with a control structure made of neural tissue (cell-type 1) connecting them. More complexity has been introduced in the revised version of this method in [Dellaret 1994b]. Related work is described by Lee [Lee 2003]

Sims [Sims 1994a] [Sims 1994b] encodes directed graphs of both the morphology and the control architecture in the genotype. The morphology contains a description of the dimension of the blocks. The control architecture describes the neural circuitry of the corresponding morphological unit. The genotype is subject to genetic operators. The phenotype contains sets of rectangular joints at the centre of opposing faces with one, two or more degrees of freedom. Each block can house a number of neurons. These neurons can receive information from the same block or from any other blocks. In this way a signal can propagate throughout the body. Every animat is evolved based on a simulated virtual world, with which it interacts realistically, thus allowing its fitness to be assessed. Sim's approach allows virtual animats to swim, walk or display following behaviours [Sims 1994a].

4.9 Context of the Current Research

The research outlined in this thesis describes a system that allows a neural network, which is used to control a robot, to evolve in a structured but open-ended way. In dealing with such a complex problem, a monolithic neural network often becomes too large and complex to design and manage. The only practical way around the problem is to design modular neural network systems consisting of simple modules. While, as has been reported, there has been some work on combining different modules in a system in the various fields of neural networks, statistics and machine learning, little work has been done on how to design those modules automatically and how to exploit the interaction between individual module design and module combination [Liu 1998]. The approach used here addresses the issues of addition of modules to networks, the automatic determination of the number of modules and neurons and the exploitation of the interaction between individual modules. None of the other work surveyed examines these issues in the context of an evolving network.

Growing ANNs (Section 4.4) and Simple Incremental Learning of ANNs (Section 4.6) can be classified into two categories. Firstly, constructive algorithms starts as a minimal network (a network with topology) and adds new layers, nodes and connections, if necessary, during training. Secondly, destructive algorithms do the opposite – for example, starting with a fully connected network and deleting the unnecessary layers, nodes, and connections during training. Most of the networks discussed in these sections are feed-forward layered networks.

The technique explained here places the robot in a developing environment, and allows both this environment and the robot's body form, sensors and actuators to become more complex and sophisticated as time passes. Again, although some work presented in Section 4.7 and 4.8 of this chapter has a passing similarity to this, it is different in almost all detail to the research reported in this thesis.

Finally, in the work presented in this thesis, modules of neurons are added incrementally until a function is mastered. Each module is trained until its fitness does not increase further. The weights and connections of the added module were retained and further modules are added. Only the weights and connections of the new module are trained. This is similar to new structures being built upon older ones (while retaining the older structure's functionality). This, too, is quite unique in detail among other published research.

In summary, this research proposes an unique approach, wherein such complex general behaviour is learned incrementally, by starting with simpler behaviour and gradually making the task more challenging and general. It is hoped that, as the network develops, intelligence will eventually emerge.

4.10 Summary

This chapter has reviewed the important work related to this project which has previously been carried out, in the following areas:

I. Evolutionary ANNs
II. Growing ANNs
III. Modular ANNs
IV. Incremental ANNs
V. Complex systems and
VI. Body Brain Evolution

The research presented here has been put into the context of existing literature, and the originality of the work emphasised.

The next chapter provides detailed explanation about the growth components for the evolution of modular artificial neural networks.

Chapter 5

Components for Evolution of Modular Artificial Neural Networks

5.1 Introduction

In this chapter, the methods and components used for modular evolution of Artificial Neural Networks (ANNs) are discussed.

The first section describes the two different neuron models that have been used in the research. The ANNs used to produce locomotive gaits are based on two different types of actuator; both of these actuators are illustrated and explained in the second section. The third section describes the development of the robot's morphology and the ANN which controls it. Finally, the Evolutionary Algorithm and Modular Growth Algorithm are described in detail. The chapter also provides a foundation for understanding the remaining chapters in the thesis.

5.2 Neuron Models

The first neuron model, which was used to simulate motor functions is shown in Figure 5-1. This is a 'spiky' or 'pulsing' unit which loosely simulates the operation of biological motor neurons. As a consequence of the complexity of the nerve cells found in the brain, simplifications were introduced in the functionality of the model. The model was designated the MMM neuron (after its designers MacLeod, Muthuraman and McMinn). This neuron is very similar to the one developed by McMinn [McMinn 2002a] for use in legged robot systems and is based on the known behaviour of motor neurons, especially in terms of Long and Short Excitatory Post-Synaptic Potentials [Brodal 1992].

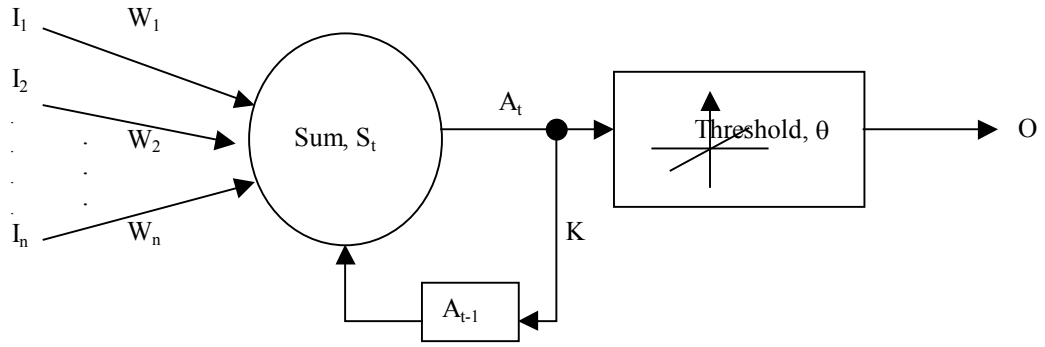


Figure 5-1 MMM Neuron model

The neuron operation and formulae are as follows;

$S_t = I_1W_1 + I_2W_2 + \ldots I_nW_n$ *At time t*

$A_t = S_t + A_{t-1} K$ *Neuron activity at time t. K is a constant (leaky integrator)*

If $t_1 > 1$ then *t_1 is a constant defined later*

If $A_t > \theta$ then *θ is the threshold*

O = 1 *for t_1 time periods*

O = -1 *for t_2 time periods*

- Unit behaves as a pulse-width modulated neuron

If $-\infty < t_1 < 1$ then

$$O = \frac{1}{1+e^{-S}}$$

Unit behaves like a Multi Layer Perceptron (MLP) neuron.

The chromosomes for genetic training are as follows:

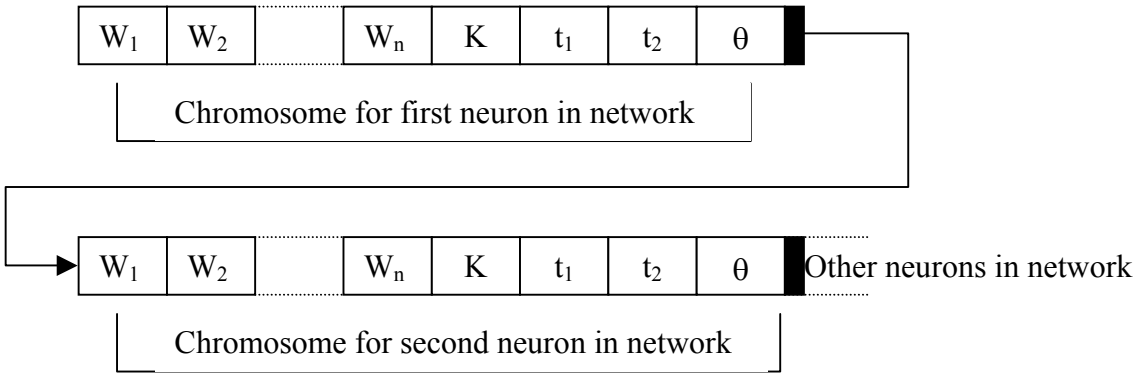


Figure 5-2 First Neuron Model Chromosome Parameters

Neuron Parameters	Description	Parameters Value
W_1 to W_n	The weights of the neuron	Unconstrained, initial values between –1 and +1
K	Weighting constant of previous inputs	$0 \leq K \leq 1$
t_1	*On* time of neuron	$-10 \leq t_1 \leq 100$ ($-\infty$ to 0 = sigmoid neuron)
t_2	*Off* time of neuron	$0 \leq t_2 \leq 100$
θ	Neuron firing threshold	Unconstrained, initial values 0 to 0.5

Figure 5-3 Neuron Parameters Table

The operation of the model is as follows: If the sum of the weighted inputs (I_1W_1 + I_2W_2 +… etc) plus another term (A_{t-1} K) is greater than the threshold, then the neuron fires and produces a pulse for time t_1 followed by no pulse for time t_2, (Figure 5-4). The (A_{t-1}) term in the formula is the activity of the neuron in the last time step and K is a constant term (K < 1). The (A_{t-1} K) term means that the neuron's activity depends both on the current weighted inputs and also on the previous ones – so "smoothing out" or integrating short pulses. Such a response is commonly known as a "Leaky Integrator" [Arbib 1989]. If the evolutionary algorithm sets t_1 to be less than 1, the neuron behaves as a "standard" sigmoid perceptron. Similar neurons occur in the biological motor system [Brodal 1992].

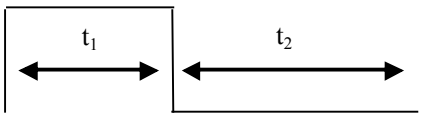


Figure 5-4 Neuron Output

The neuron parameters and connection weights are coded into an evolutionary training algorithm, as shown in Figure 5-2. The initial weight values for the ANN are randomly chosen between -1 and +1. The weighting constant (K) of the previous input is selected between 0 and 1. A K-value of equal or greater than 1 indicates positive feedback and the neural network can be said to be in an unstable state. There are also some constraints on the time factors (t_1 and t_2), otherwise simulations become unrealistic, the maximum "on" and "off" time for the neurons being fixed at 100 time steps. There is a 10 percent probability that the evolutionary algorithm will evolve a standard McCulloch-Pitts neuron as described above.

The neuron firing threshold value is initialized randomly between 0 and 0.5. Neuron connection weights and threshold values were not constrained to any limit. This information is summarized in Figure 5-3.

Genetic operators are applied to the string in the same manner as the traditional Genetic Algorithm approach. The neuron's operation and formulae are illustrated in Figure 5-2.

The second neuron model [Muthuraman 2003a] used in the project is a more flexible leaky integrator type and is similar to the "Spike Accumulation and delta-Modulation" neurons described by Kuniyosh and Berthouze [Shigematsu 1996] and shown in Figure 5-5. In that paper the authors were investigating the usefulness of their self-organizing neural network architecture for aspects of autonomous robot control. The structure of a single neuron is depicted in Figure 5-5. This neuron has three parameters associated with it: *alpha(α)*, *T* and *P*. All of these parameters are fixed by the evolutionary algorithm.

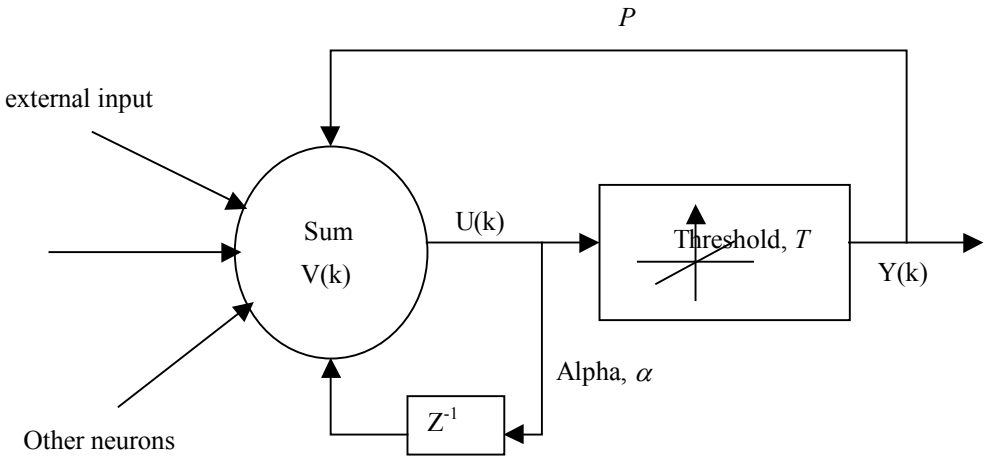


Figure 5-5 Spike Accumulation and Delta-Modulation Neuron Model

Alpha is a feedback factor, which controls the proportion of feedback of the previous internal value into the neuron (in a similar way to K in the previous model). Alpha is a positive constant with a value less than one. Parameter *T* is the threshold and parameter *P* controls how strong the influence of the final output on the internal state is. V (k) represents the internal state of the neuron. A negative value for *P* ensures the resetting of the neuron's internal state *V* after firing a pulse.

The leaky integration of inputs is given by:

$$U(k) = \sum_{j=1}^{J\ nfnt} W_j(k)X_j(k) + \alpha V(k-1)$$

At time (k) the neuron activity U (k) is sum of the inputs multiplied by the weights ($W_1X_1 + W_2X_2 + \ldots$ etc) plus another term ($\alpha * V(k-1)$).

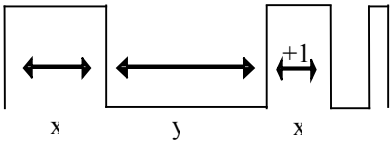
The output Y is given by:

$$Y(k) = G[U(k) - T]$$

where T is the threshold parameter and G[z] is the threshold function: G[z]=1 for z>0, and G[z]=0 otherwise. Finally, once an output pulse Y is produced, the internal state V (k) of the neuron is updated by:

$$V(k) = U(k) - pY(k)$$

Figure 5-6 shows typical output waveform for this neuron model,



Stream of +1's are represented by x
Stream of zeros are represented by y

Figure 5-6 Spike Accumulation & Delta-Modulation Neuron Model Output

The genes used to evolve this model are arranged as shown in Figure 5-7.

Neuron 1			Neuron 1			
Neuron Threshold, T	Feedback factor, P	Feedback factor, α	Neuron Threshold, T	Feedback factor, P	Feedback factor, α	Other Neurons

Chromosome 1

Module 1		Module 2		Module N
Cnnt from Neu: Cnnt to Neu: Cnnt Status: Cnnt Weight:	Cnnt from Neu: Cnnt to Neu: Cnnt Status: Cnnt Weight:	Cnnt from Neu: Cnnt to Neu: Cnnt Status: Cnnt Weight:	Cnnt from Neu: Cnnt to Neu: Cnnt Status: Cnnt Weight:	Other Neurons

Chromosome 1

Figure 5-7 Second Neuron Model Chromosome Parameters

Cnnt = Connection , Neu = Neuron

Two separate population of chromosomes were used to evolve the network. The first set of chromosomes was for the different types of neuron parameters and the second set was for neuron connection status and its weight values. Feedback factor alpha (α) always has a value less than one. The threshold T and parameter P are initialized with a value between –5 and 1. Each connection to/from a neuron will have a value of '0' or '1'. A zero represents no connection and a one represents the presence of a connection. The weight values are initialized in the range –0.5 to +0.5 for presence of a connection, otherwise they were set to zero.

Both the above neuron models have been used in the following chapters for the evolution of modular neural networks. The reasons for having different neuron models will become clearer in the following sections.

5.3 Evolutionary Algorithm

An Evolutionary Strategy (ES) [Schwefel 1995] [Recenberg 1973] was used to evolve the neuron parameters, network topology and connection weights. The ES was chosen because it operates directly on the parameters of the system itself, rather than the genes which lie behind the system. Furthermore, an ES had proven to be successful in previous work [McMinn 2002b].

The topological structure of an ANN has a significant role in its information processing capability. Searching for an optimal topology can be formulated as a search problem in the architecture space. There are several characteristics of such a surface, (as indicated by [Miller 1989]), which make ES-based evolutionary algorithm a good candidate for searching the surface. These characteristics, according to Miller, are:

- There are many of possible connections in the network.
- The surface is complex and noisy since there is no direct mapping between an architecture and its performance (it is based on the evaluation method).
- Surfaces may have similar architectures but quite different performances.
- The surface is multimodal since different architectures may have similar performance.

In this research, the evolution of both network topology and connection weights for an ANN were done at the same time, as shown in Figure 5-7. Combining two levels of abstraction into one increases the search space. Suppose the size of the topological space is $|S_T|$ and the size of the connection weight space is $|S_W|$, then the size of the two level search space is $|S_W + S_T|$, while the size of the one level search space is $|S_W \times S_T|$[Yao 1993]. The evolution of neuron parameters and network topology connections with its associated connection weights was performed on separate populations to reduce the length of a chromosome as the network grew bigger; so, in general, two separate populations of chromosomes were evaluated.

A $(\mu + \lambda)$ ES was used to evolve the action layer of the ANS, as it was proven the most successful setup in McMinn [McMinn 2002b]. The $(\mu + \lambda)$ version populates the next generation with μ chromosomes from the best of μ parents and λ children from the current generation. The population size was set to 700. At each generation the best 100 chromosomes were chosen to be the parents and breed 600 offspring, giving a ratio for μ:λ of 1:7. In several experiments, different numbers of parents and offspring were used, but the ratio was maintained. Each chromosome in the population was evaluated based on a fitness function described later in this chapter. These chromosomes were then sorted into descending order. Crossover was used to create offspring from two parent chromosomes, randomly selected from the elite section of the population. The probability of the best parent chromosomes being selected to reproduce offspring was set to 0.85. The mutation probability for each gene was set to 0.25 (meaning that for each offspring created, on average each gene stands a 25% chance of being mutated). Genes were mutated by adding or subtracting a small value returned from a Gaussian random number function with mean value of zero and standard deviation of 0.05.

5.4 Actuator Models

One of the primary tasks of the research was to evolve an ANN to generate patterns of activity (the lower layers of the ANS) for bipedal and quadrupedal locomotion of a simulated robot. Therefore, an actuator model was required to test the output produced by the network. The robot leg model based on an actual robot and shown to be accurate in [McMinn 2002a] was reused to generate bipedal locomotion walking

patterns. For each leg, there is one active degree of freedom (the hip) and one passive degree of freedom (the knee). The knee can only bend forward and locks when bent backwards. These models have been shown to work well, as the neural networks simulated using them display the same behavior on physical robots [McMinn 2002b]. The models have been used to produce results in several papers using different systems [Shigematsu 1996], [McMinn 2002b] and the results were checked from time to time on the physical robots on which they were based to ensure their compliance. In the case reported here, the investigation started with the robotic equivalent of a Mudskipper. This means that the robot can drag itself about using two front legs that have one active and one passive degree of freedom type. The simulated robot leg is shown in Figure 5-8.

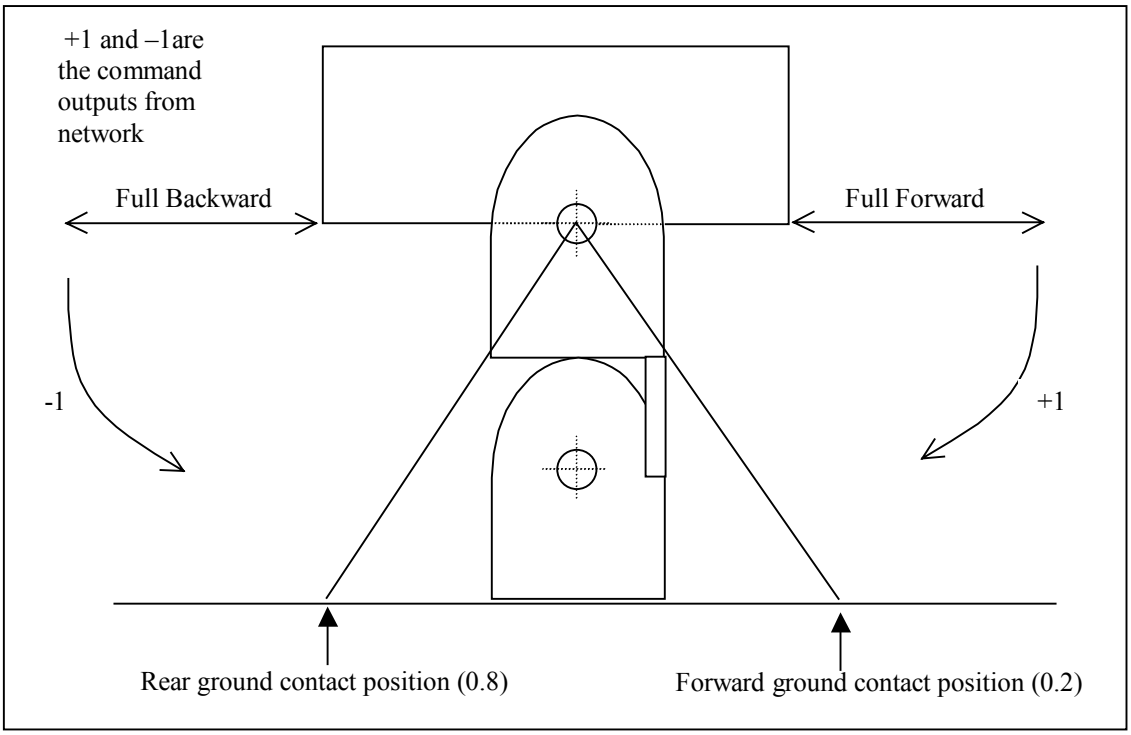


Figure 5-8 Simulated Single Robot's Leg

An output value of 1 from the network will force the leg to move back and a value of –1 will move the leg forward, as shown in Figure 5-9. The leg is in contact with the ground and the knee is locked between positions 0.2 and 0.8, as shown in Figure 5-8. 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 robot was only allowed to move

when the legs were moving in opposite directions from an initial position for a fixed period of time. This is loosely similar to a human walking gait.

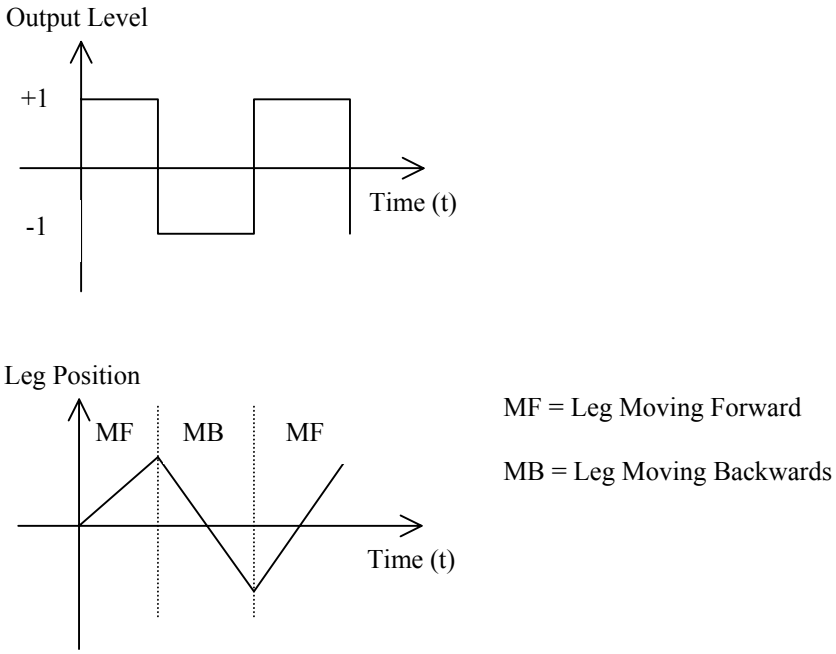


Figure 5-9 Neuron output and actuator leg position

The fitness function used to evaluate the performance of the CPG is the distance over which the simulated robot moves. The simulation time for all the experiments is set to 500 time steps. There are 50 positions between the fully forward and fully backward point. The leg can move one position in one timestep. The leg is at position 10 (0.2×50) when the knee is at the forward ground contact point and at position 40 (0.8×50) when the knee is at the rear ground contact point. The fitness function is a counter which clocks up the steps taken as the leg move backwards. For example when the leg is moved from forward contact point to the rear contact point, the robot has propelled itself forward 30 (40-10 = 30) steps.

In the other actuator configuration used, both degrees of freedom are active [Muthuraman 2003b]. This actuator model is illustrated in Figure 5-10.

In the first configuration, shown in Figure 5-10 (b), the first degree of freedom corresponds to a hip joint which can move in the horizontal plane through about 180 degrees. The first joint is allowed to move 90 degrees backward and forward from the mid positions, which gives a full range of 180 degrees.

When the leg is moving forward from the rear, it has to lift the second degree of freedom until the forward position is reached and then place it on the ground..

The second degree of freedom allows the leg to move 45 degrees up or down from its horizontal position. This movement is controlled by the same motor mechanism described above. This type of configuration is loosely analogous to insect leg movements.

Although these leg arrangements appear different, networks evolved for the one active degree of freedom arrangement can be used as the basis for the two active degree of freedom system because the horizontal leg joint corresponds to the "power stroke" in the simpler system and has a corresponding angular movement.

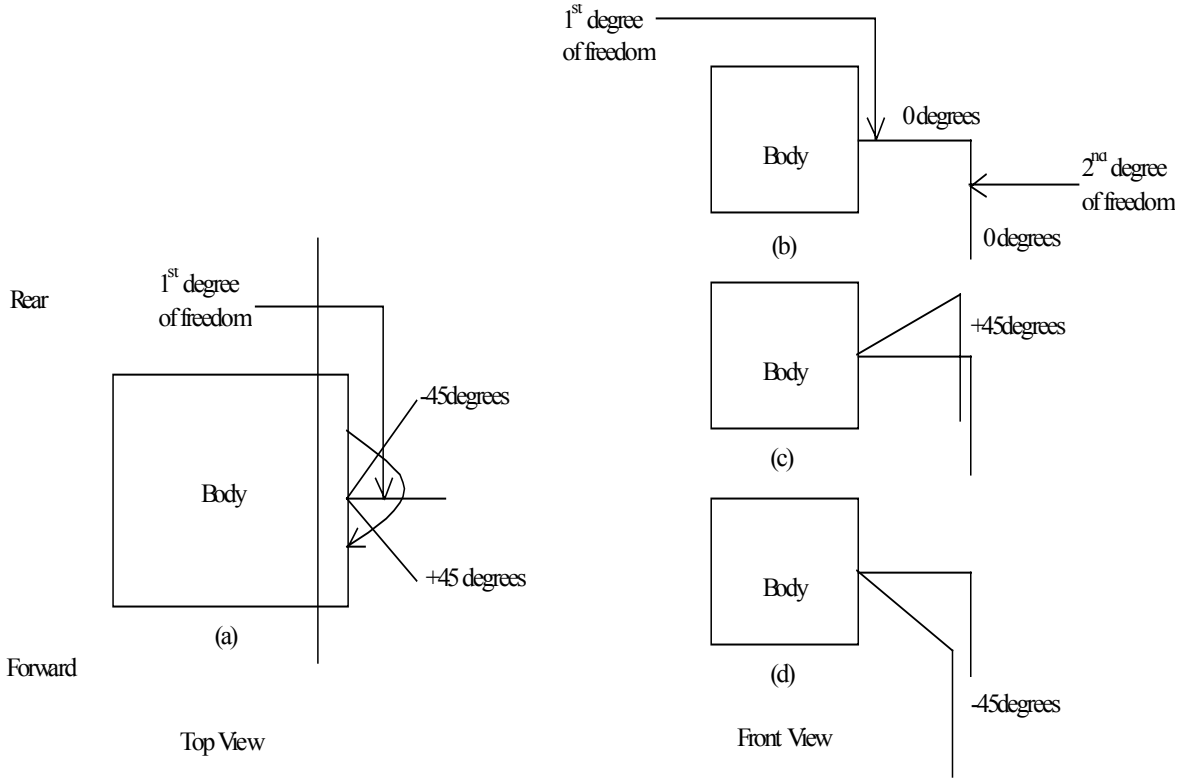


Figure 5-10 Leg model with 2 degree of mechanical freedom

In this type of actuator there are 180 positions between the fully forward and fully backward point. The robot's leg has to move from the forward to rear position within the desired range. Two different ranges have been used in experiments. The first range is between forward ground contact point 0.75 and rear ground contact point 0.25. This means the leg has free movement between positions 135 (0.75×180) and 45

(0.25×180). In this case, the robot's leg makes a full stride from forward to the rear position and the robot has moved 90 steps (135-45 = 90) forward. The second range is between the forward ground contact point, 0.65 and rear the ground contact point 0.35. Figure 5-11 illustrates the range for the leg movement.

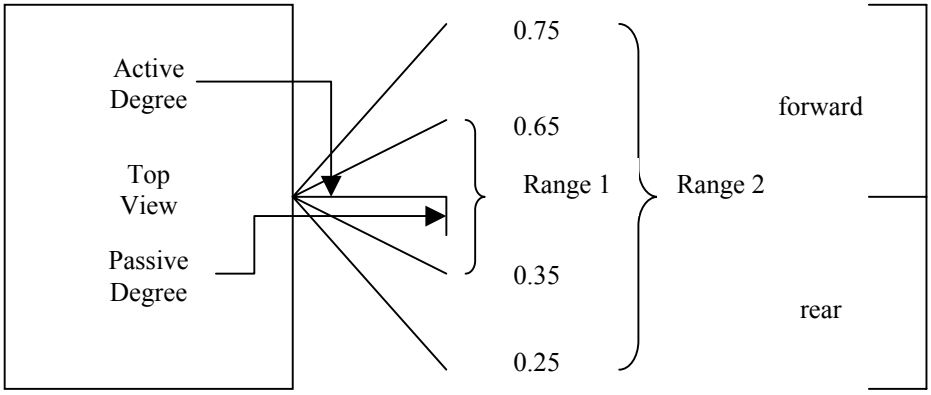


Figure 5-11 Robot's leg movement range

Range 1 was used in the first configuration for the leg with one active and one passive degree of freedom and Range 2 for leg with two active degrees of freedom.

5.5 Robot Development Morphology

As the network grows, an appropriate evolutionary path must be chosen to allow the system to develop from a simple form to a complex one [Muthuraman 2003b]. In this research, the study started with a very simple robot - the robotic equivalent of a Mudskipper. This means that the robot can drag itself about using two front legs of the one active, one passive degree of freedom type. Next the system was deconstrained so that the legs were of the two active degrees of freedom type. The system moved from this bipedal situation to a stable quadrupedal body form. Figure 5-12 shows the general progression. These stages will be discussed in detail accompanied with results in next few chapters.

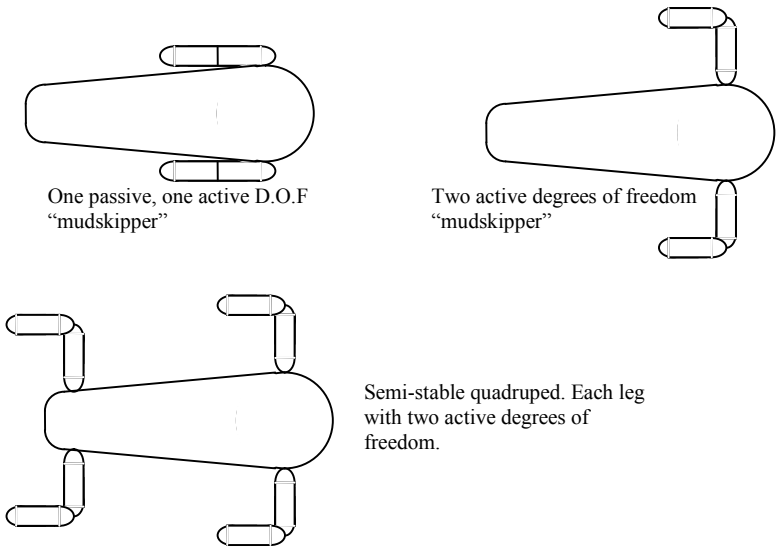


Figure 5-12 Robotic body development

It should be noted that, although a predefined body plan has been used in this example, it would also be possible to allow an evolutionary algorithm to choose the body plan form (for example, from pre-arranged building blocks) as part of the algorithm [Sim 1994].

5.6 The Principle of the Artificial Evolutionary System

The basis of the research reported here is the application of the biological principles outlined in the previous sections to an artificial system.

The technique used has its origins in the paper "Evolution and Devolved Action" by MacLeod [MacLeod 2002] (included in Appendix A of this thesis). As outlined earlier, this paper discusses several different methods for evolving networks. These methods were subsequently refined in later papers [McMinn 2002b], [Muthuraman 2003a] into the system adopted here.

For ease of comparison with previous work, the technique is demonstrated using a legged robot but, as discussed later, the general principles are applicable to many other systems.

The neural network evolution is illustrated in Figure 5-13 and proceeds as follows:

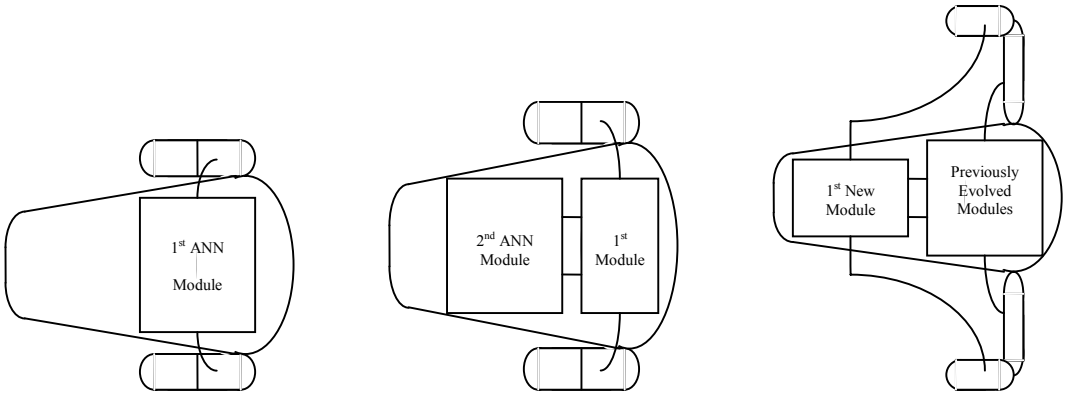


Figure 5-13 Evolution of robotic body plan

1. Initially the robot's body plan is made as simple as is practically possible.

2. Next, a Neural Network Module is added to the robot's control system. This network is trained until its fitness does not increase further. The trained weights of this network are then fixed and do not change as further networks are added.

3. If the system has not reached its maximum possible fitness, then a new module is added on top of the previous network and its weights are trained (again, after training, these weights are fixed).

4. The process outlined in point three above is repeated until the fitness (the robot's performance) has reached its maximum possible level with the robot's current configuration (or, if maximum fitness information is not available, until fitness does not increase with the addition of subsequent modules).

5. Once the evolved network has reached its maximum fitness, with its current configuration, either the body plan or the environment of the robot is allowed to become slightly more complex - in the terminology used here, it is *deconstrained*.

6. The algorithm then repeats this whole process using the networks developed in the previous iteration as a fixed basis to build on. By adding new modules on top of old it builds up the network, one part at a time, until the maximum fitness with that body / environment configuration is reached; the robot is then deconstrained again and so on.

The central point is that, at each stage within this process, new networks build upon older structures from previous iterations and only the weights of the new modules are trained.

5.7 Implementation of the Evolutionary System Technique

The description of the software used in the project in implementing the evolutionary ANN technique to generate results presented throughout Chapter 6 to 8 is presented in Appendix E.

Chapter 6

Initial Results

6.1 Introduction

In this chapter, the initial results obtained from simulating the Direct Growth Method are presented and discussed. Results are presented showing the technique in operation with a simple body form.

6.2 Results from Single Functions

The first problem investigated was the evolution of a Central Pattern Generator (CPG) which could produce the basic gait patterns for bipedal locomotion using the one passive, one active degree of freedom leg with the most basic (mudskipper) body form. Firstly, the MMM neuron model described in Section 5.2 of Chapter 5 was used to implement the CPG. In this case the actuator model is slightly modified so that the leg joint is forced to move up to the knee lock reset point from the forward ground contact point before the robot propels its body forward on the next stride as shown in Figure 6-1.

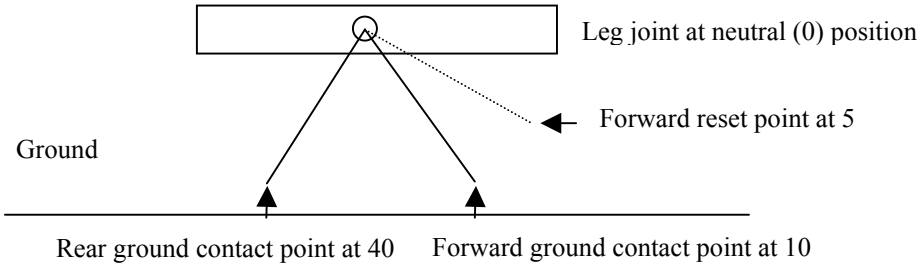
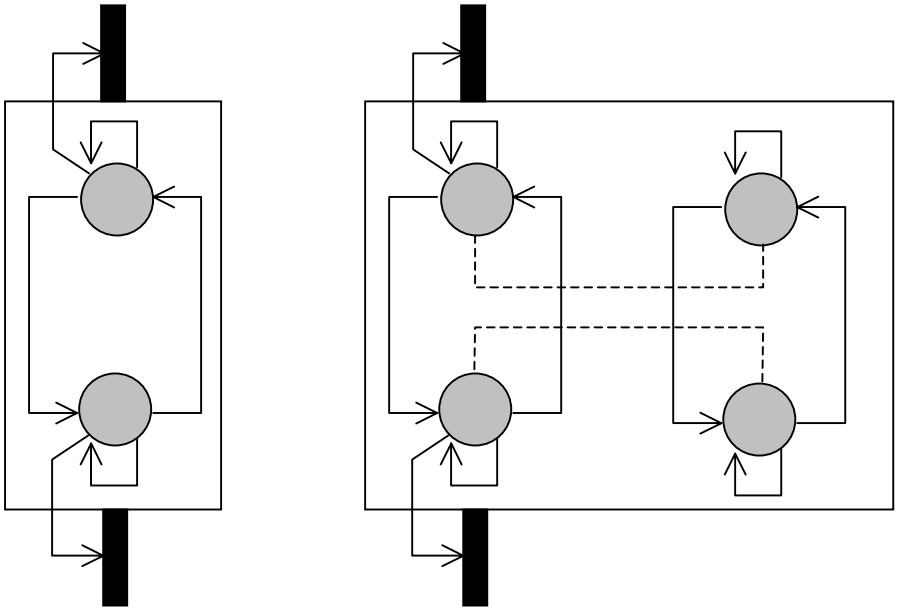


Figure 6-1 Modified actuator model

The initial number of neurons in the CPG was set at two because there were two actuators present, each of which must be connected to a neuron. The simulated robot was stable in all directions because it was only the production of the appropriate gait patterns that was under investigation. The fitness score for each chromosome was how far the robot moved from its initial position within 500 time steps (therefore, higher scores were better). Two different modules (firstly, with one neuron and secondly with

two neurons) were added to grow the network, while preserving the neuron parameters and inter-neuron connection weights in the previous modules. All the modules were fully connected. The configuration and growth of the network with two initial neurons proceeded as shown in Figure 6-2. Solid lines show possible connections. The modules were added until the fitness reached its maximum value, and increasing the number of modules thereafter made no difference to the fitness.



(a) First module. (b) Second module added.

Figure 6-2 Growth scheme for single degree of freedom. (a) First module placed and ready to train (b) First module fully trained; second module placed and ready to train

Figure 6-3 shows the resulting robot leg positions, when modules with a single neuron were added to an initial module containing 2 neurons. The best pattern (highest fitness) is when both the legs fluctuate between position 5 and 40, out of phase and the pattern repeats in this range, to give a maximum distance of 430. This corresponds to 14.25 complete strides within the simulation time.

Studying the graph (Figure 6-3 a)), one can see that the left leg is in phase with the right leg at the beginning of the oscillation and the gait pattern stabilizes after this. There were no oscillations in the position (between position 5 and 40) of the robot legs in the beginning when the network size is small but the oscillation becomes clearer in the latter part of the experiment, Figure 6-3(c). The distance moved by the robot with

two neurons in the first module is 341 steps ((d)). The distance remained the same after the second module is added.

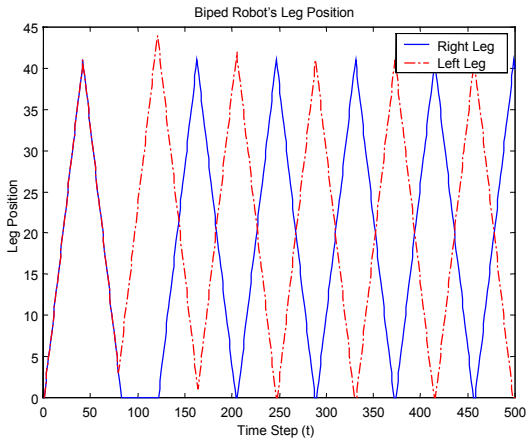
Let us consider the operation of the network as more modules are added while freezing the neuron parameters and connection weights of the previous modules. When a new module is added, there are many possible connections between neurons. In this case, for example, when a second module of one neuron is added to an initial module of 2 neurons there are 5 possible connections (including the recurrent connection to itself). More connections are possible as the number of neurons is increased in the module or the number of modules. The solution search space expands as number of connections increases. The larger the search, space the more difficult it becomes for the ES to find a good solution. One of the probable reasons for no increase in fitness is that there were not enough neurons in the new module to influence the previous modules.

After the third module is added the distance increased to 373, an increase of 32 steps. This increase may not be possible without the presence of the second module. The distance remained the same for the next three modules. When the sixth module is added the distance increased by 4 steps and remained the same thereafter with increasing number of modules of one neuron. There is no large increment in distance moved after the third module.

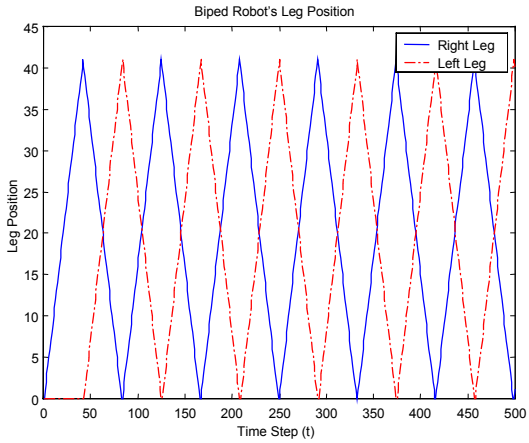
When there is no increment in distance after a new module is added, the previous modules can be said to have reached a stable structure. Most probably, more neurons are required in the new module to modify the initial behavior of the stable structure. In this case, one neuron in a module is not adequate to give a great improvement.

It also can be seen from Figure 6-3(a) to (c) that the leg oscillates between positions 0 and 40, which are not within the desired range. The leg always goes to the 0^{th} position, Figure 6-3(a) - (c), from the rear ground contact point. This means that the distance count loses 5 steps when the leg moves from the rear to the forward position. From Figure 6-3(c), on average there are 12 complete strides between leg position 0 and 40. Therefore the total number of steps was 60 less than the maximum possible. The distance moved by the robot in Figure 6-3(d), increases with increasing number of modules. The maximum distance moved with six modules is 377 steps.

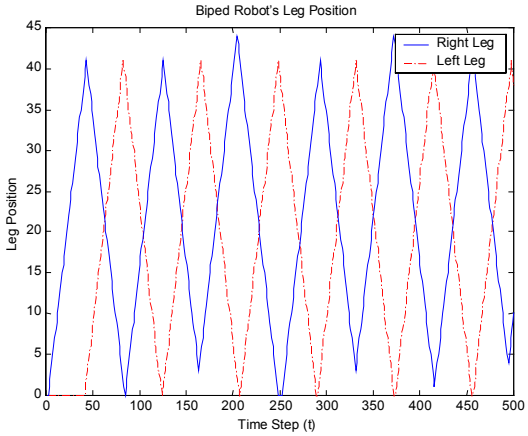
Note: x:y:z where x,y,z… refers to number of neurons in a module



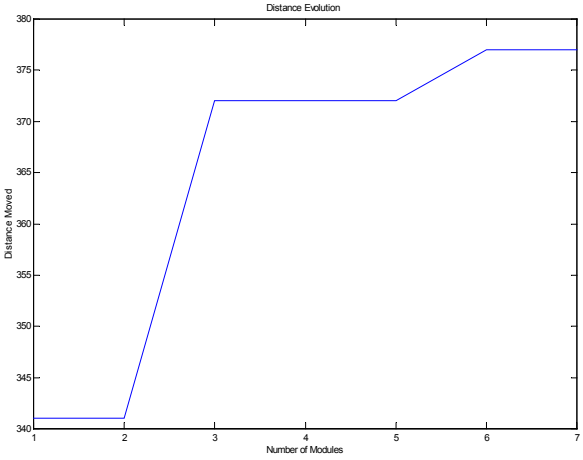
a) 1 module, 2



b) 3 modules, 2:1:1



c) 6 modules, 2:1:1:1:1:1



d) Distance moved (fitness) with increasing number of modules. Output to the actuator taken from the neurons in the 1st module i.e. neuron 1 & 2

Figure 6-3 Leg positions of a bipedal robot and the improvement of fitness when modules with single neuron were added to the previous modules

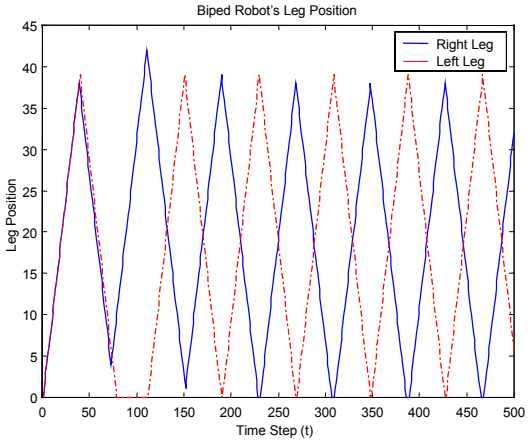
Figure 6-4 shows the leg positions of the robot and the distance moved when modules with 2 neurons were added to the system. In both the legs started to oscillate between position 5 and 45 when there were 4 neurons in total. This behaviour does not occur when modules of one neuron are added to the existing network. The oscillations continue to increase as the number of modules increases. This improves the distance moved by the robot.

The robot moved 358 steps with 2 neurons in the initial module. The rate of change of steps when the second module was introduced was 31. The distance moved increased to 389 steps. The rate of change decreased to 5 and 2 for the third and fourth module. Further changes remains constant at 2. The maximum distance moved was 396.

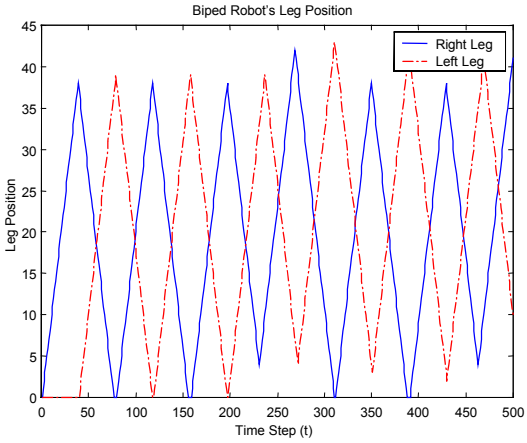
It can be seen from Figure 6-4 (e), that the distance moved increases with an increasing number of modules, but it is still not possible to reach the theoretical maximum distance. The distance moved increases by 22 steps when the network is grown with a module with 2 neurons compared to when the network is grown with a single neuron module. A good solution was still not achievable by growing the network with 2 neurons in a module.

Even though having 2 neurons or more in the new module may provide more connections, neuron functionality also seems to have an important role in determining the growth of the network. In the MMM neuron model the timing parameters, t_1 and t_2 of the neurons are fixed; there is no flexibility to modulate this information. The addition of new modules only provides the required phase shift for a particular gait, in this case bipedal locomotion. This shows that the timing information of the neuron is very important.

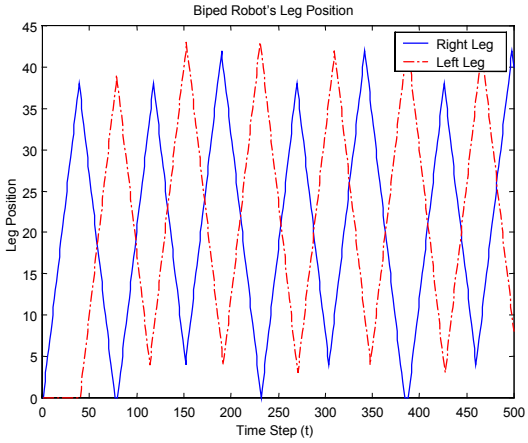
Note: x:y:z where x,y,z… refers to number of neurons in a module



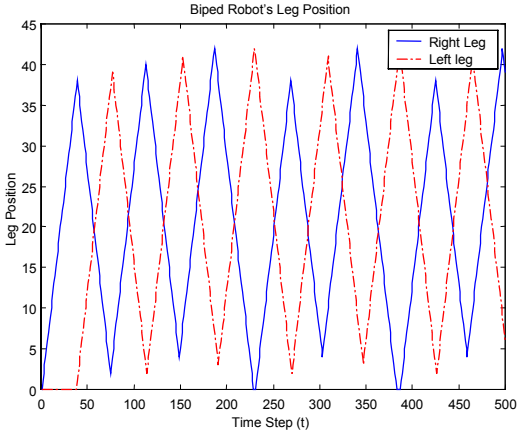
a) 1 module, 2



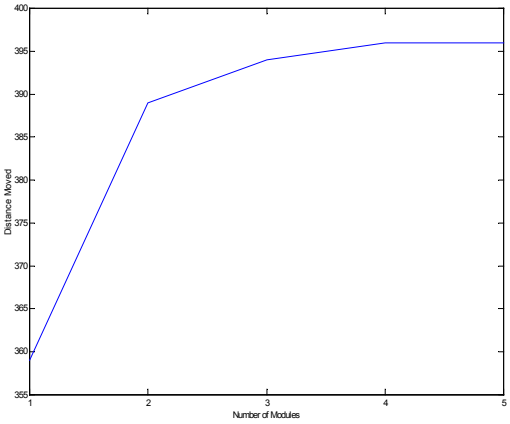
b) 2 modules, 2:2



c) 3 modules, 2:2:2

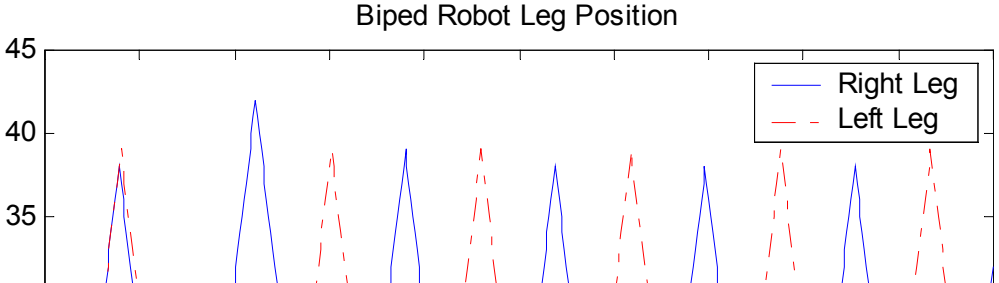


d) 4 modules, 2:2:2:2

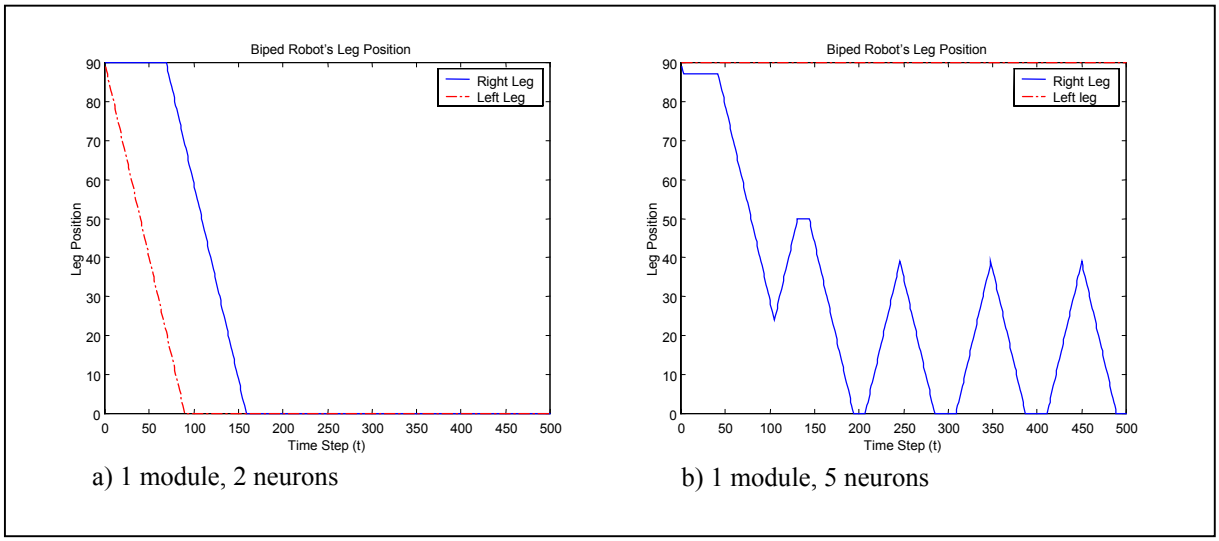


e) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the 1st module i.e. neuron 1 & 2.

Figure 6-4 Leg positions of the robot and the distance evolution when modules with 2 neurons were added to the previous modules



In the next experiments, the MMM neuron model described in Section 5.2 was used to implement the lower layer of the ANS and was tested on an actuator with 2 degrees of freedom as shown in Figure 5-10. However, when this was implemented, it was found that the network failed to evolve to a solution, which moved any distance. The result in Figure 6-5 below shows the robot's leg positions when 2 and 5 neurons are used in the initial module. The left leg position with five neurons is at position 90; therefore it is not shown clearly on the graph.



a) 1 module, 2 neurons

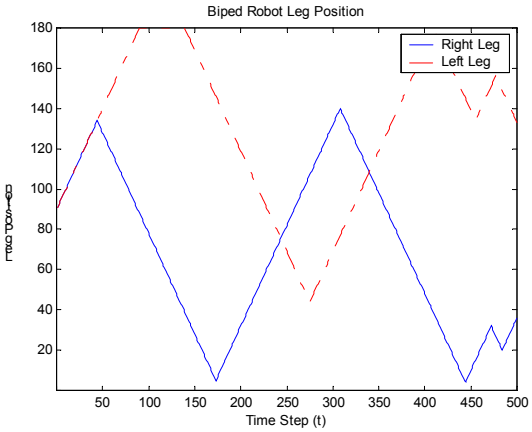
b) 1 module, 5 neurons

Figure 6-5 Robot's Leg Position with New actuator model

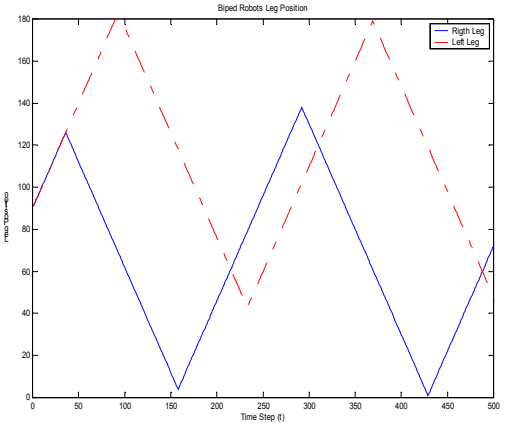
This result meant that the system had to be examined to establish why it was failing. It was discovered that this failure was due to the neuron model used.

The above results (Figure 6-5 (a) and (b)) suggested that the MMM neuron model described in Section 5.2 was not capable of producing the required outputs for bipedal locomotion using the 2 active degree of freedom model actuator. This is because the neuron model has a fixed on (t_1) and off (t_2) time; this causes the neuron to fire for the time fixed by the evolutionary algorithm. The neuron does not therefore reduce or increase its firing rate in response to influences from other neurons. Moreover, in further experiments (below), it was found that influence from other neurons is very important.

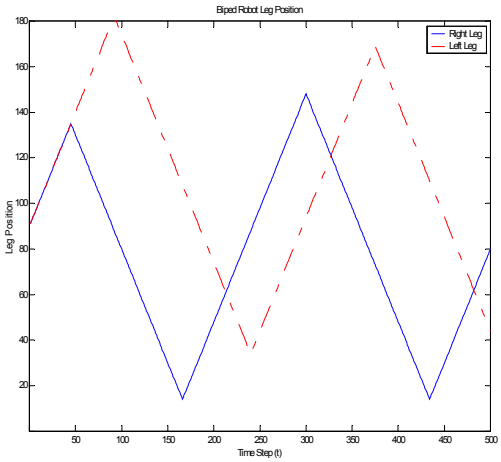
Note: z:y:z where x,y,z… refers to number of neurons in a module



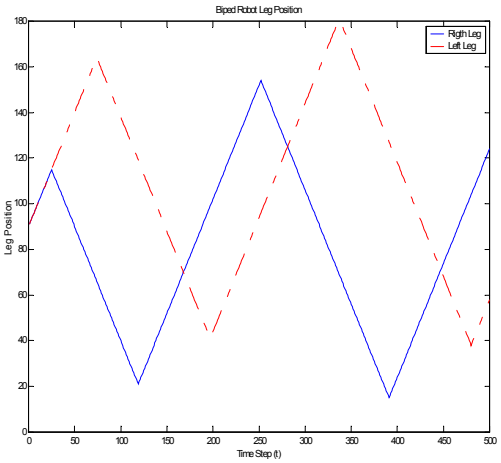
a) 1 module with 2 neurons



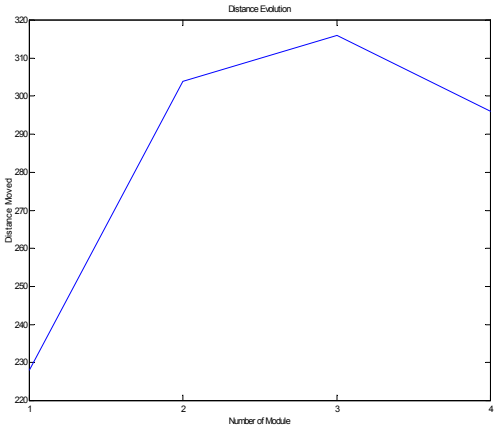
b) 2 modules, 2:1



c) 3 modules, 2:1:1



d) 4 modules, 2:1:1:1



e) Distance moved with increasing number of modules. Output to the actuator taken from the neurons in the 1st module i.e. neuron 1 & 2.

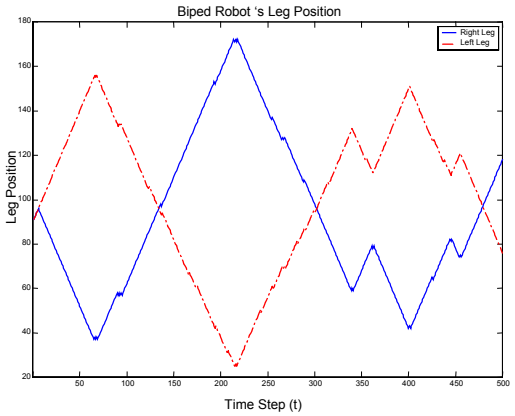
Figure 6-6 Leg positions of the robot when modules with 1 neuron were added to the previous modules

Figure 6-6 shows the results of using the new neuron model (Spike Accumulation and Delta-Modulation) described in Section 5.2 to evolve a bipedal gait when one neuron is added to the existing modules for the actuator model shown in Figure 5-10 of Chapter 5.

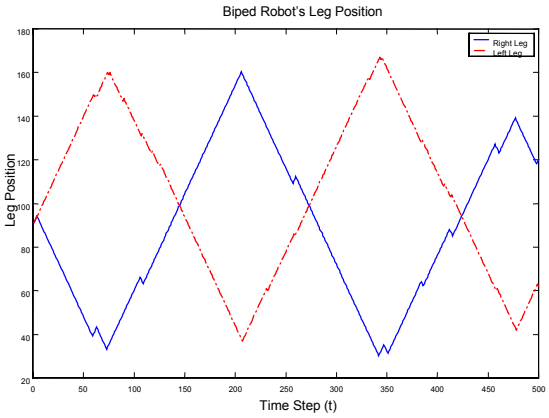
From Figure 6-6(e), the distance moved by the robot increases with increasing number of modules with one neuron. The leg position (Figure 6-6 (a-d)), oscillates between position 0 and 180 without reaching zero like the previous neuron model (Figure 6-3 and Figure 6-4 does with the first actuator model in Section 5.2. This shows that this new neuron model is capable of controlling biped locomotion with these actuators. The distance moved decreases further when a fourth single-neuron module is introduced. There are three possible reasons for the decrement in the distance moved. The first is the inability of the neuron model itself to modulate the firing activity. Secondly, the connection pattern between neurons (within and between newly added modules) is incorrect; in all the experiments described so far, all the neurons in the network were fully connected. Thirdly, when a new module was added to the network, the ES was not able to evolve the best connection weights to increase the distance moved by the robot. Inconsistent activity in the network can cause the decrement in the distance.

Figure 6-7 shows the robot's leg positions when two neurons are added to the existing modules. From (e), the distance moved by the robot increases for the first two added modules and then decreases for the latter two modules. The robot's leg position is much improved compared with the single neuron module results. This shows that the number of neurons in a module is very important. Later experiments will give more insight into this point. From Figure 6-6 and Figure 6-7, it may be noticed that the fitness increases quickly at the beginning and then starts decreasing when more modules were introduced.

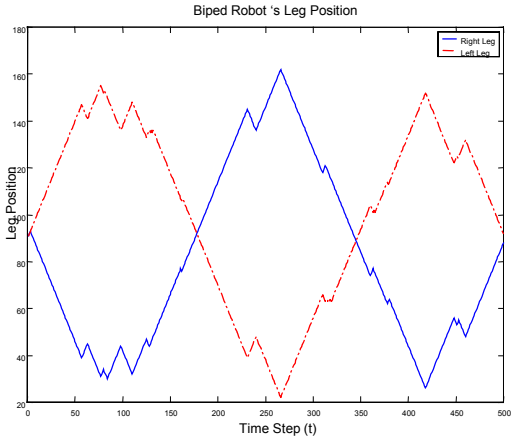
Note: x:y:z where x,y,z… refers to number of neurons in a module



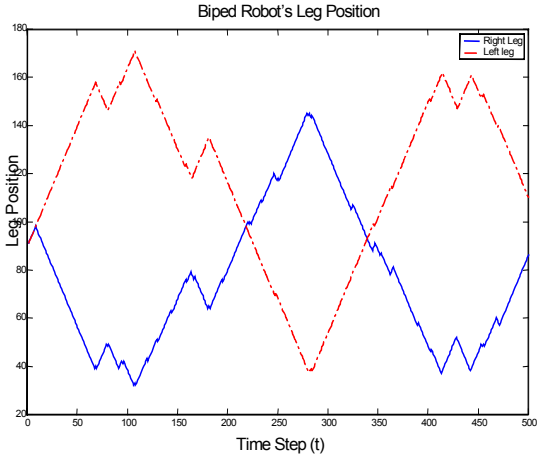
a) 1 module with 2 neurons



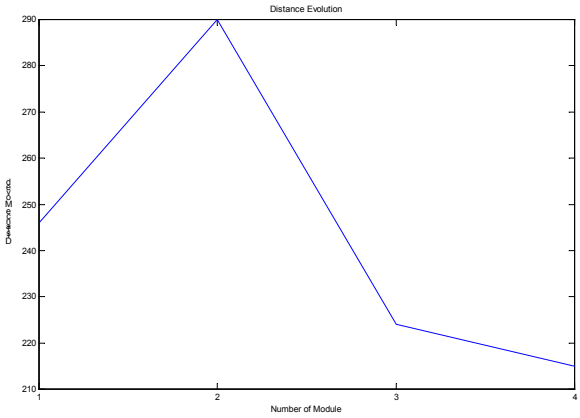
b) 2 modules, 2:2



c) 3 modules, 2:2:2



d) 4 modules, 2:2:2:2



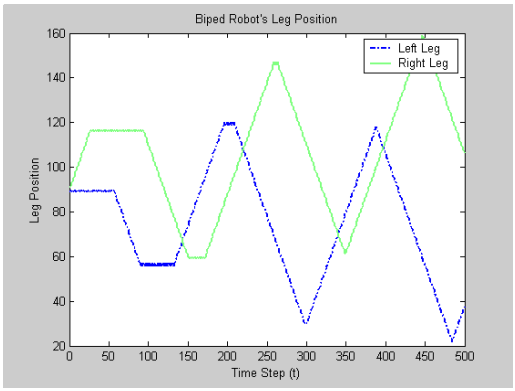
e) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the 1st module i.e. neuron 1 & 2.

Figure 6-7 Leg positions of the robot when modules with 2 neurons was added to the previous modules

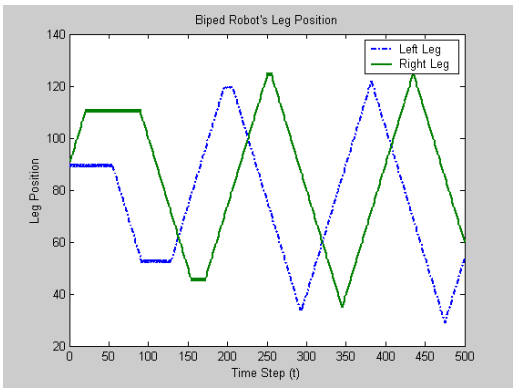
During these tests, a second important discovery was made (the first being the importance of the neural functionality outlined above). This was that allowing all connections to be present - that is, allowing a fully connected network - caused the evolution to either slow down or stop completely. This problem was resolved by allowing the Evolutionary Algorithm to choose the connections within the network as well as their weights. The reason that the connection pattern is important may be that a fully interconnected pattern means that all neurons in the previous module are affected by the new module. While some of these connections cause improvements in fitness, this may be counteracted by other connections which cause a decrease. Although it could be argued that unused connection weights will evolve to zero anyway, it was found that evolution proceeds much more quickly by simply allowing the deletion of connections.

The initial experiments with this approach involved adding a module with one neuron to the previous modules. Figure 6-8 shows the leg positions of the robot for this configuration. The robot managed to move a distance of 261 steps with 2 neurons in the initial module. The distance increased with increasing number of modules and saturated at 310 after the fourth module. The growth strategy of adding a module with one neuron could not evolve fully towards the best solution.

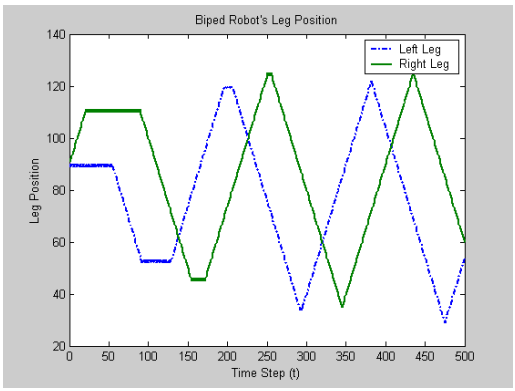
Note: x:y:z where x,y,z… refers to number of neurons in a module



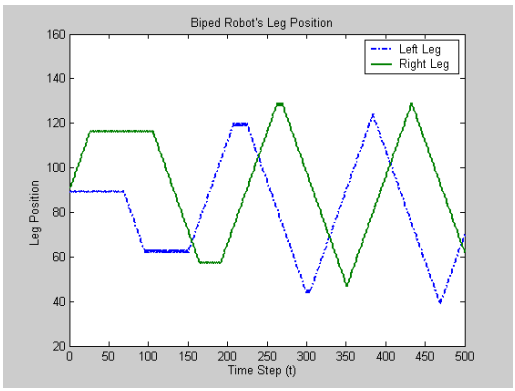
a) 1 module with 2 neurons



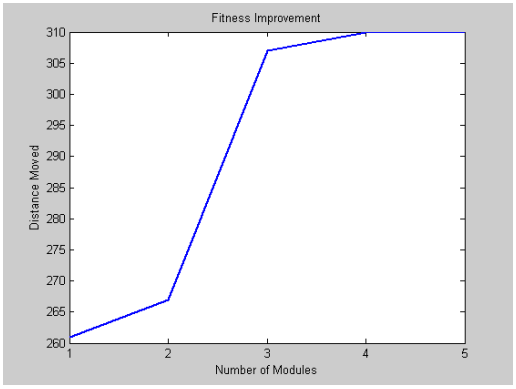
b) 2 modules, 2:1



c) 3 modules, 2:1:1



d) 4 modules, 2:1:1:1



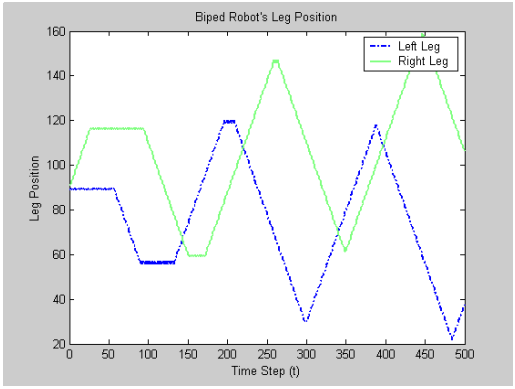
e) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the 1st module i.e. neuron 1 & 2.

Figure 6-8 Leg position of the robot when modules of one neuron were added to the network with connections evolved by the ES

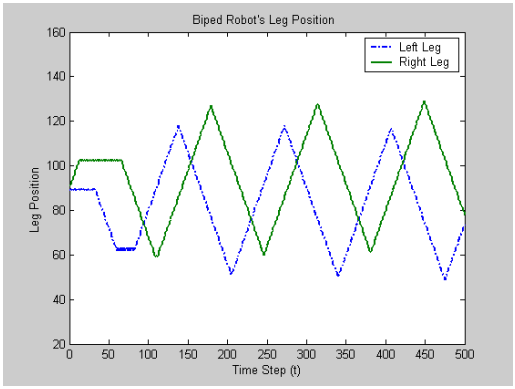
The first module used previously to illustrate the growth in adding a module with one neuron was used again in this experiment. Figure 6-9 shows the leg positions when a module with 2 neurons was added to the previous modules. There were an improvement 89 of steps in distance when the second module was added. The distance continued to increase with an increasing number of modules. The maximum distance moved was 420 steps with six modules. The distance saturated and remained at 420 with increasing number of modules thereafter. There were 12 (six modules of two neurons) neurons in total. Adding 2 neurons in a module showed a great improvement in the results compared to adding a module with one neuron but maximum distance still could not be reached.

A conclusion that can be drawn by analyzing all the results from the previous experiments is that there should be a minimum number of neurons in the new module for it to have a maximum potential for incremental growth towards the best solution. The number of neurons required depends on the mapping difficulties that the new module has to overcome to reach the solution.

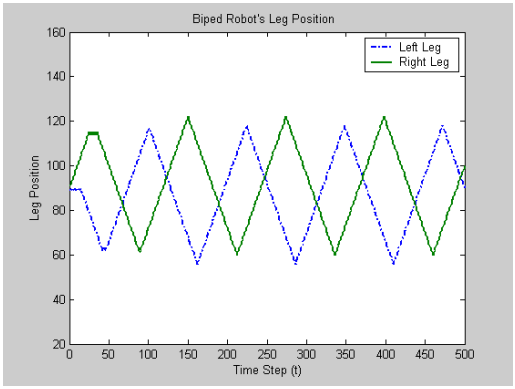
Note: x:y:z where x,y,z… refers to number of neurons in a module



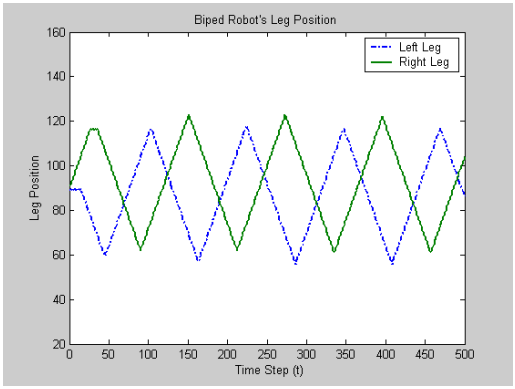
a) 1 module with 2 neurons



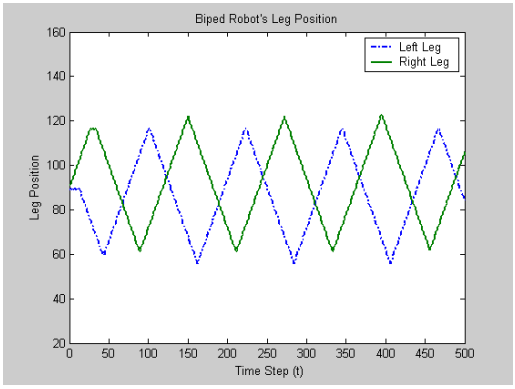
b) 2 modules, 2:2



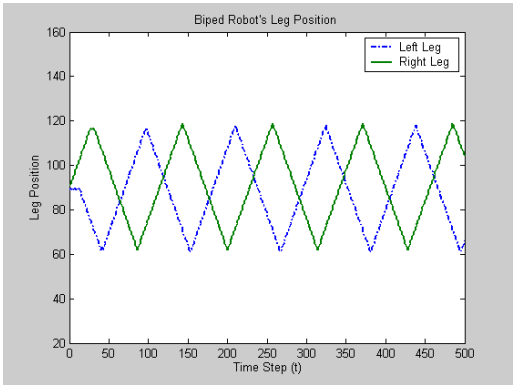
c) 3 modules, 2:2:2



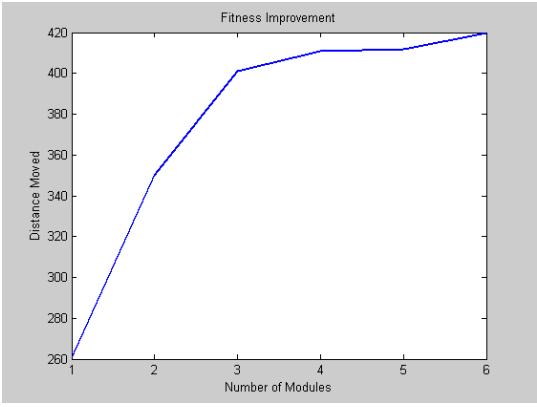
d) 4 modules, 2:2:2:2



e) 5 modules, 2:2:2:2:2



f) 6 modules, 2:2:2:2:2:2

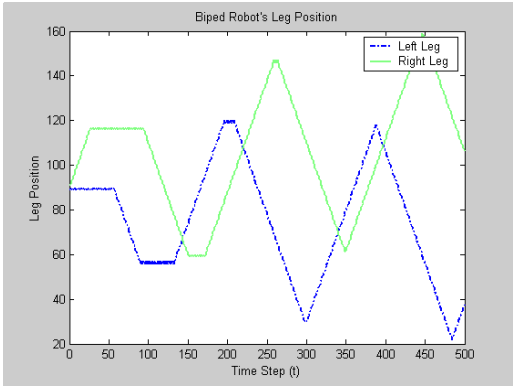


e) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the 1st module i.e. neuron 1 & 2.

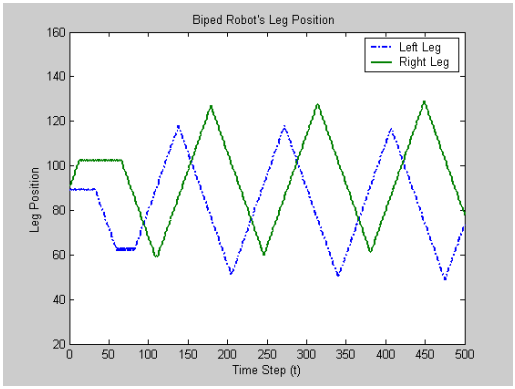
Figure 6-9 Leg position of the robot when modules of two neurons were added to the network with connections evolved by the ES

Figure 6-10 illustrates the distance travelled with different numbers of neurons in the modules. The result was promising, and the distance moved and the leg patterns improved as number of modules increased. A module with two neurons was trained. The robot was able to move a maximum distance of 261 in 500 time steps - see Figure 6-10 (a). Then, a module with two neurons was added. The distance moved increased to 350 – see Figure 6-10 (b). Finally, a module with three neurons was added and the distance increased to 440 – see Figure 6-10 (c). The distance moved never changed thereafter, with an increasing number of neurons and modules. Figure 6-10 (d) shows the fitness improvement as modules are added to the network. The total number of neurons to reach the maximum distance for a bipedal locomotion is 7. Figure 6-11 shows the neuron connections between neurons for all three modules.

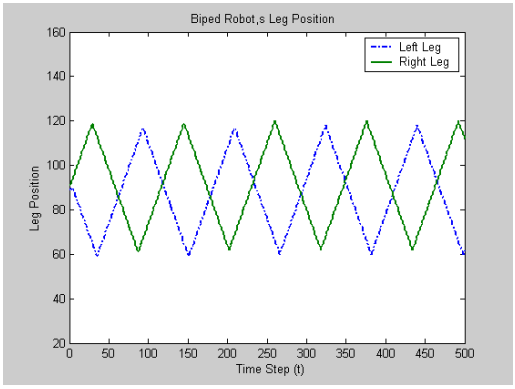
Note: x:y:z where x,y,z… refers to number of neurons in a module



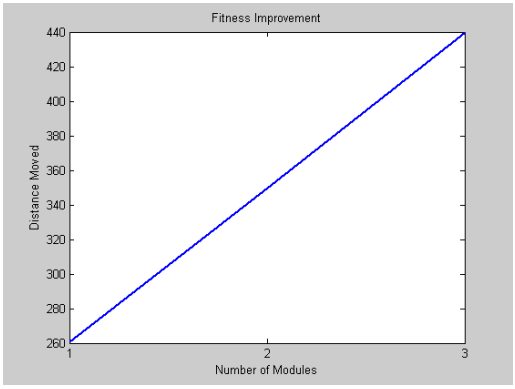
a) 1 module with 2 neurons



b) 2 modules, 2:2



c) 3 modules, 2:2:3



d) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the 1st module i.e. neuron 1 & 2.

Figure 6-10 Leg position of the robot when variable number of neurons were added to the new modules with connections evolved by the ES

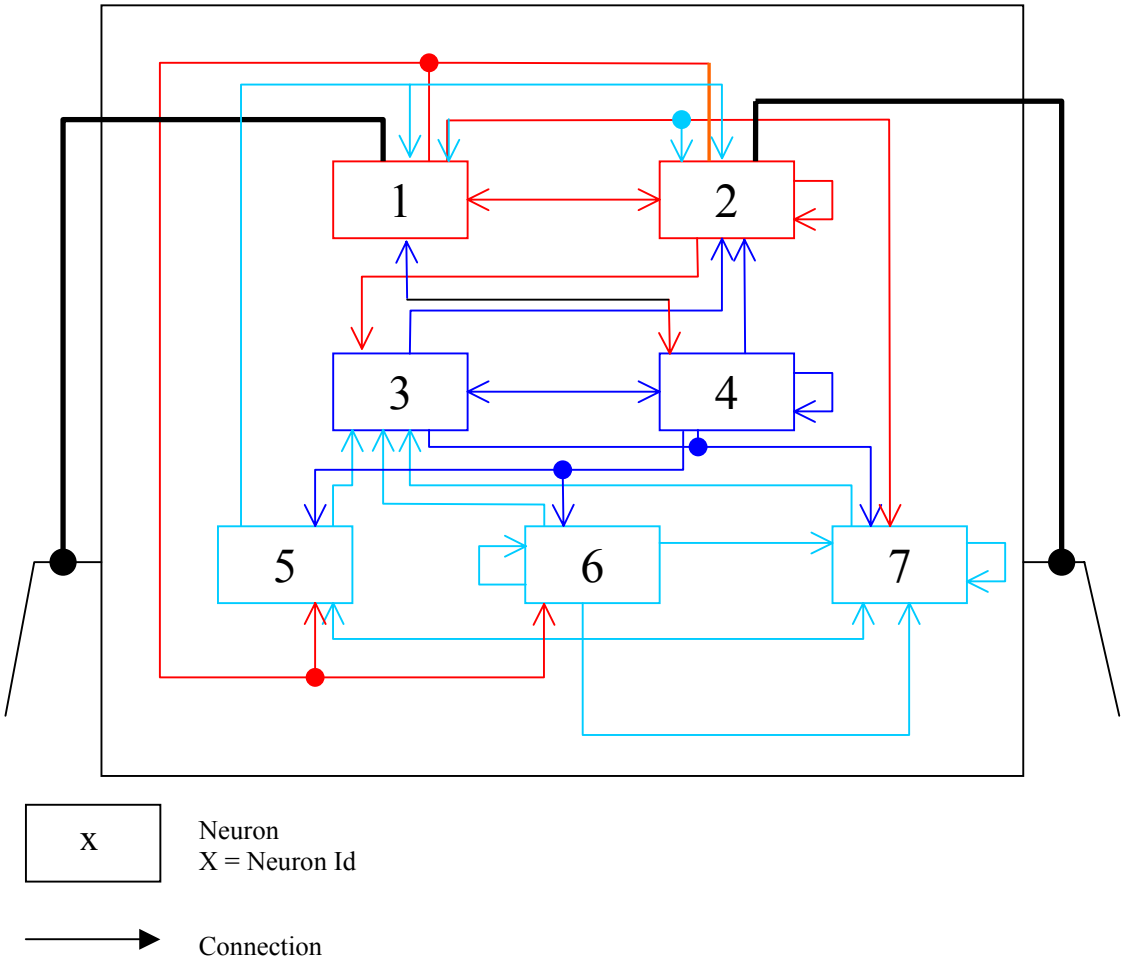


Figure 6-11 Robot's body with neural connections for 1 active 1 passive degree of freedom

All the neurons in the network are assigned with a numerical Identity (Id) in order of addition to the network. Table 1 below shows the number of modules in the network and the neuron identities in that module. Module 2 to 3 are the new modules evolved on top of the previous modules. Module number 1 is the initial output module.

Module Number	Neuron Ids
1	1, 2
2	**3, 4**
3	**5, 6, 7**

Table 1 Module number and neuron Ids

Table 2 shows the evolved connections between neurons when module number 2 and 3 are formed.

Neuron Id	Connection from Neuron Id
1	2, **4, 5, 7**
2	1, 2, **3, 5, 7**
3	*2*, 4, 5, 6, 7
4	*1*,3, 4
5	*1, 2, 4, 7*
6	*1, 2, 4*
7	*1, 2, 3, 4, 5, 6, 7*

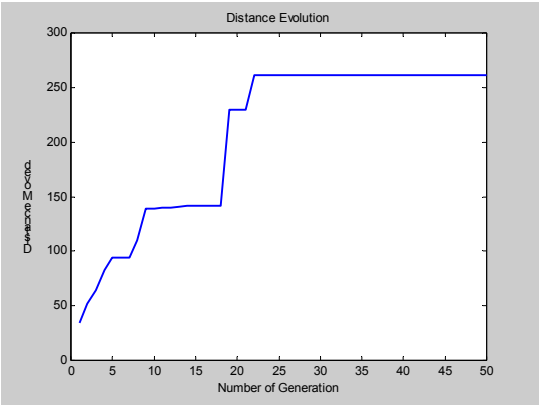
Table 2 Evolved connections to and from neurons in the network

By analyzing the connectivity table (Table 2), we can see that there is at least one connection formed from the new module to the output module, shown in bold. It is also noticeable that fewer connections are formed from the new module to previously evolved modules. From Table 2, more connections are formed from the previous modules to the new module, shown in italics.

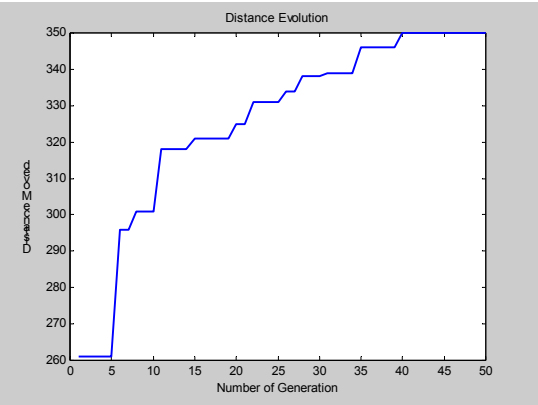
The important point to note is that, if the evolutionary algorithm does not find a good solution, the synapse weights connecting the new module to the previous modules turn out to be zero. From Figure 6-12 (a) the maximum distance reached was 261. When a new module with 2 neurons was introduced, the initial fitness was preserved for few generations before the distance increased further. This showed that the evolutionary algorithm managed to find that the previous modules (having already acquired some degree of knowledge about the problem) were still able to give the maximum distance, even although the new module made the overall system worse.

A network with 12 neurons was trained and the distance moved was 395. There could be 144 (12^2) connections between neurons if all the neurons are fully connected. A simple mathematic calculation will reveal that there are 2.23×10^{43} possible network topologies. Since the ES has to find optimal weights for the connections, this indicates that a big ANN is not always the best solution (because of the large search space). The final solution for a problem might be very small in a large space; incremental growth therefore has an advantage under such circumstances.

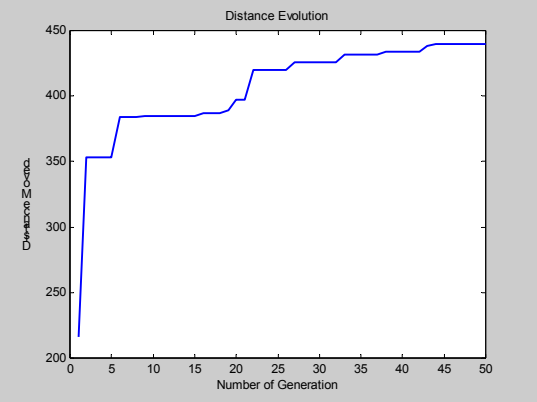
Note: x:y:z where x,y,z… refers to number of neurons in a module



a) 1 module with 2 neurons



b) 2 modules, 2:2



c) 3 modules, 2:2:3

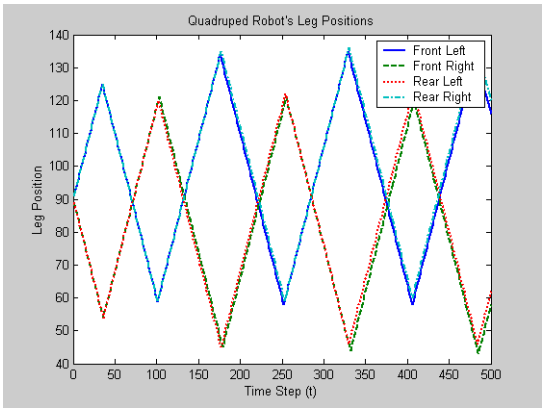
Figure 6-12 The evolution of distance travelled when variable number of neurons were added to the new modules with connections evolved by the ES

6.3 Quadruped

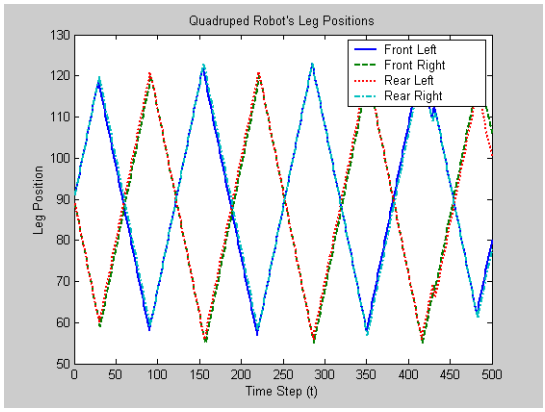
A network to produce a quadruped trot gait based on the actuator model with 2 active degrees of freedom (Figure 6-13) was evolved. The total number of modules required to produce the gait was 6. The modules contained 5, 3, 2, 4, 4, and 5 neurons respectively. In the previous experiment for bipedal locomotion there were two neurons in the initial module. Each neuron in the module is connected to the first active degree of the actuator. There were 5 neurons in the initial module for this experiment. It was found that having 4 neurons in the initial module did not produce the required phase shift between the legs. Irregularities in the leg position can be seen in the first 3 modules (Figure 6-13 a – c). The leg position stabilised within the desired range thereafter. A total of 23 neurons are required to successfully evolve the trot gait

to the maximum distance possible. Figure 6-13 shows the leg positions of the robot and distance evolution as new modules are added to previously evolved modules.

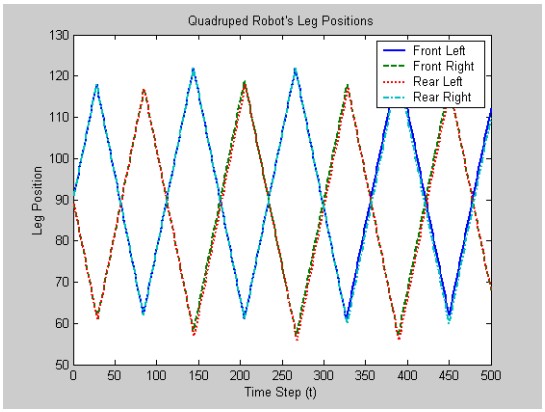
Note: x:y:z where x,y,z… refers to number of neurons in a module



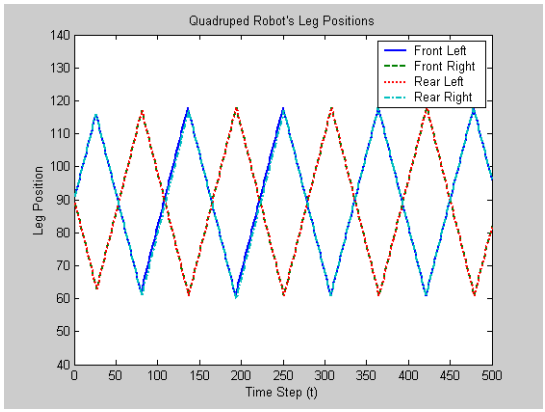
a) 1 module with 5 neurons



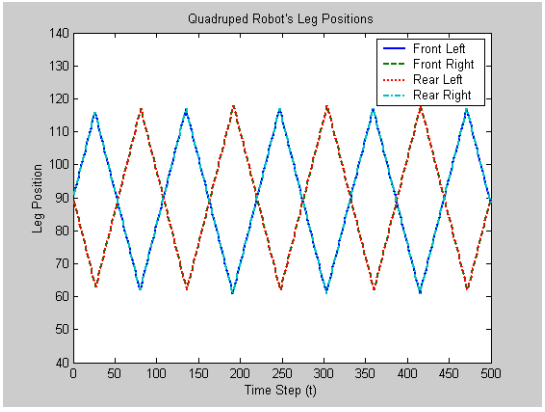
b) 2 modules 5:3



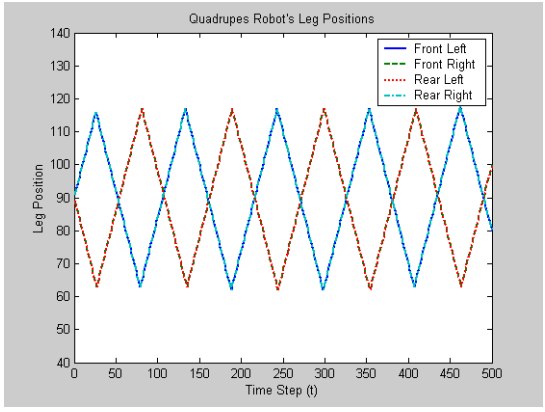
c) 3 modules 5:3:2



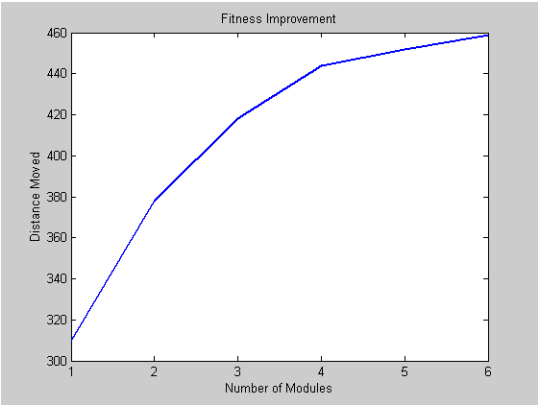
d) 4 modules 5:3:2:4



e) 5 modules 5:3:2:4:4



f) 6 modules 5:3:2:4:4:5



g) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the 1st module i.e. neuron 1 & 2.

Figure 6-13 Quadruped trot gait leg positions

6.4 Permissible Module Connections

Another area addressed in larger networks is that of localising the neural module's connections. At present, the networks used are small enough to allow any neuron to be connected to any other. However, in large networks, this becomes impractical and smaller connection areas (for example only to the previous module layer) may be required. This type of growth could be called uni-directional because modules are only added in front or at the rear of existing modules.

To analyse the effect of permissible connections in a large network, two different experiments were carried out. In the first experiment, modules are only connected to the rear of the last module. Connections are not allowed between other modules (for example connections between the second and the initial module). The outputs are taken from the initial module. This method is illustrated in Figure 6-14.

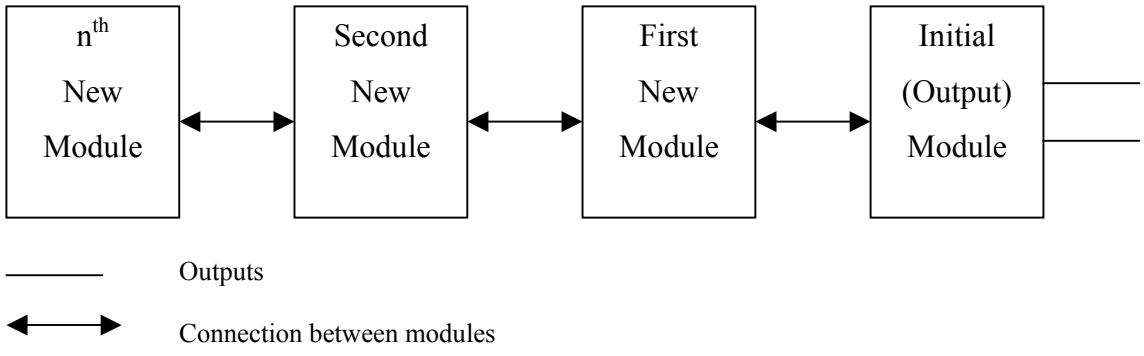


Figure 6-14 Adding modules at the rear of initial module

In the second experiment, modules are added in front of the last module. Again, connections are not allowed between other modules. In this method, the outputs are taken from the newly added module. Any neurons in this module could be selected to be the output neuron. The disadvantage of this method is that there will always be a minimum number of neurons in the module. The number of neurons is determined by the number of actuators. For example, a minimum of 4 neurons are always required in the new module to control a quadruped robot with a single degree of freedom. In the previous method, the number of neurons in the initial module is always fixed. Figure 6-15 illustrates this method.

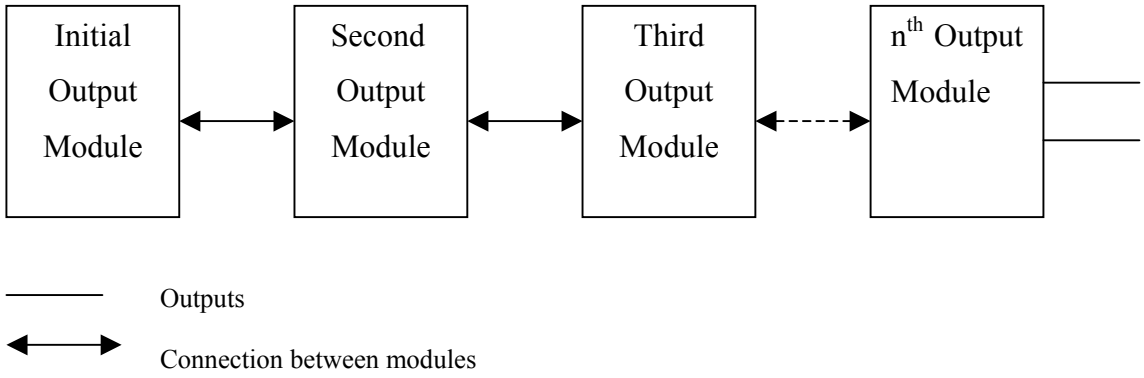


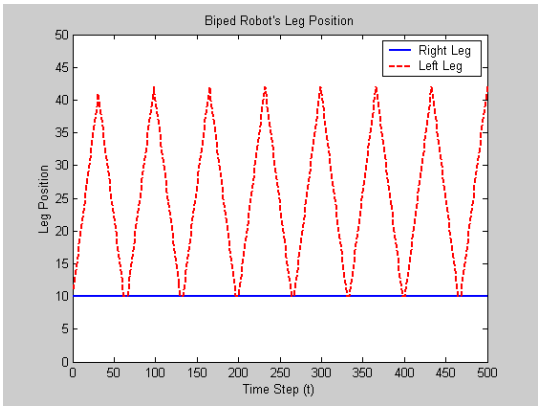
Figure 6-15 Adding modules at the front of the last module

The actuator model described in Figure 5-8 (section 5-4 of Chapter 5) was used for these experiments. The discussion below starts with the second experiment and then continues with the first.

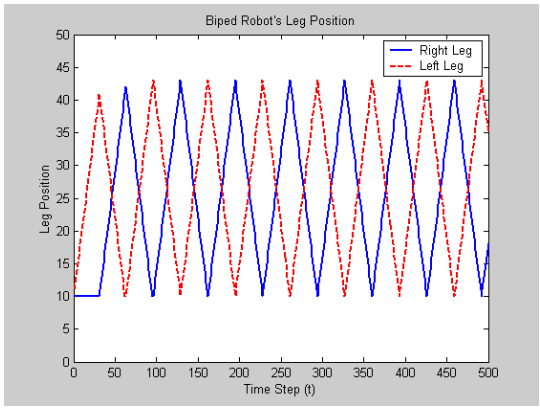
Figure 6-16 shows the leg positions of the robot and the distance moved when modules with 2 neurons are added in front of the last module. Modules with a minimum of 2 neurons were required to control the bipedal robot because there were 2 actuators (legs with one active degree of freedom). A total of 3 modules with 2 neurons in each was required to produce a bipedal walking gait. The robot managed to move a distance of 240 with 2 neurons in the initial module. It can be seen from Figure 6-16 (a) that the right leg is held at position 10 and the left leg oscillates within the desired range. There is no obvious reason for this output leg pattern. This could be the best solution the ES evolved with 2 neurons in the initial module. Then, a module with two neurons was added. The distance moved increased to 450 – see Figure 6-16 (b).

Next, a module with 2 neurons was added and the distance increased to 480 – see Figure 6-16 (c). This is the maximum distance that the robot could move within the specified time scale. Figure 6-16 (d) shows the distance improvement with increasing number of modules.

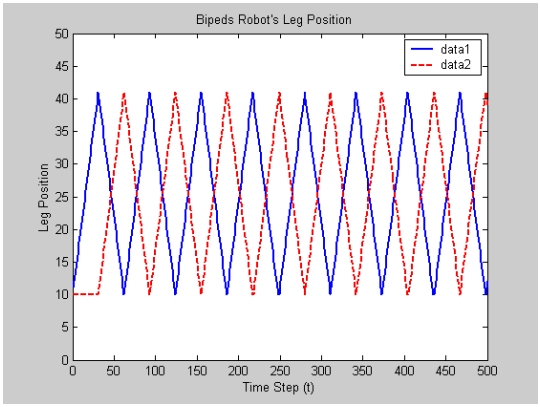
Note: x:y:z where x,y,z… refers to number of neurons in a module



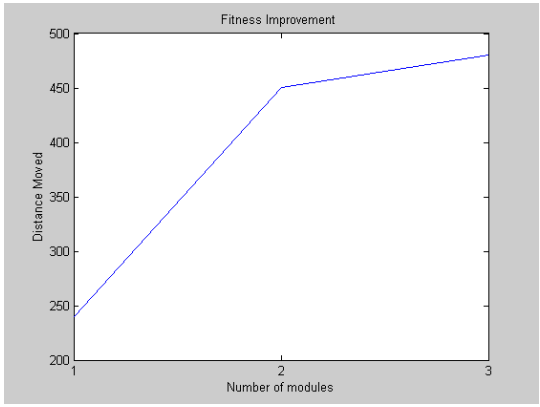
a) 1 module with 2 neurons



b) 2 modules 2:2



c) 3 modules with 2:2:2



d) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the new module.

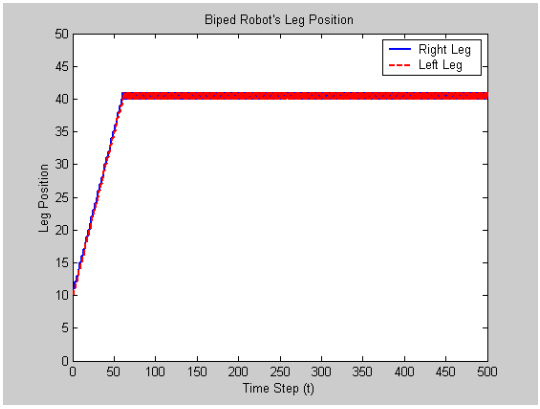
Figure 6-16 Adding modules at the front of the last modules

In this type of growth, the previous modules are behaving like an input to the new module. The new module behaves like a new function (F (New)). The previous modules (F (Old_n) where n is the number of previous modules) becomes a subset of the new function (F (New (Old))). This method is very similar to the Tiling Algorithm (as mentioned in Chapter 4). However, in the Tiling Algorithm, all the neurons in the new module are fully connected to the neurons in the previous module. This is not the case with the growth technique presented here.

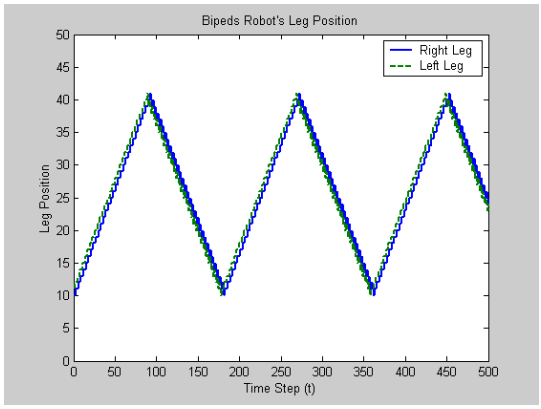
This method may be not biologically viable, because the connections to the outputs may not always change as new modules are evolved.

The results of the first experiment illustrated above in Figure 6-17 were examined. Figure 6-17 shows the leg positions of the robot and the distance moved when modules of neurons are added at the rear of the last module. In Figure 6-17 (a and b) both the right and leg are nearly identical. Figure 6-17 (d) shows the increment in distance moved with increasing number of modules. The distance moved never changed thereafter, with an increasing number of neurons and modules. There were 2, 2 and 4 neurons in each module.

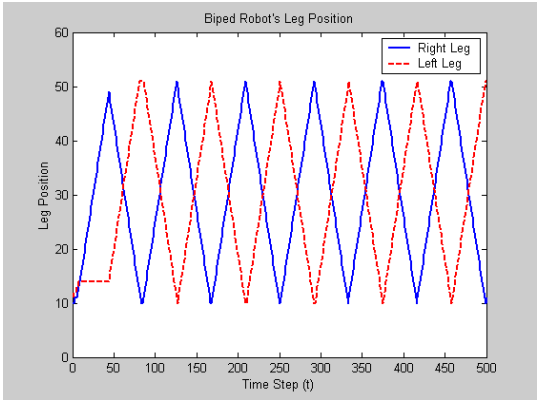
Note: x:y:z where x,y,z… refers to number of neurons in a module



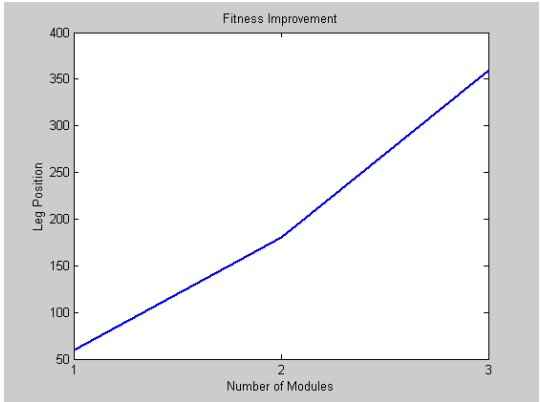
a) 1 module with 2 neurons



b) 2 modules 2:2



c) 3 modules with 2:2:4



d) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the initial module.

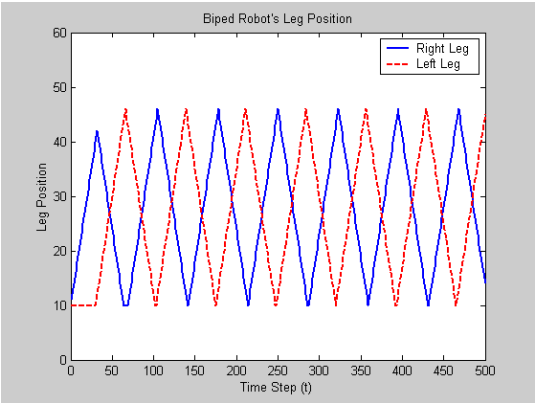
Figure 6-17 Adding modules at the rear of initial module

The significance of this technique will be become apparent after the addition of the third module. This is because the new (third and n^{th} module, where n is number of modules) modules added after this will have a smaller effect on the previous modules (n –1 modules). It is apparent from Figure 6-14 the technique that the newly added module can only affect the previous module. It can be seen from Figure 6-17 (c) that there was a significant improvement in the leg positions when the third module was introduced. The reason for different numbers of neurons in a module has already been discussed in Section 6.2 of this chapter. It was also found that the fitness never increased with increasing number of modules with variable number of neurons thereafter. The maximum possible distance could not be achieved with this type of growth. One possible reason is that there is smaller influence from the newly added module to the earlier modules in the network as more modules are added due to the chain nature of the network structure.

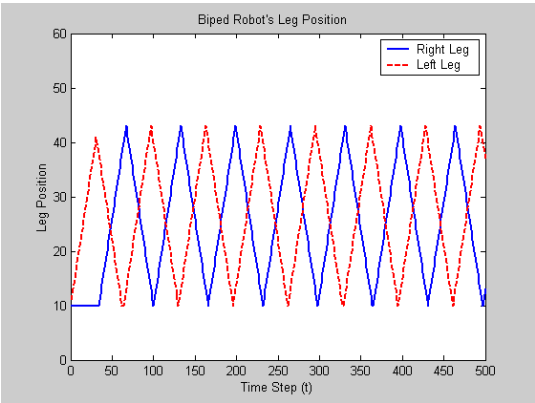
We will now incorporate the second growth technique (Figure 6-15) into the network evolved previously (Figure 6-18). Two modules with 2 and 5 neurons were added to the existing network. It was found that fitness increased with increasing number of modules. The distance moved saturated at 450 steps with despite an increasing number of neurons and modules thereafter.

Figure 6-18 shows the leg positions and distance improvement of the robot for the two newly added modules. Even though the maximum possible distance (480) could not be achieved, the distance travelled was increased by incorporating the first growth method. These results show that bi-directional growth is also an option with large networks.

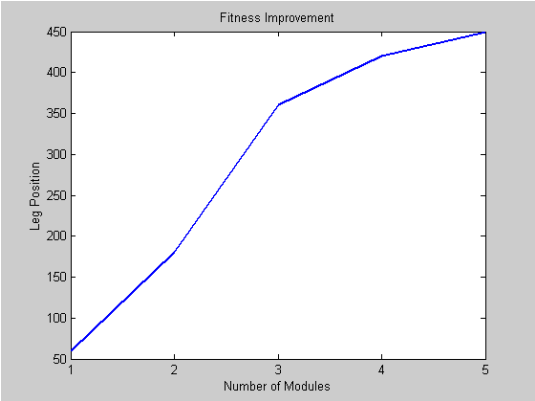
Note: x:y:z where x,y,z… refers to number of neurons in a module



a) 4 modules with 2:2:4:2



b) 5 modules with 2:2:4:2:5



d) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the new module.

Figure 6-18 Adding modules at the front of the last module

6.5 Discussion

In obtaining these results, the objective was to evolve systems which could be compared with previous work done by McMinn [McMinn 2000] [McMinn 2002a].

A total of 7 neurons were required to successfully evolve a bipedal walking gait with the direct growth method (Figure 6-10). The number of generations required to evolve the best bipedal gait was less than 100 (Figure 6-12). It was also found that, when a new module was added, the fitness increased quickly for the first few generations. This shows that the previous modules in the network are contributing to the increment of the fitness. The number of generations was fixed at 50 for every new module added to previously evolved network, unless otherwise mentioned.

[McMinn 2002b] used a more conventional model with a fixed network size and functionality to obtain neural networks capable of both bipedal and four legged gaits. The total number of neurons in the Central Pattern Generator (CPG) used by McMinn was four neurons and these were fully connected (recurrent connections). The final evolved CPG had a lower number of neurons. The suggested number of processing units for the CPG is $2 \times n$ where n is the number of legs (or joints if there are multiple degrees of freedom per leg) based on Golubitsky [Golubitsky 1998]. However, the processing units assumed in the $2 \times n$ suggestion of Golubitsky [Golubitsky 1998] are complex mathematical oscillators, rather than the simple types of neurons as used by McMinn. The Spike Accumulation and Delta-Modulation neuron used in this research is much simpler than the one used by McMinn. McMinn [McMinn 2002b] required 1000 generations to evolve a network to produce a bipedal walking gait. The bipedal walking and jumping gait is the most basic. The number of generations is high because the connection weights and neuron parameters are trained until the best walking pattern is found.

The next gait evolved was the pronk. In this gait all the legs move simultaneously and in phase. The initial set-up of McMinn's network for quadruped gaits is shown in Figure 6-19. The input to the network was a tonic signal, connected to all the neurons in the network. Four outputs were taken from unique neurons. There was no tonic signal provided to the networks used to produce bipedal (walking, jumping) and quadrupedal (trot, pronk) gait in this research. The network could be said to be self oscillating (generating an output without an input signal).

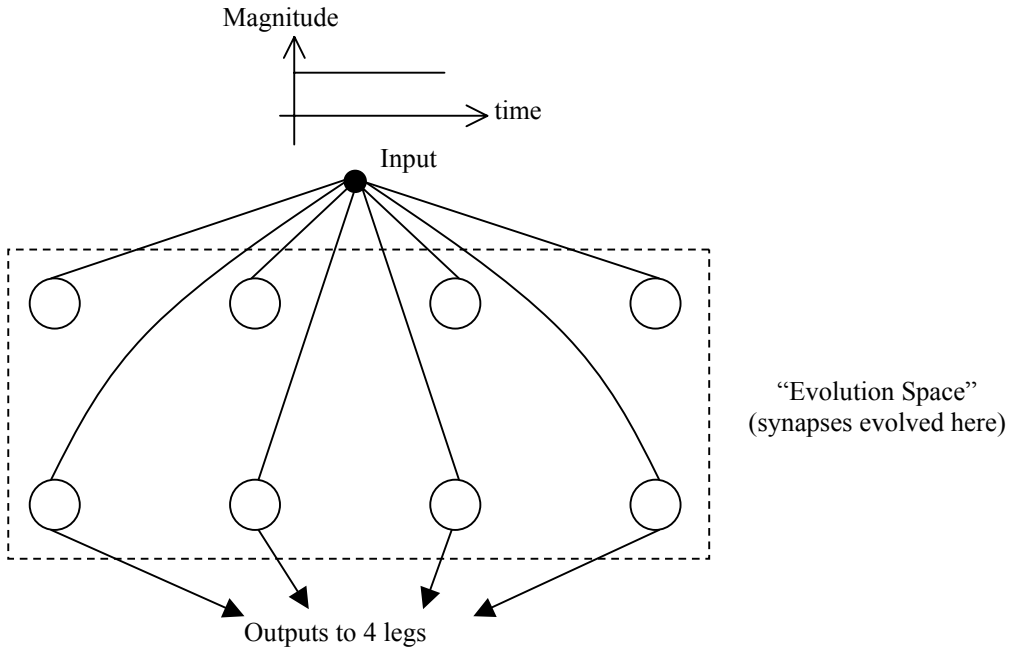


Figure 6-19 McMinn's ANN setup for evolving CPGs (Reproduced by permission of McMinn)

There were 23 neurons in the network evolved using the growth strategy. The total number of generations required to successfully evolve quadruped trot gait was 104 (see Appendix C, Section C.1). The optimal number of neurons for the same network evolved by McMinn was found to be 16 (rather than the initial setting of 8) which allowed all four legs to be controlled and contributing to the appropriate output patterns. McMinn required 1500 generations [McMinn 2002b] to generate the same gait. The main difference is that this system is open-ended and flexible enough for continued development over and above these simpler systems as will be seen in Chapter 7.

Similarly results for bipedal jumping and quadruped pronk gaits were produced and presented in Appendix C, Section C.2.

Chapter 7

Results From Multiple Functions

7.1 Introduction

The results in the previous chapter were based on a mechanically simple robot. In this Chapter, further results are obtained using a more complex robot body configuration. There are also discussions and results on other applications of the growth technique, which illustrate the universality of the approach.

7.2 Evolution of the Body-plan

A major part of the modular evolution scheme is not the evolution of the neural network itself, but the evolution of the robot in terms of its body plan and the environment it is interacting with. Another way of looking at the evolution of the environment is to say that it is the fitness function – in other words, the fitness function changes and evolves along with the robot. Total evolution is illustrated in Figure 7-1.

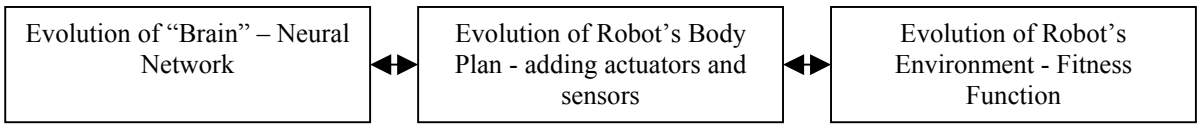


Figure 7-1 Total evolution

All these aspects must go hand-in-hand during the robot's development. Let us take them separately.

Firstly, consider the body plan. This is informed by two separate branches of science: Evolution and Embryology. Evolution is the development of animals over vast periods of time, starting in the pre-cambrian era over 570 million years ago, with single celled animals. More insight into this can be found in Section 3.3.7 of Chapter 3. Embryology is the study of the development of the embryo, which echoes Evolution (the embryo starts as a single cell and passes through a similar pattern of development to evolution – as through it is replaying the evolutionary history of the animal).

There are two things which have to be added to the robot's body as it evolves: sensors and actuators. With actuators, the proposal is to start with one degree of freedom and evolve the model progressively to its final form by adding one further degree of freedom at a time. In the case of limbs one joint at a time can be added. Limb movement sensors also have to be added if needed.

Note that there is a limit to body plan evolution as far as actuators are concerned. For example, in the case of a robotic biped, when all the joints are in place and are able to be controlled (similar to the evolution of an austrapithicus), then only the "mind" (Neural Network) and with it the environment follow. In this Chapter we will consider the evolution of ANN for two degrees of freedom per leg is considered. More information about sensor, environment and mind evolution is presented in the next chapter.

7.3 Results from Further Degrees of Freedom

Once it was established that the technique could be used to grow even a single function as described in Chapter 6, the research moved on to consider multiple degrees of freedom. This was tackled by adding a joint to each of the biped's legs as shown in Figure 7-2 and described previously in Section 5.4.

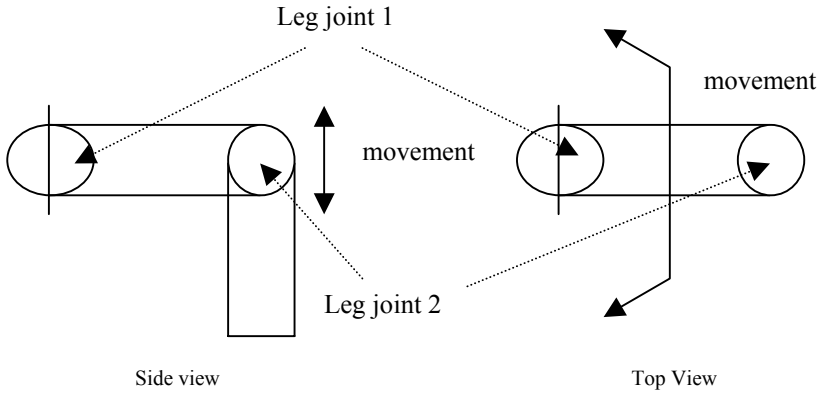


Figure 7-2 Leg with two active degrees of freedom

The initial robot body plan was one with one passive and one active degree of freedom leg as shown in Figure 7-3. The robot's leg has to move from the forward to rear position within a desired range. ANNs were successfully evolved previously to produce a bipedal walking gait for Range 1. However, in the new case, the environment is deconstrained so that the controlling network has to produce walking motion for the new range (Range 2 as shown in Figure 5.11).

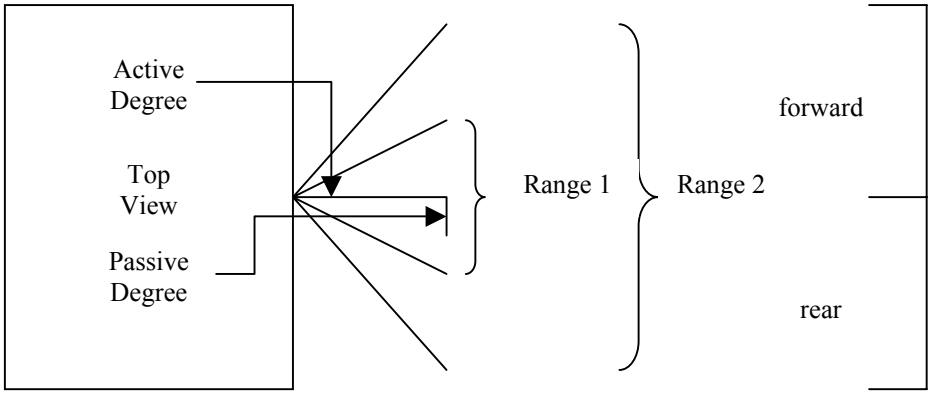


Figure 7-3 Robot's leg movement range

The arrangement of the networks for this task is shown in Figure 7-4. The previous (single active degree of freedom) system is retained and new modules are added to build up the network for the newly added functionality. Connections are allowed to the previously developed network so that the new sections can take timing cues. It was decided to use this method rather than to add external sensors on the new leg sections in order to test the system's flexibility (external sensors would make the problem easier). The Evolutionary Strategy used was as previously explained.

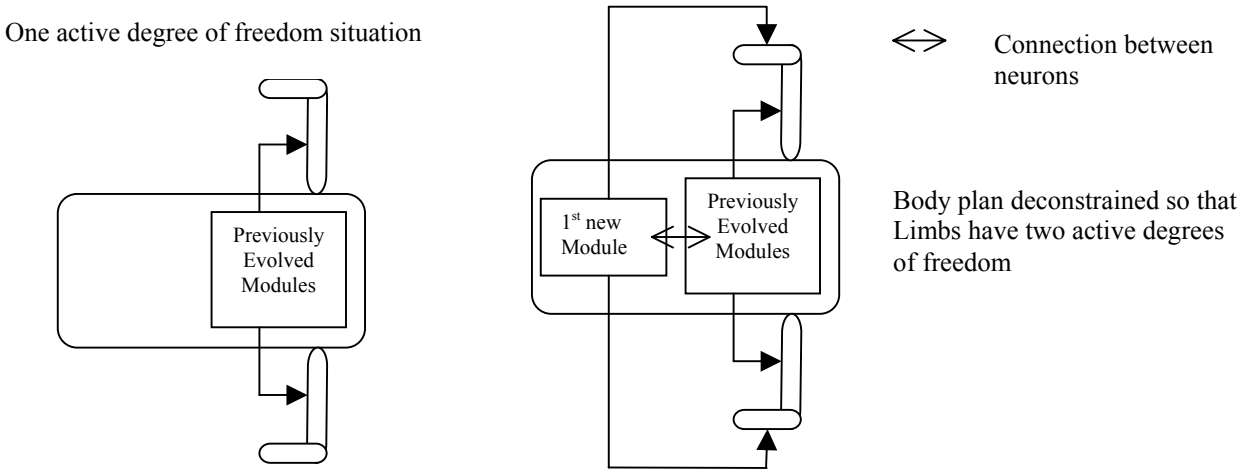


Figure 7-4 Arrangement of body-form for second degree of freedom limbs

The Spike Accumulation and Delta Modulation [Shigematsu 1996] neuron model described in Chapter 5 was used to evolve the network to control the second degree of freedom. There were 3 modules with a total of 7 neurons in the previous network for the single degree of freedom. The first new module with 2 neurons was added to the network. Each neuron in this module was permanently connected to the second joint of the actuator as shown in Figure 7-5.

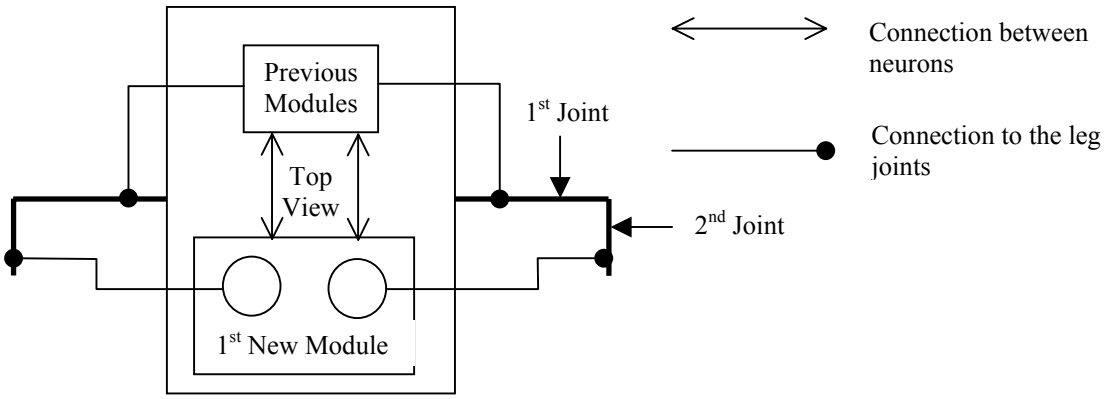
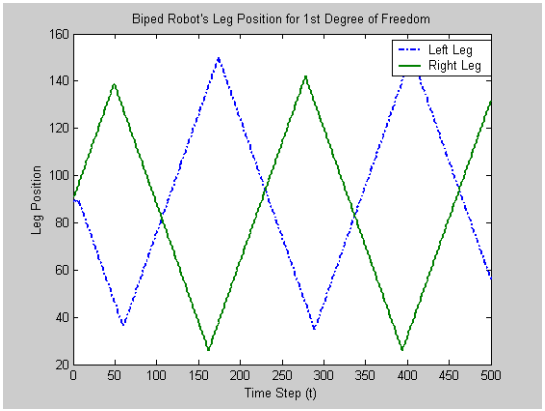


Figure 7-5 Neuron connections

The neurons in this new module have to produce a +1 pulse to lift the joint off the ground and a zero pulse to rest the joint on the ground. ANNs were evolved for this purpose and Figure 7-6 (a – f) shows the leg positions for both the joints after the initial module has trained and the final module is added. Figure 7-6 (g) shows the improvement in fitness as the new modules are added. The entire ANN to control both active degrees of freedom is shown in Figure 7-7. The robot's leg position for the remaining modules are presented in Appendix C, Section C.3. Neurons 1 and 2 in the initial module control the first joint of the actuator and neurons 8 and 9 control the second joint.

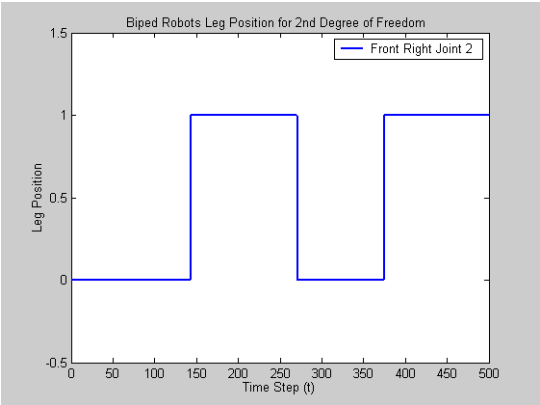
Note: x:y:z where x,y,z… refers to number of neurons in a module

Initial Module

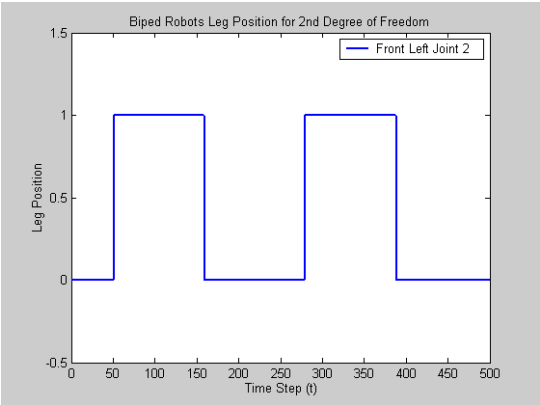


a) Leg Position for both first degree of joints:

4th module with 2 neurons (2:2:3:2 previous modules)

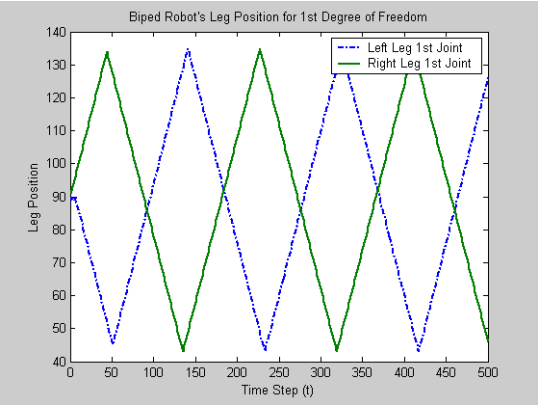


b) Leg Position for front right second joint



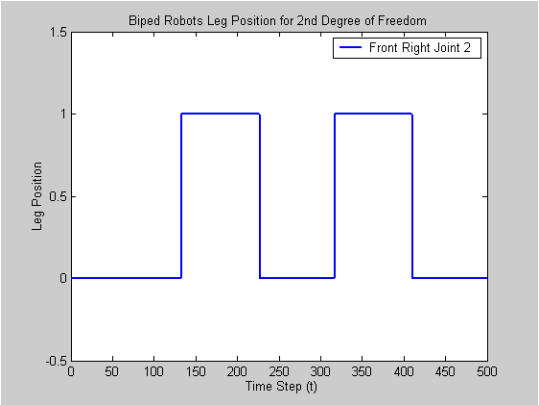
c) Leg Position for front left second joint

Final Module

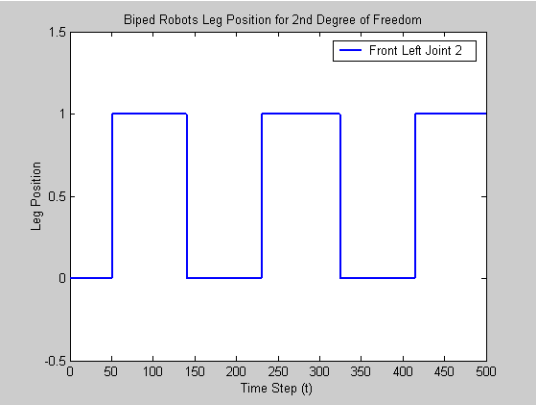


d) Leg Position for both first degree of joints:

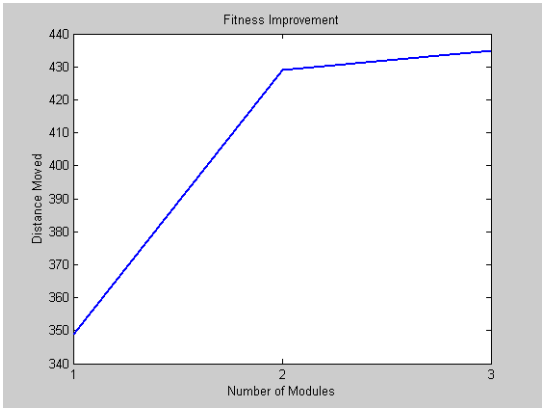
6^{th} module with 1 neuron (2:2:3:2:1:1 final modules)



e) Leg Position for front right second joint



f) Leg Position for front left second joint



g) Distance moved with increasing in number of modules. Output to the actuator taken from the neurons in the 1st module i.e. neuron 1,2,3 & 4.

Figure 7-6 Bipedal walking gait leg positions for both the active joints

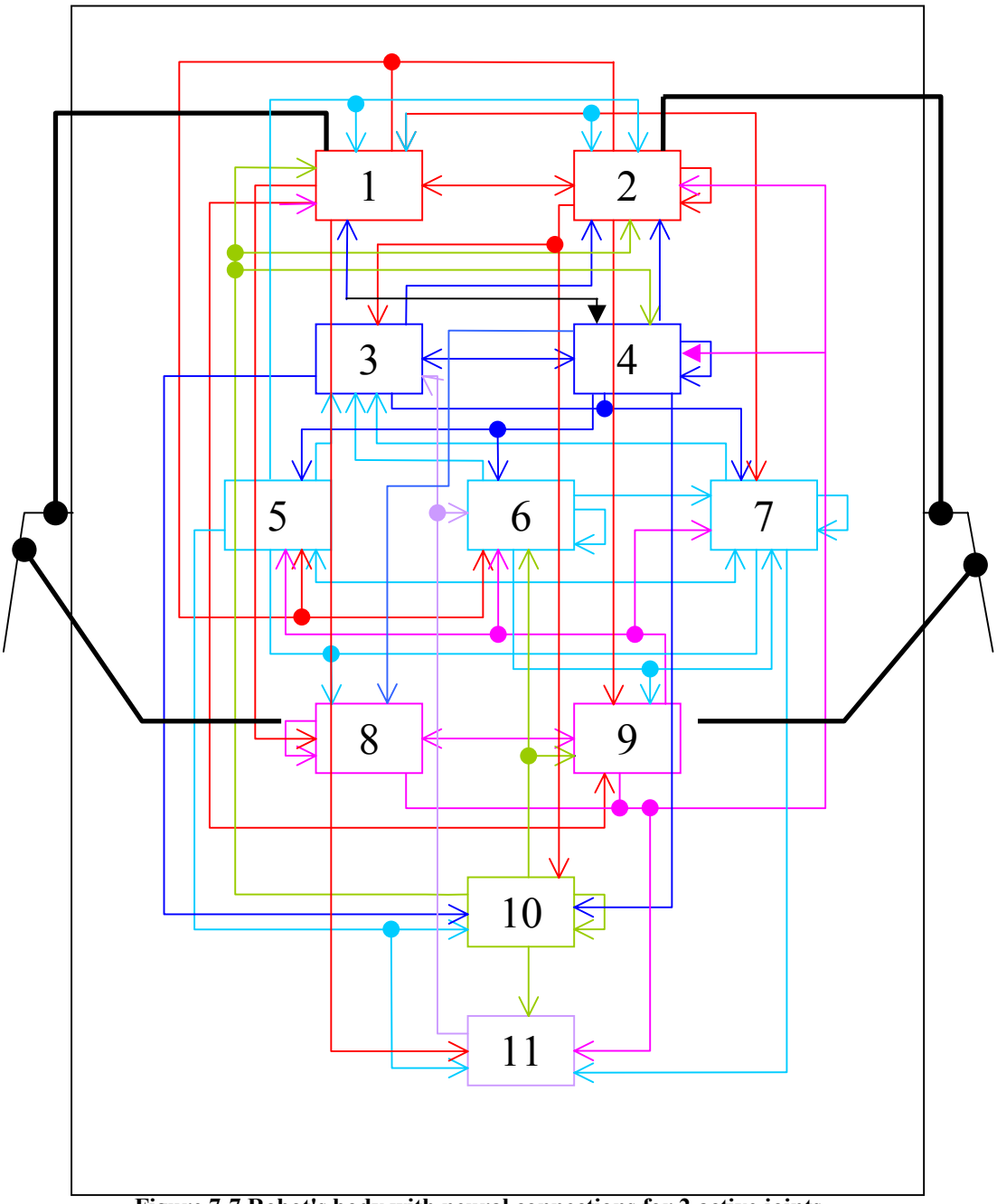


Figure 7-7 Robot's body with neural connections for 2 active joints

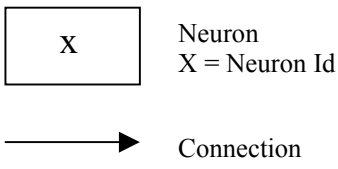


Table 1 below shows the number of modules in the network and the neuron Ids. Modules 4 to 6 are the new modules evolved on top of the previous network. A total of 4 neurons are required to control the second joint and to produce the bipedal walking gait. Modules number 1 and 4 are the output modules.

Module Number	Neuron Ids
1	1, 2
2	3, 4
3	5, 6, 7
4	**8, 9**
5	**10**
6	**11**

Table 1 Module number and neuron Ids

Table 2 shows the evolved connections between neurons when modules number 4, 5 and 6 are formed.

Neuron Id	Connection From Neuron Id
1	9, 10
2	8, 9, 10
3	11
4	8, 10,
5	9,
6	9, 10, 11
7	9
8	**1, 4, 5, 7, 8, 9**
9	**1, 2, 6, 7, 8, 10**
10	**2, 3, 4, 5, 10**
11	**1, 5, 7, 8, 9, 10**

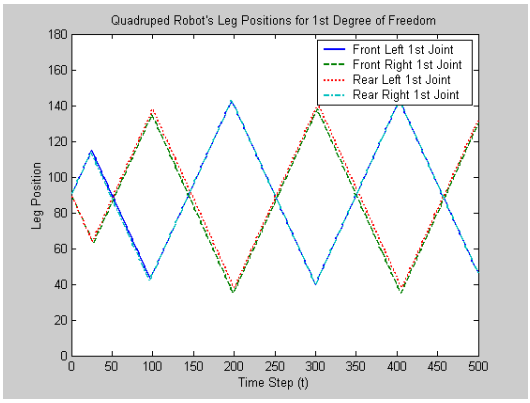
Table 2 Evolved connections to and from neurons in the network

By analyzing the connectivity table (Table 2), it can be seen that more connections have evolved from the previous modules to the neurons in the new module. It is also noticeable that only a few connections are formed from the new module to previously evolved modules. Neurons in the new module are often connected to the neurons in the output module. The new module behaves as a signal filter. It observes the unwanted signals from other modules and outputs an improved signal to other parts of the network. The network structure resulting from the system outlined appears, to the casual observer, to be a fully interconnected network. However, closer inspection of its functionality shows that different areas of the network are specialized to handle different functions – a structure similar to that present in the biological brain, where localized regions of an apparently interconnected structure perform specific tasks. This is a direct result of the evolutionary process.

Similarly ANNs have been evolved to control a quadruped with 2 active joints per leg. Again, the body plan was deconstrained as shown previously in Figure 7-3. New modules were evolved to control the second joint on top of the existing network used to produce trotting gait in a quadruped, as described in Chapter 6, Section 6.3. In this quadrupedal configuration neurons in the new module have to produce a +1 pulse to rest the joint on the ground and a zero pulse to lift the joint off the ground. Six modules with a total of 23 neurons were required in total to produce the gait (trot). Figure 7-8 (a – j) shows the leg positions of all the joints after the initial module is trained and after the final module is added. Figure 7-8 (k) shows the improvement in fitness as new modules are added. The robot's leg positions for the remaining modules are presented in Appendix C, Section C.4.

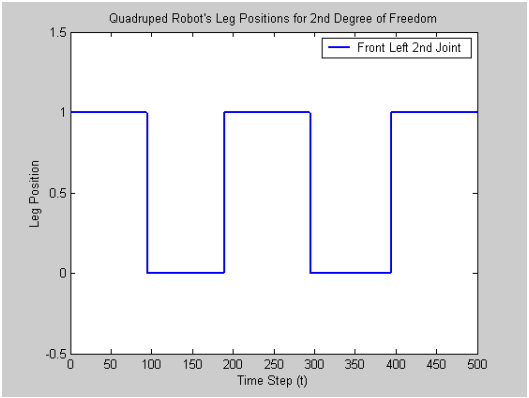
Note: x:y:z where x,y,z… refers to number of neurons in a module

Initial Module

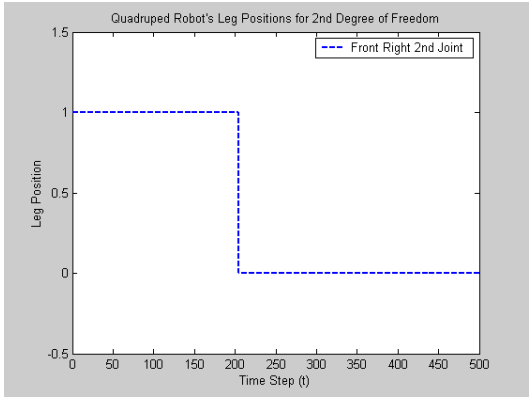


a) Leg Position for both first degree of joints:

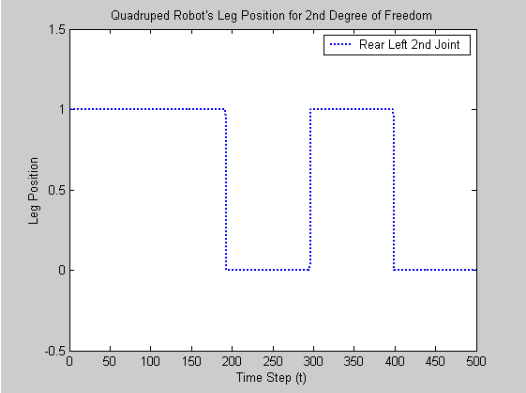
7^{th} module with 4 neurons (5:3:2:4:4:5 previous modules)



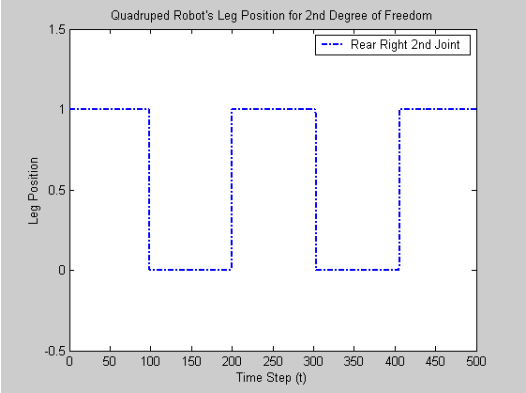
b) Leg Position for front left second joint



c) Leg Position for front right second joint

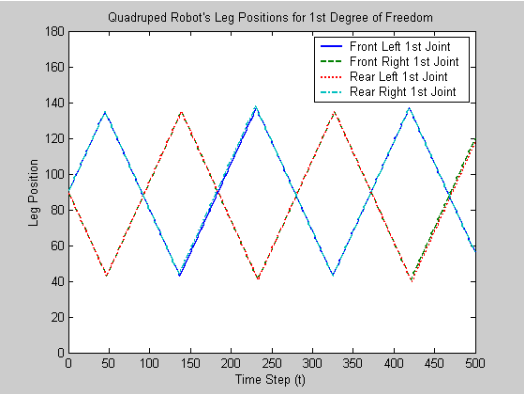


d) Leg Position for rear left second joint



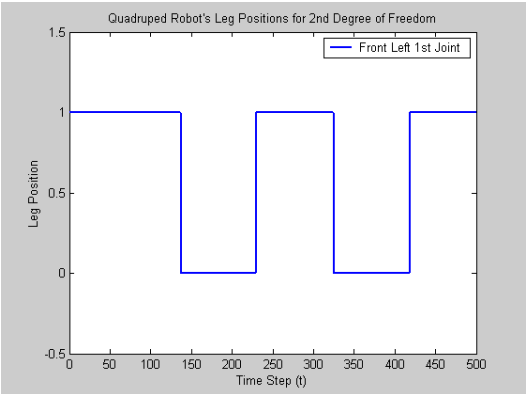
e) Leg Position for rear right second joint

Final Module

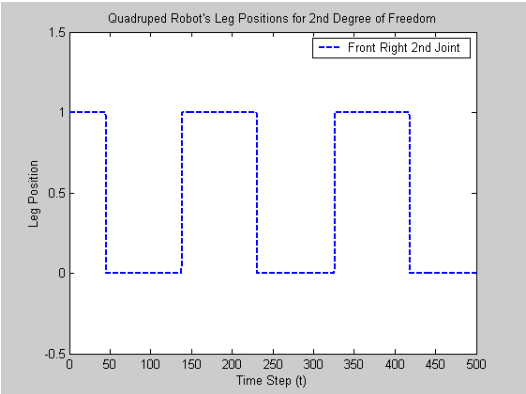


f) Leg Position for both first degree of joints:

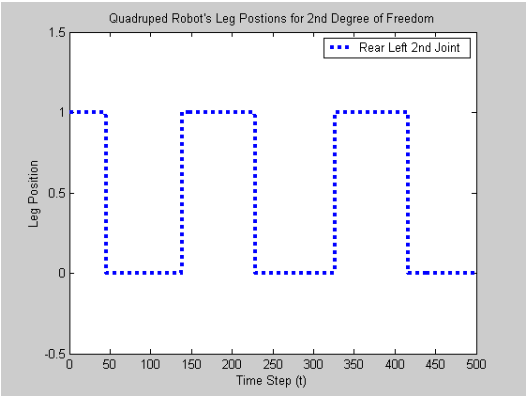
11^{th} module with 2 neurons (5:3:2:4:4:5 final modules)



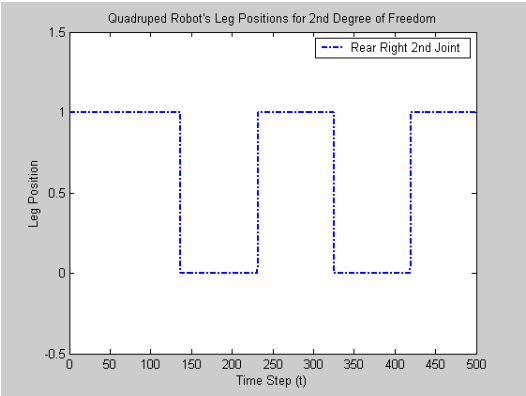
g) Leg Position for front left second joint



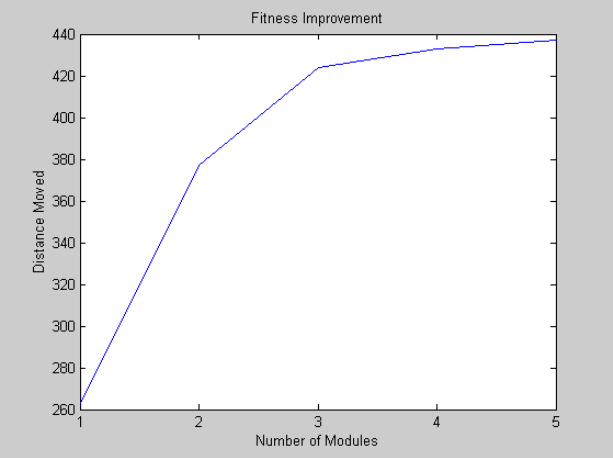
h) Leg Position for front right second joint



i) Leg Position for rear left second joint



j) Leg Position for rear right second joint



k) Distance moved with increasing in number of modules. Outputs to the actuator are taken from the neurons in the initial module.

Figure 7-8 Quadruped trot gait leg positions for both the active joints

Table 3 shows the distribution of number of neurons in the new modules. The module number starts from 7 because there were already 6 modules in the initial network. Since there were 4 actuators in the quadruped, we need 4 neurons in the seventh module to control the second joint. Each neuron in this module will be connected to the second joint of all the actuators. The total number of neurons required to successfully control the second joints and produce the trotting gait for the new range is 35 in 11 modules.

Module Number	Number of Neurons
7	4
8	3
9	2
10	1
11	2

Table 3 Number of neurons in each module

7.4 Copy And Paste Technique

Having successfully tested the evolutionary idea on bipedal and quadrupedal systems, the work was expanded to investigate the reuse of successfully evolved modules in "copy and paste" evolution (making previously evolved sub-units available for reuse in the system). This would mimic the biological scenario of whole strings of DNA being copied to other areas within the genome and would be useful in evolving repeating structures. For example, extra limbs or body sections are common genetic mistakes from incorrect copying of genes. It was therefore felt that it would be reasonable to allow the algorithm to reuse previously evolved networks (including their sensors and actuators).

A biped was successfully evolved in this manner by taking two single legs with one active degree of freedom sections and allowing the algorithm to grow an intermediate network, which interfaced the two pre-evolved sections. The leg positions oscillate in range 2 as described earlier. This interface (or translation) network was built up in exactly the same way as was previously described (Section 5.6 of Chapter 5). Figure 7-9 illustrates the "copy and paste" concept.

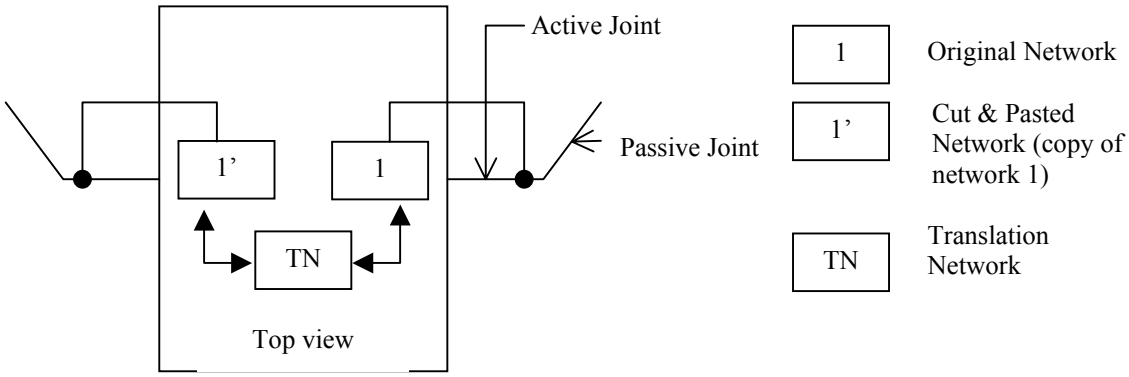
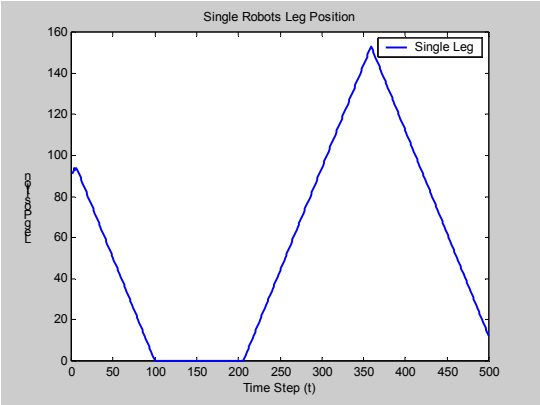


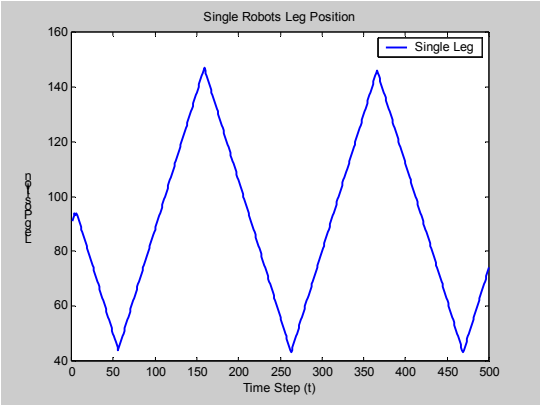
Figure 7-9 Illustration of copy and paste technique

Figure 7-10 (a- b) shows leg positions of a single leg as modules are added. Two modules are required to produce the hopping gait. There were 2 and 3 neurons in each module. Figure 7-10 (f) shows the increase in fitness levels as modules are added (up to module 2 for this setup). Later, this network is used with copy and paste technique, and the translation network is evolved to produce a walking gait. Figure 7-10 (c - e) shows leg positions for both the legs as modules are added. Figure 7-10 (f) shows the fitness improvement as modules are added to the network for the system. Three translation modules were needed to produce the walking gait within the desired range. There are two neurons in the first and second modules and three neurons in the final module. A total of 17 neurons was required to produce the walking gait.

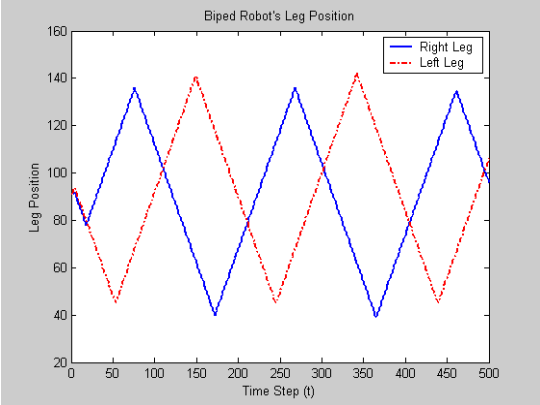
Note: x:y:z where x,y,z… refers to number of neurons in a module



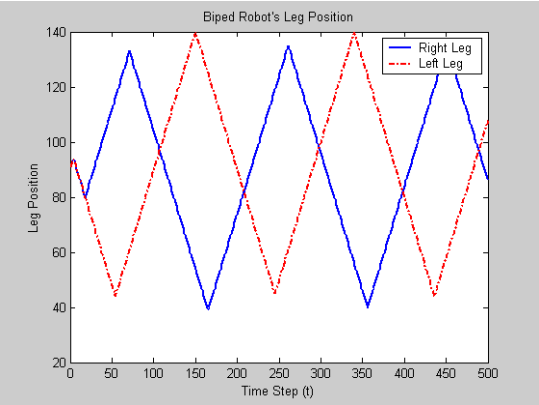
a) Single leg position, 1 module with 2 neurons



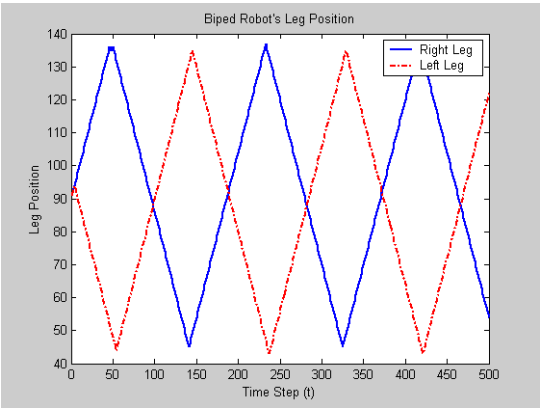
b) Single leg position, 2 modules 2:3 neurons



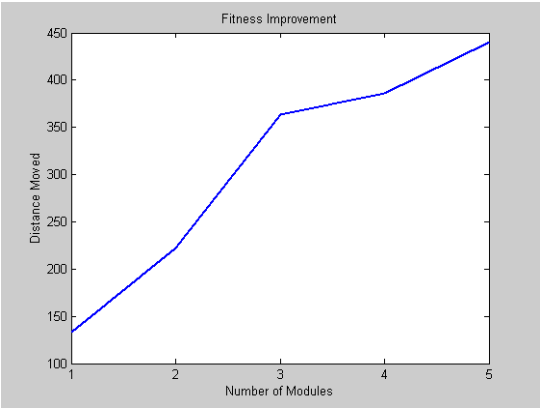
c) Previously evolved subunit reused for bipedal walking gait, 5 modules 2:3:2:3:2



d) Biped robot's leg position, 6 modules 2:3:2:3:2:2



e) Biped robot's leg position, 7 modules 2:3:2:3:2:2:3



f) Fitness Improvement with increasing number of modules

Figure 7-10 Bipedal walking gait using the Cut and Paste technique

From Chapter 6, three modules with a total of seven neurons were required to evolve this gait. This "copy and paste" technique generates more neurons in the network. If a jumping gait is to be evolved for a biped or a pronk for a quadruped, this technique may be useful as individual legs do not have to be evolved separately. In the worst scenario, (complete deletion of the translation network), it would still enable the individual networks to function as normal (although not synchronized).

7.5 Dual-Gait Network

To further test the system, it was decided to attempt a network which was capable of producing several gaits and switching between them. This is usually considered a difficult problem and the biological mechanism behind such a translation has not yet been discovered. This experiment was started with four neurons in the initial module. The ES determines the connections among the neurons in the module. The first two neurons are responsible for producing a walking gait and the others are for producing a jumping gait in a bipedal system. In this system, the neurons are not directly connected to the actuator. It is assumed that there is a switch (which could of course be another neuron) to relay the network signal to the actuator. Figure 7-11 illustrates the method.

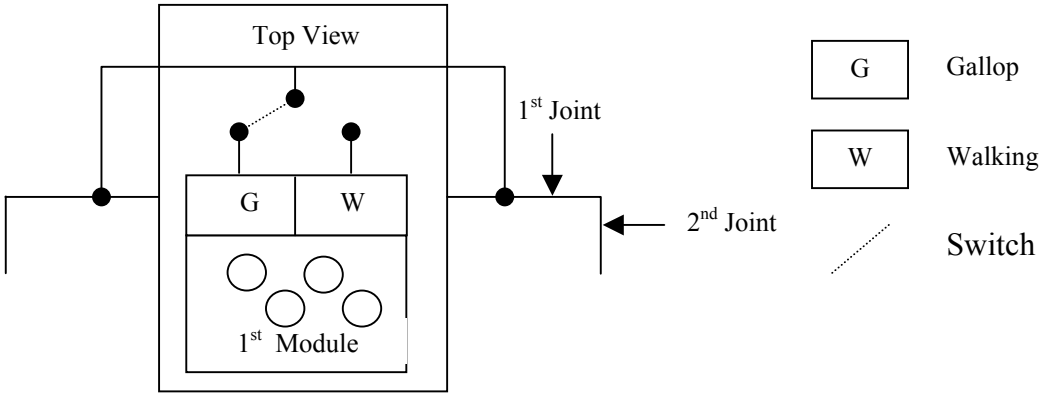
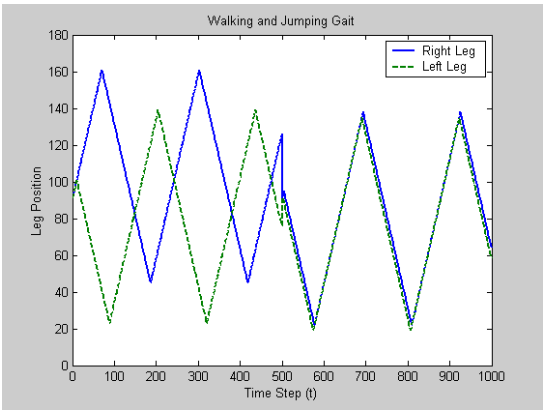


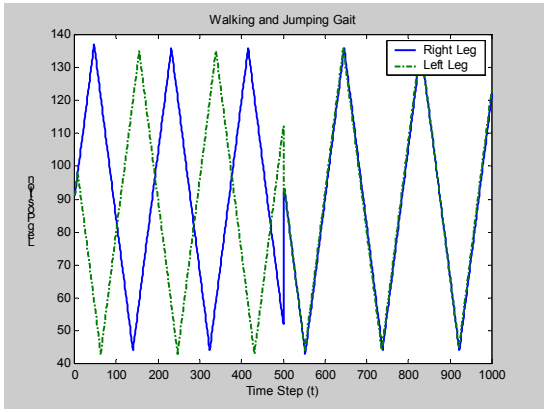
Figure 7-11 Double Gait System

The network was evolved as previously described. Figure 7-12 shows the leg positions for the different gaits as modules of neurons are added to the previously evolved network. The initial network with 4 neurons failed to switch between the different gaits. The problem was solved with 5 neurons. The extra neuron could behave like a pace maker to switch between the gaits.

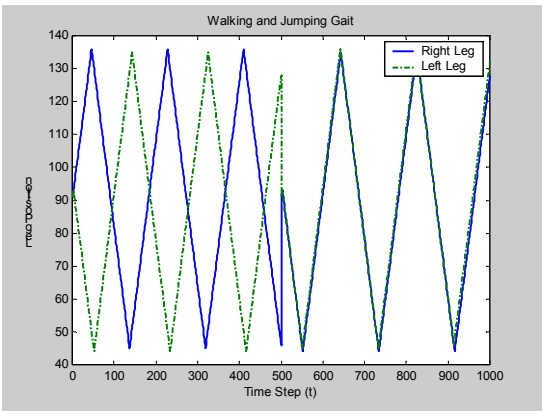
Note: x:y:z where x,y,z… refers to number of neurons in a module



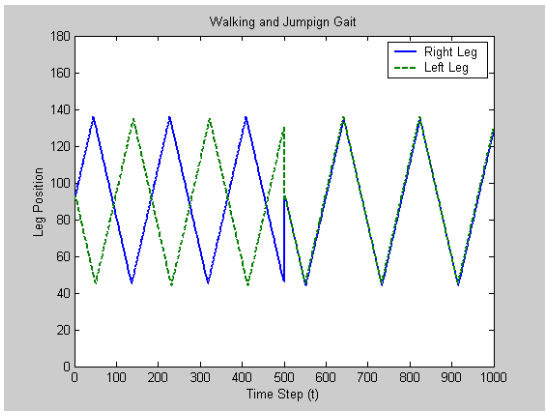
a) 1 module with 5 neurons



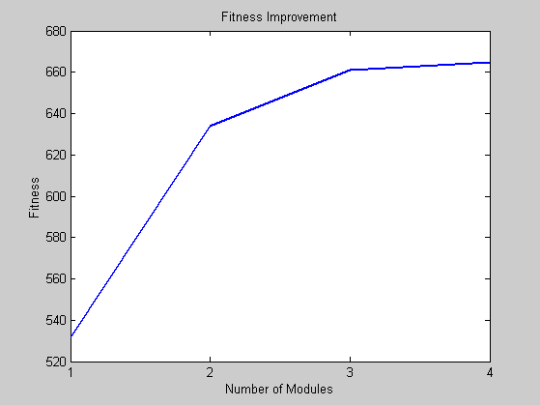
b) 2 modules with 5:3 neurons



c) 3 modules with 5:3:2 neurons



d) 4 modules with 5:3:2:1 neurons



e) Fitness improvement with increasing number of modules

Figure 7-12 Different Gaits obtained

7.6 Discussion

McMinn used an alternative strategy [McMinn 2002] for evolving Central Pattern Generator (CPG) networks. This strategy was described in Chapter 2 Section 2.3 in layman's terms. The alternative strategy was based on the observation that neural networks perform best when large homogenous networks are split into several smaller modular ones, each of which can operate as an independent unit, but can work together to form a larger whole [MacLeod 1999]. To accomplish this, the CPG networks were separated into two functional modules. The first unit performs the task of an oscillator and the second modifies the oscillations to form the appropriate gait patterns. The connection between the two units is similar to the work presented by Prentice [Prentice 1995]. In McMinn's work the first unit (the oscillator) was the previously evolved biped walk CPG.

The second requirement was the pattern generator which converts the bipedal walking pattern into the quadruped gaits. The networks were reduced to eight neurons since the simpler task allows fewer neurons to be used. The quadruped gaits of 'walk', 'trot', 'pace', 'gallop' and 'pronk' were successfully produced by McMinn [McMinn 2002] using this approach. The number of generations required to evolve all these different gaits was reduced to 500.

The “copy and paste” technique is similar to the alternative strategy used by McMinn. A biped was successfully evolved in this manner by taking two single legs with one active degree of freedom sections and allowing the algorithm to grow an intermediate network, which interfaces the two sections. A quaduped could be evolved in the same manner (by “copy and paste” the bipedal network).

A total of 23 neurons were required to evolve a quadrupedal trot gait using the direct growth technique. There were 12 neurons in McMinn’s quadruped CPG. This technique provides a flexible evolutionary alternative to the more rigid structures even though more neurons were required using the direct growth technique. The numbers of generations required are also lower (Appendix C, Section C.1) than McMinn’s alternative strategy (500). It was found that using this approach, the performance of the CPG was improved and was quicker to evolve, while the network remained modular. However, the superiority of the modular scheme is shown by its success in evolving different gaits while the homogenous or coupled (alternative strategy) network of similar size could not.

Chapter 8

System Integration

8.1 Introduction

The results presented in Chapter 6 and Chapter 7 show the applicability of the Incremental Evolution (IE) technique to robotic control systems. It has been shown that the method allows the robot's body plan and the controlling neural network to build from a simple to a complex form. The technique has been successfully used to evolve neural control systems up to the level of those required for quadruped robots. Other applications of the technique have also been discussed in the latter part of Chapter 6. In this Chapter, the experiments will concentrate on incorporating the technique into a more advanced robot with a vision system. Later, the technique is used to grow and incorporate both locomotion and vision into the same structure to form a system.

8.2 Vision System

Since the discussion in Chapter 6 and 7 was based on networks which mainly control outputs (producing walking patterns), it was also decided to build networks for a vision system using a similar method. This provides a contrast since such networks are involved in processing inputs. In particular, to provide a difficult but realistic task, the network was configured to mimic a toad's behavior as reported by Ewert [Ewert 1985, 1987] developed by Arbib [Arbib 1995] and implemented by Reddipogu [Reddipogu 2002] (see Section 2.4 of Chapter 2).

Before proceeding further, consider the development of the human sensory system. There are limits to body plan evolution as far as actuators and sensors are concerned. For example, in the case of a robotic man, when all the joints are in place and able to be well controlled (the robotic equivalent of an austrapithicus), then only the "mind" neural network will continue to evolve with a more complex environment.

The same idea applies to sensors – for example, Sight, Hearing, Smell, Taste and Touch. It is likely that these will be evolved along with or after basic locomotion (going up the ANS model starting at the bottom).

We can assume that all such systems start with the simplest possible arrangement (just a single sensor - the equivalent of "one degree of freedom") and become more complex incrementally [Ewert 1985]. Let us consider sight as an example. This would start in nature as just a light sensitive spot on the skin of the animal and develop eventually into an organ capable of forming an image. Figure 8-1 shows the development of the vision system from a single pixel.

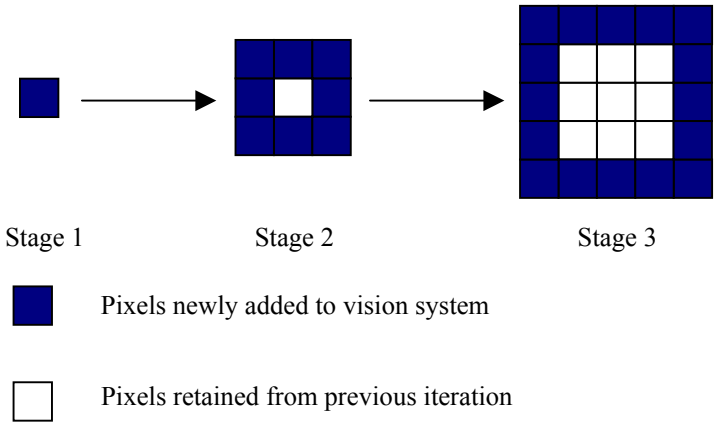


Figure 8-1 Vision system

To do this, the input sensor and the range of patterns to which it is exposed are allowed to grow in a similar way to that previously explained for the body plan. The pixels on the grid can be in two different states, either 'ON' (black pixels) or 'OFF'(white pixels). There are three different stages involved in the evolution of the vision system explored here. It starts as a single pixel in Stage 1. Then a 3 x 3 sensor block was added to vision system in Stage 2. Finally, a 5 x 5 block was added. Figure 8-1 illustrates the evolution at different stages. Appropriate leg patterns have to be produced on the 4 output neurons. Figure 8-2 shows the progression in sensor complexity with the desired leg patterns for different inputs. The repertoire of patterns available ranges from simple fight or flight responses to the identification of obstacles in the field of view.

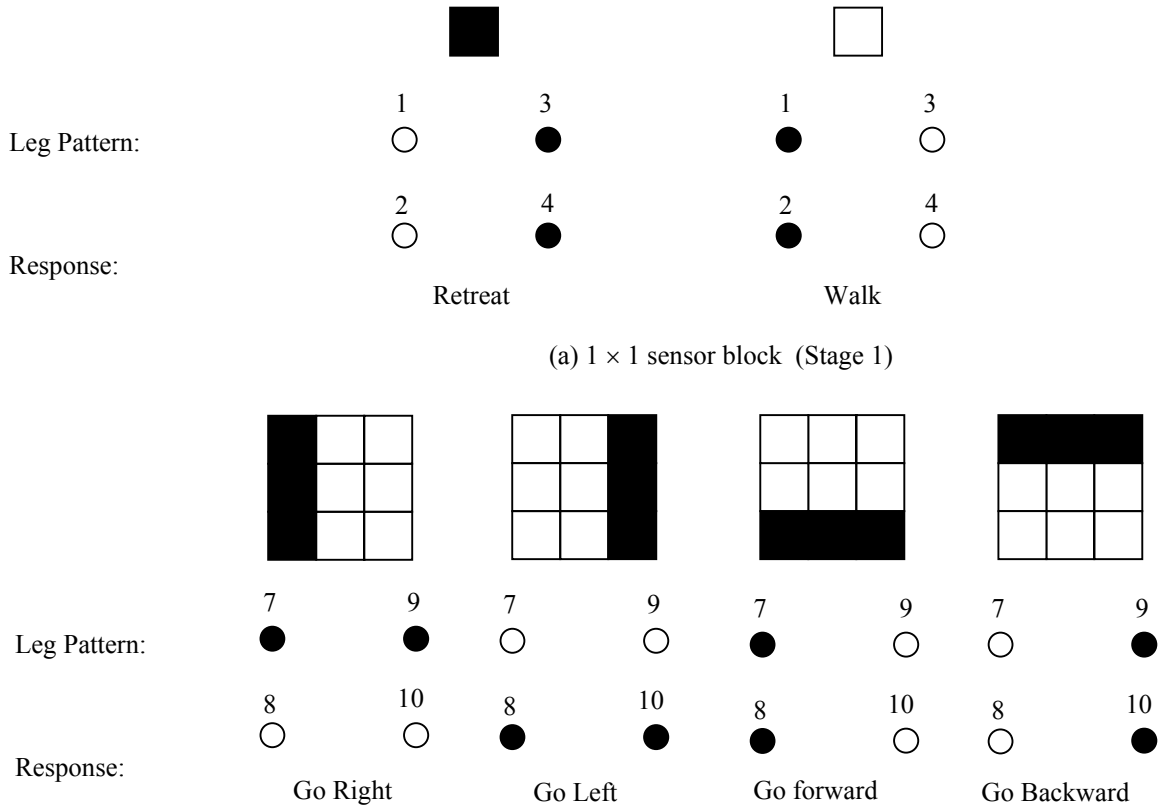


Figure 8-2 Evolution of vision sensor complexity

The discussion below is based on the Stage 1 evolution of the vision system but is applicable to the other stages as well.

The leg pattern indicates which output gait should be triggered for an input. Firstly, a module with 4 neurons was trained to produce the initial leg pattern (retreat). The network was awarded a score of 10 for successfully producing the correct output pattern. Then, the connection weights and neuron parameters of the current module were frozen. Secondly, a new module was added to the previous network in order to train both the patterns (retreat and walk). The network was awarded 20 points if it managed to reproduce the correct output pattern for both these inputs. The vision sensors (pixels) are fully connected to the first module and connections to other modules are determined by the EA. The outputs were always taken from the first module.

The reason for connecting all the sensory inputs to the first module was to make sure that, at least at one stage, the sensory inputs are relayed to all the neurons. Figure 8-3 illustrates the above explanation.

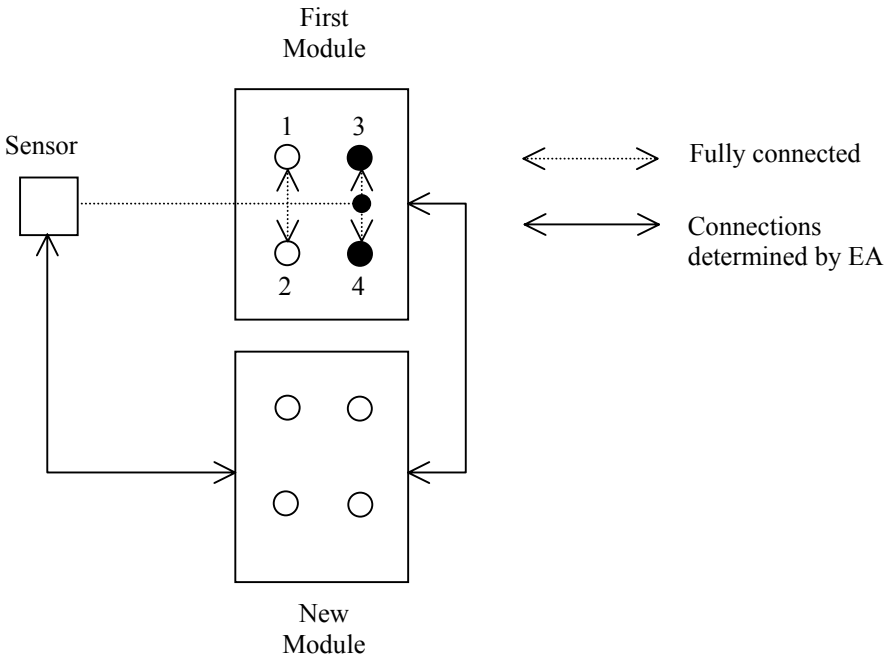


Figure 8-3 First stage evolution

The Spike Accumulation and Delta Modulation [Shigematsu 1996] neuron model described in Section 5.2 of Chapter 5 was used to evolve the modules. The duration for all the vision experiments is 1 timestep.

A module with 4 neurons was trained successfully to produce the retreat response. Then a new module with 2 neurons was added to produce both (retreat and walk) leg patterns. The explanation on different numbers of neurons required in the newly added modules has been given in Chapter 6. Figure 8-4 shows the output of the leg patterns for different inputs and the fitness improvement as new modules were added to the previous modules.

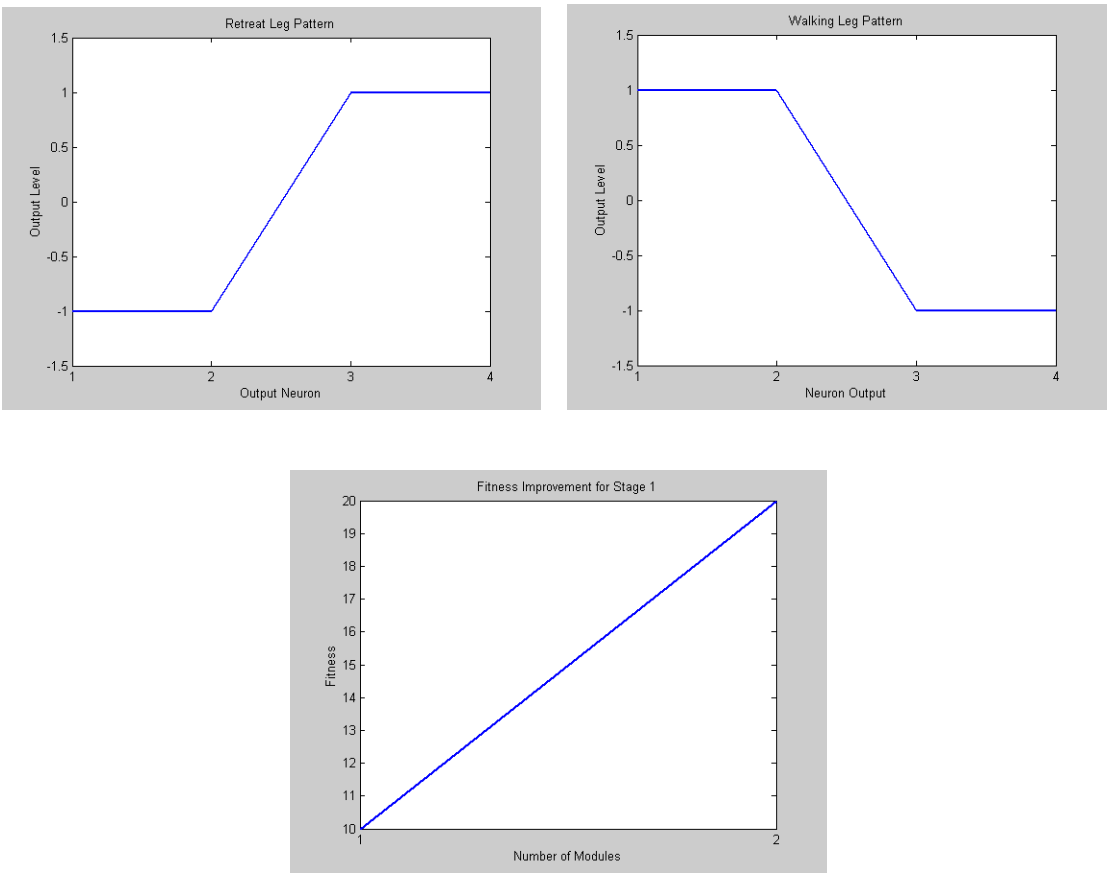


Figure 8-4 Vision output for stage 1

Next, a 3 × 3 sensor block was added to the vision system as shown previously in Figure 8-2 (b). A new controlling network was evolved at each stage. Connections were not allowed between the different stages (although there is no specific reason for doing so). Two modules, each with 4 and 3 neurons have been evolved to produce the

"Go Right" and "Go Left" responses. Later, new modules with 2, 3, 4, and 5 neurons were added but these modules failed to produce the third leg pattern ("Go Forward"). Figure 8-5 shows the fitness (score) improvement for the second stage of the vision system. It can be seen from the graph that the fitness levels off at 20 with an increasing number of modules thereafter.

Figure 8-5 Vision output for stage 2

It seems that the network has problems producing 3 or more different leg patterns. It is very likely that the neurons have difficulty dividing the solution space into different domains. Another experiment (equivalent to Figure 6-15, Section 6.4 of Chapter 6) was conducted where the outputs were taken from the newly added module. Figure 8-6 illustrates the concept.

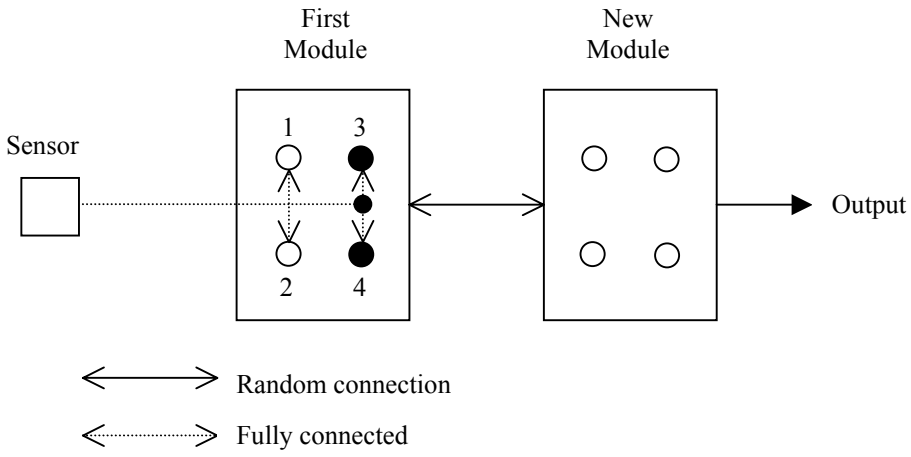


Figure 8-6 Adding new module in front

Even with this technique the network failed to produce all the required patterns. At this stage it was thought that the neuron functionality might be causing the problems. Similar problems were faced when the MMM neuron model was used for the evolution of the bipedal locomotion at the beginning of the research (refer to Section 6.2). It was thought a simplified neuron model might perform better.

The most common type of artificial neuron model was used and is shown in Figure 8-7. This is the modified standard “McCulloch-Pitts" or “Perceptron” type neuron [McCulloch 1943] with a threshold function. The operation of this neuron model can be summarized as follows: The weights of the connections (w_n) represent the strength of the synapse in a biological neuron. The total input to the neuron is calculated as the weighted sum of all inputs. The weighted sum is normalized using a function, commonly the sigmoid function. The sigmoid function produces an output in the range 0 to 1. The threshold is fixed at 0.5. If the output of the sigmoid function is greater than the threshold, then the neuron fires and produces a pulse (an output value of 1), vice versa no pulse (an output value of -1). Only the connection weights are trained when this type of neuron model is used.

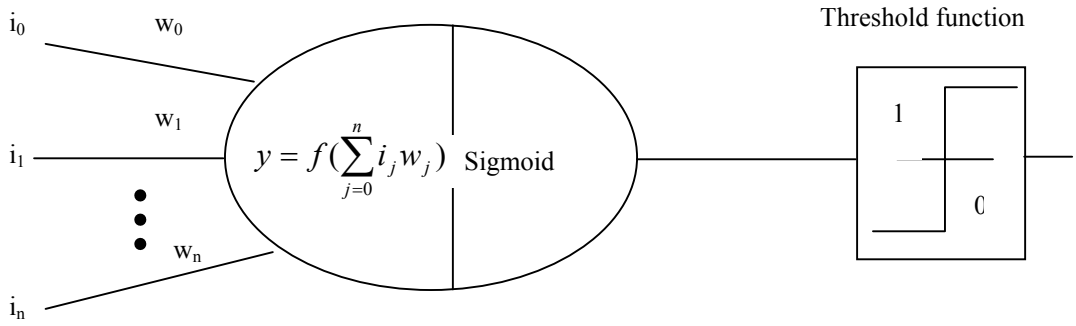
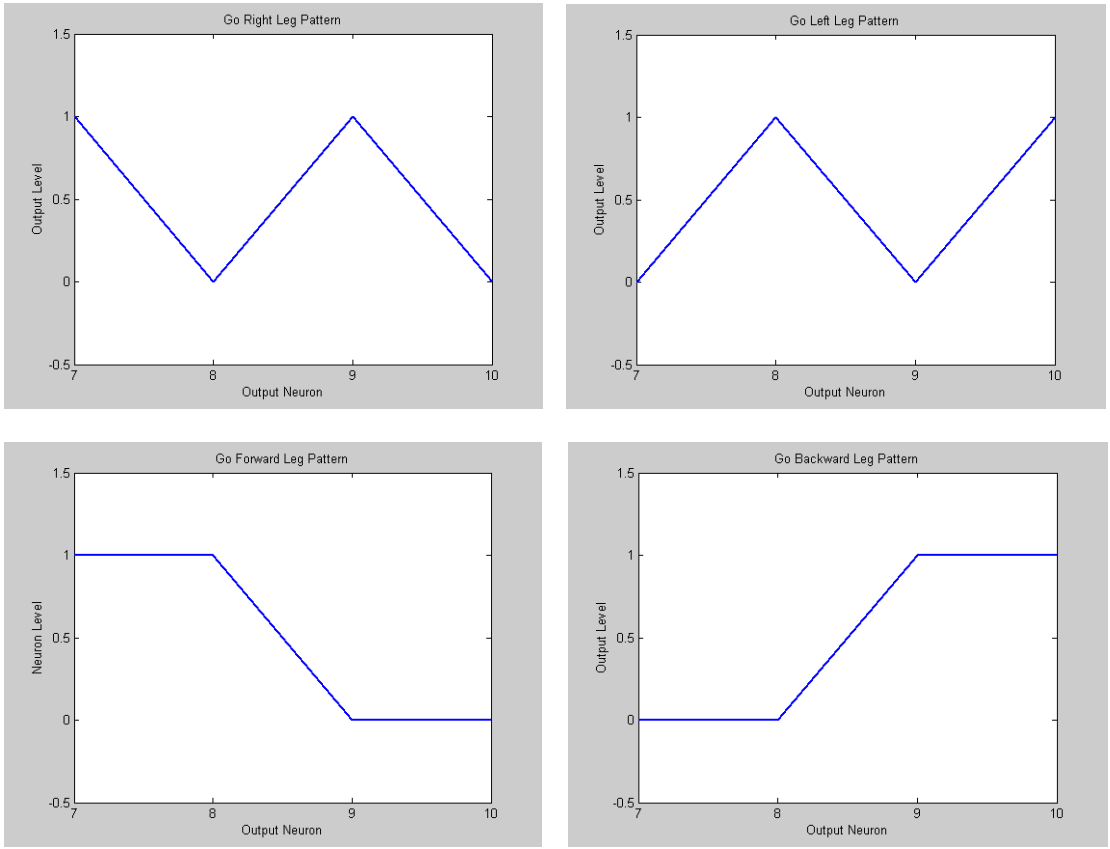


Figure 8-7 Modified Standard McCulloch-Pitts neuron with threshold function

The initial experiment concentrated on evolving a network to produce all the four different leg patterns in Stage 2. This is because previously we had difficulties in evolving a network to integrate the different leg patterns at this stage. The same technique illustrated in Figure 8-3 was initially used for this experiment. A network with two modules each with 4 and 3 neurons has been used to master the first two patterns. The network failed to produce the third pattern when a new module was added. It was very difficult to predict what was causing the problems. The technique illustrated in Figure 8-6 showed successful results when it was used for evolving

locomotive networks (see Section 6.4). This technique was then considered together with the neuron model shown in Figure 8-7.

A network with 4 modules, each with 4 neurons, was successfully evolved to produce all the 4 patterns. There were 4 neurons in each new module because 4 output neurons are required for each pattern. Figure 8-8 (a-d) shows the output leg patterns for the respective inputs for stage 2. Figure 8-8 (e) shows the fitness improvement as new modules are added to the previous modules. These results show that the neuron functionality is very important to network success.



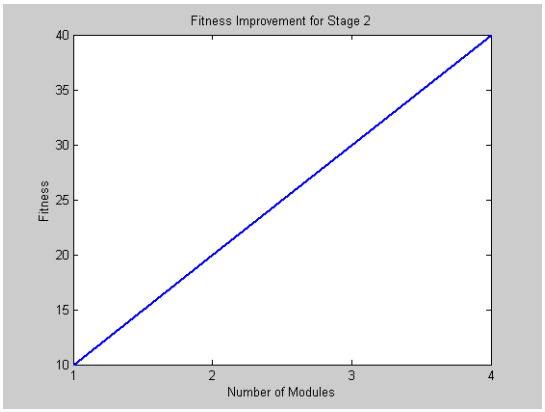
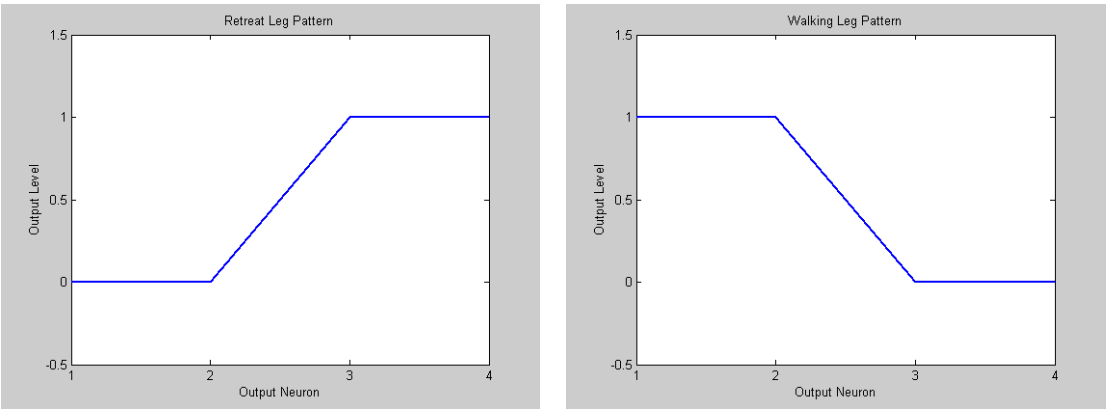


Figure 8-8 Output leg patterns for respective inputs for stage 2

Figure 8-9 shows the output leg pattern for stages 1 and 3. Networks have been grown in the sequence shown in Figure 8-2 to successfully integrate all the patterns presented. These results show that the technique of adding new modules in front of the previously evolved modules is very useful when the traditional approach fails.

Stage 1 Leg Pattern



Stage 3 Leg Pattern

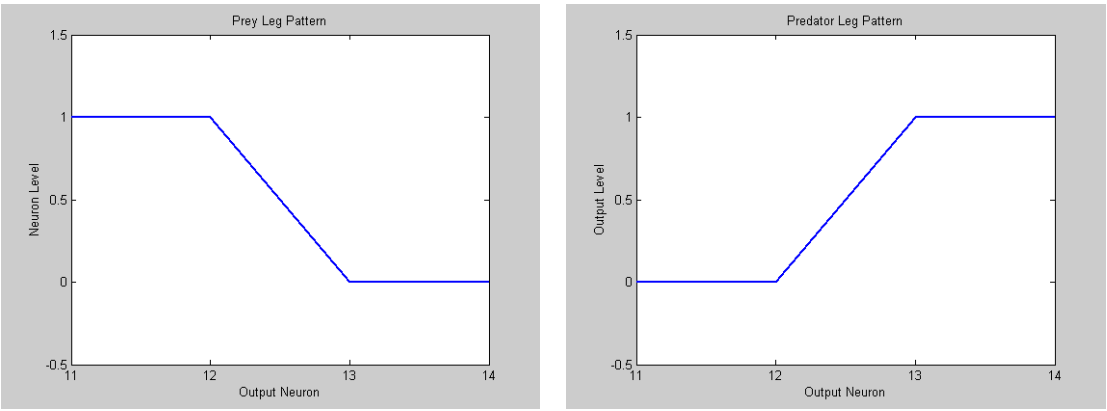


Figure 8-9 Output leg patterns for respective inputs for stage 1 and 3

[Reddipogu 2002] used a fixed neural network topology to mimic toad's vision system. The connection weights were trained until the network successfully learnt all the different input patterns. Reddipogu used Evolutionary Algorithms for Reinforcement Learning (EARL) to train the network. The learning algorithm took more than 13000 generations to master all the different visual patterns. It is hoped that this new evolutionary technique will be able to evolve a network with superior performance with lesser number of generations.

8.3 Integration of Locomotive with Vision Networks

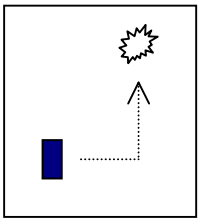
As explained in Section 7.2 of Chapter 7, if the robot is to become smarter, it must be introduced to an environment to which it can adapt. However, there seems little point in starting with a full scale (unconstrained) environment. There are simply too many (potentially conflicting) possibilities for it to contend with. The environment must be allowed to evolve along with the robot as previously described (that is, "deconstraining" the environment, an equivalent term to the process of "sensor and leg joint deconstraint" in the body plan).

An analysis of the sort of tasks of different complexities that simple animals can undertake indicates a possible forward direction. Table 1 below lists all the objects used to illustrate the progression.

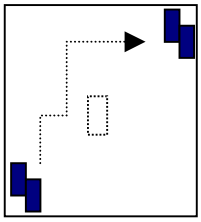
Objects	Explanation
	Light source
	Simplest Animals
	Simplest Invertebrates
	More complex invertebrates
	Path
	Obstacles
	Mates
	Food
	Predator

Table 1 Objects and its representation

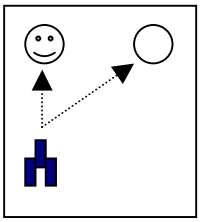
1) Simplest animals – Go towards light



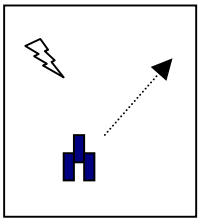
2) Simplest invertebrates – Recognise and avoid obstacles (plus skills of stage 1)



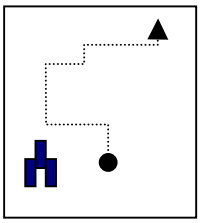
3) More complex invertebrates – Recognise food and mates (plus skills of stage 1 and 2)



4) Flatworms type animals – Recognise and flee from predators (plus skills of stage 1, 2 and 3)



5) Fish type animals – Path finding and learning (mission skills) (plus skills of stage 1, 2 , 3 and 4)



6) Reptile / Bird Skills – Manipulation e.g. Object which must be removed, etc
7) Higher Skills in mammals – tool skills, etc.

These different degrees of environmental interaction must be added one at a time in a thoughtful way to the robot. This may be accomplished through the addition of changing targets to the system in the changing environment or alternatively, by making the fitness function of the robot gradually more complex as it develops. The neural networks required to control the robot would be grown in similar ways to those previously described (see Section 5.6 of Chapter 5). It is clear with this technique that the neural network that has been evolved to interact with a particular environment will still be present even after a new network has been grown for another environment. This is useful because the previously evolved network could be re-used when the same environment re-occurs.

Returning to our previous work, separate networks exist for the locomotion and vision systems. The next stage was to grow networks to interface the first stage of the vision system to the previously evolved single degree of freedom bipedal walking and jumping gait. This problem is somewhat similar to the environment number 3 (recognise food and mates) illustrated above since there are two different possible conflicts to deal with. The other stages (Stage 2 and 3) of the vision system are not considered in the discussion since the interest is in proving that the technique can be used to integrate multiple different networks to form a system. The growth algorithm was unchanged from that described in Section 5.6. The network allowed different locomotive gaits to be triggered when different visual patterns were input. In this case, the bipedal walking gait will be triggered when the walking leg pattern is present at the vision system and vice-versa for gallop. The interface network can be said to be a 2×1 multiplexer because one from the two different input channels (bipedal walking and jumping gait network) will be selected to be the output depending on the input selection (coming from the vision system) at any time. Figure 8-10 shows the system configuration for the above case.

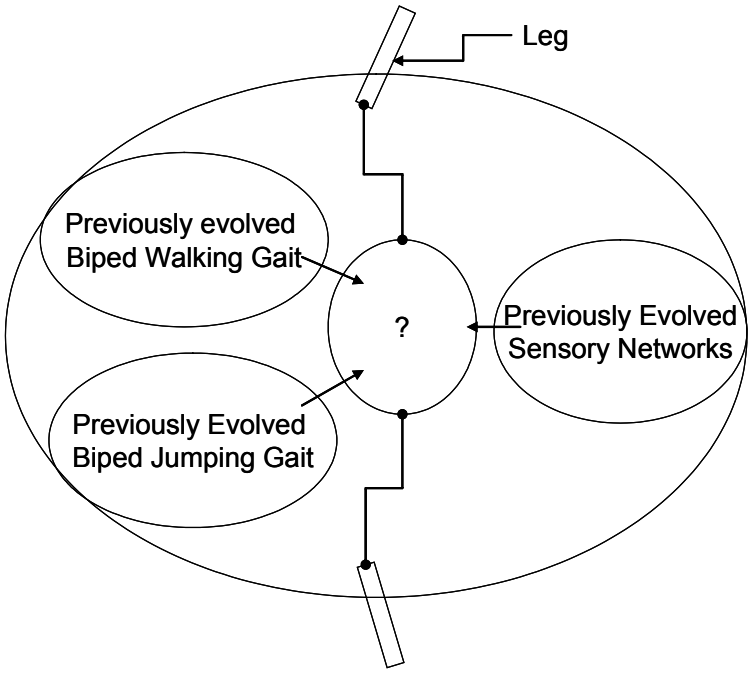


Figure 8-10 System configuration

The ES is not allowed to form connections from the new module to the previously evolved networks since this could modify the original behaviour of the networks. The Spike Accumulation and Delta Modulation neuron described in Chapter 5 was used to evolve the interface network. The fitness function for the interfacing network is a measure of the number of leg positions successfully relayed from the locomotive network to the output module for a triggering input. Each time an output neuron relayed the correct output to the actuator, the network was awarded a score of 1. Since there were 2 neurons in the output module, a maximum score of 2 can be awarded for a single timestep. A total score of 1000 could be achieved for simulation of 500 timesteps. In this case, the maximum fitness was 2000 since there were 2 locomotive gaits (walking and jumping).

A total of 5 modules with 2, 4, 3, 5 and 2 neurons was required to integrate the vision and the locomotive networks (refer to Chapter 6 for more explanation on the requirement for variable number of modules and neurons). Figure 8-11 shows the fitness improvement as each new module is added. Figure 8-12 shows the individual fitness for each gait as new modules are added. Table 2 gives a more detailed breakdown of the fitness in both Figure 8-11 and Figure 8-12 above.

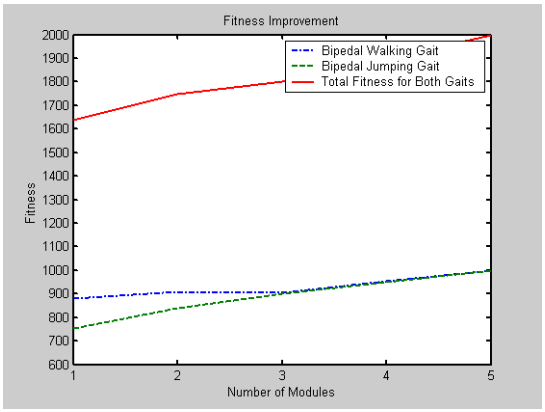


Figure 8-11 Fitness improvement for the system as new modules are added

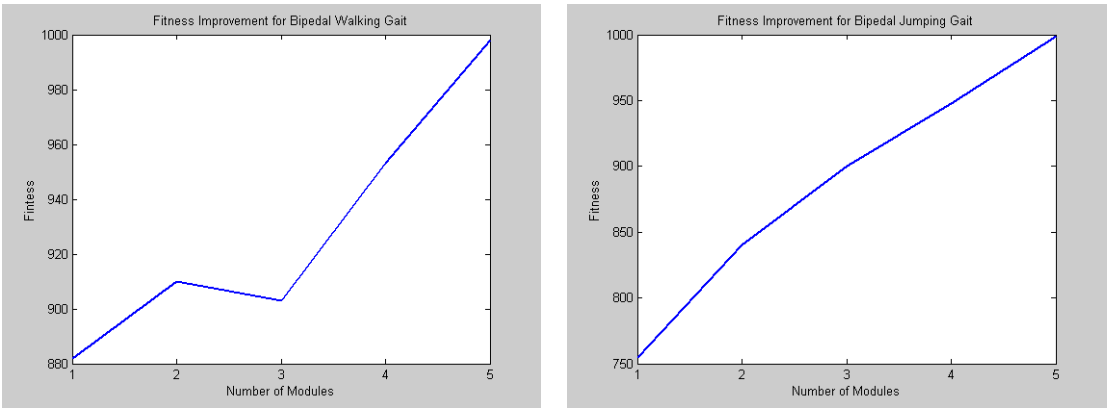


Figure 8-12 Fitness improvement for each gait as new modules are added

Module Number	Fitness for Bipedal Walking	Fitness for Bipedal Jumping	Total Fitness
1	882	755	1637
2	910(+28)	840(+85)	1750
3	903(-7)	900(+60)	1803
4	953(+50)	948(+48)	1901
5	998(+45)	999(+51)	1997

Table 2 Breakdown down of fitness scores for each gait for all the modules

From Table 2 it can be seen that there is a gradual increment in fitness for the bipedal jumping gait. The fitness dropped by 7 to 903 (Bipedal walking) and increased by 60 to 900 (Bipedal jumping) when the third module was introduced to the network. The probable reason for the decrement in the fitness that is, the ES could not manage to

evolve a set of weights and neuron parameters for both the gaits. There is also no requirement in the fitness function to make sure an increment in distance moved is achieved when a new module is introduced to the network. The gain entirely depends on previously evolved modules and the ES. It also can be seen that there is symmetry in the increment of individual fitness scores from the third module onwards. The number of generations required to achieve the fitness level was 1500 for each module. The number of generations needed is relatively large compared to the number of generations required for much simpler tasks presented in Chapter 6 and Chapter 7. One possible reason could be the neuron functionality. The interface network (as mentioned before could be a 2 × 1 multiplexer) has to integrate all three different networks. In electronics a multiplexer can be built using logic gates. If neurons in the network have to function like any of those logic gates, without any doubt the number of generation required to evolve a network would be fewer. Also the network was evolved to integrate several different objective functions. Evolving networks for multiple objective functions has proved a problem in past work [Lund 1994]. Modules with 2, 3, 4 and 5 neurons were trained for 5000 generations but the fitness level was not as good as that listed in the table. This shows that there is a minimum number of neurons required in order for the system to successfully evolve. Networks could also be grown to integrate Stages 2 and 3 of the vision system. In another experiment, the growth technique failed to evolve a network to control the robot's actuator and the vision system at the same time. This shows the success of the incremental growth technique in dealing with a complex problem incrementally.

8.4 Discussion

The system described above holds promise as a solution to the problem of the open ended evolution and development of neural networks and hence to the creation of large and complex multi-functional neural systems. Since the technique adopts a systems approach to the problem, it is particularly useful in robotics and similar problems where various unrelated subsystems need to be developed and integrated in an intelligent way.

Two important findings from the research were: That the neuron used should be as flexible as possible, as it is necessary to perform many difficult mappings in both the amplitude and time domains, especially when interfacing different modules of

previously grown networks and, secondly, that the evolutionary algorithms must be able to choose the network's connections as well as their weights.

The need for a flexible neuron with evolvable functionality has led the group to consider "universal" neuron models which can potentially evolve any continuous response [Capanni 2003]. This work is at an early stage but moves away from the idea of biologically feasible models and towards evolvable processors.

One possible disadvantage of the system is that, unlike a network designed by an optimal method, these networks may be wasteful of computing resources, in that they are potentially larger, although the current simulations do not show this with small networks. Another limitation, although, again, this has not been experienced in the simulations, may be apparent in systems where evolution or growth cannot go through an obvious sequence from simple to complex as part of its development. A related problem occurs in evolutionary timetabling and scheduling systems, in which a particular module must be placed early in the sequence to avoid a "bottle neck" occurring later – that is, a particular evolutionary path may preclude certain later developments.

It can be envisaged that, as systems become more complex, there will be a need to engineer changes (deconstraint) in the Fitness Function as development proceeds, choosing carefully the required steps to allow the system to evolve in the required manner. In the end, this process would stop body plan change, once full motor control had been achieved, and allow only the evolution of behaviour, in much the same way as the human brain continued to evolve in our early ancestors, even after our body plan was essentially fixed. The issue described above is a subject for future work. The final issue is whether some flexibility in previously evolved modules would make the evolution of later modules easier.

It is hoped that, once these issues have been resolved and integrated into the framework, new and interesting intelligent behaviours will emerge out of larger and more systems-orientated networks.

Chapter 9

Suggestions for Further Work

9.1 Introduction to Chapter

This chapter is divided into three different sections. It starts by describing further possible applications of the technique described in this research. The second section describes further work on neuron functionality and learning algorithms that could be integrated with the growth technique. Finally, the Chapter concludes by discussing several other areas of further work that may prove fruitful.

9.2 Other Applications of the Growth Method

Many researchers are currently using Evolutionary Algorithm (EAs) to evolve electronic circuits. John Koza of Stanford University is one of the pioneers in this field. He has succeeded in evolving electronic circuits for analogue filters, amplifiers and robot controllers [Scientific American 2003]. One such example is an evolved cubic generator using Genetic Programming (GP). This function generator was patented by the inventors [Cipriani 2000]. Koza found that the evolved circuits perform with better accuracy than the traditionally designed ones, even though the functionality of the evolved circuits is not fully understood.

In another example, Adrian Thompson [New Scientist 1997] showed that it is possible to evolve electronic circuits in Field Programmable Gate Arrays (FPGAs). He has succeeded in evolving large numbers of digital logic gates into a circuit which performs various timing tasks. The major advantage of evolving circuits in this way is that they can be reconfigured quickly into different topologies.

In the above work, the EAs arrange and re-arrange the components in the circuit until the fitness increases and the functionality is met. The results are limited, however, by the lack of modularity in the circuits and the fact that the search space grows very quickly as the circuit size increases. However, the application of the modular evolution technique described in this research should mitigate these problems by allowing the circuit to grow slowly in complexity in a modular fashion.

Just as in the neural network examples used in previous chapters to illustrate the technique, it is possible to start with a single simple circuit module and evolve this until it reaches a high fitness level. Again, it is possible to freeze the component values of the first module and to add a second. This new module will also undergo the same process. Modules of components may be added until the desired response is achieved. The technique may be particularly useful in the design of analogue filters or matching networks, which respond well to being built up in a piece-wise manner.

The same technique described above could also be used to evolve digital filters. Deciding on a suitable structure and coefficients are common problems in digital filter design. The algorithm could start with a population of modules containing delay lines, random coefficients and an output node. Standard Genetic Algorithm (GA) operators are applied for a number of generations until a good solution (module) is found. This solution is kept and further modules are added on top of the previously evolved network until the required specification is met. The cut and paste technique presented in Section 7.5 of Chapter 7 might be useful when cascaded sections are used to produce higher order filters. Copy and paste strategies may also be useful in the design of the analogue filters mentioned above.

Deducing a mathematical equation for a non linear curve is another difficult task in which the growth technique may be useful. In one approach, a dictionary of random mathematical variables and operators may be created. These variables and operators are then used to form a population of equations. The outputs of these equations are matched with the reference curve. The equations are evaluated based on the closeness of their match with the curve. The best equation is frozen and a new population of equations are created and added to the fixed equation until a good solution is found.

EAs have many applications in mechanical engineering as can be illustrated by the satellite dish support boom design devised by Keane [Keane 1996]. An important example of the application of the growth method in mechanical engineering is in designing aerodynamic structures.

Firstly, the parameters required for a basic aerodynamic structure may be optimized using a Evolutionary Algorithm (EA) until maximum performance for that structure is attained. Next, the parameters of the initial structure are fixed and another structure is joined to the first. The parameters for this structure are then optimized. This process is repeated for all the newly added structures. As more structures are added to the basic system, the performance (fitness) is measured for the whole system. In this case and many others the parameters of the newly added structure is always dependent on the previous structures in the system.

Another particular area of interest may be in the development of control systems for advanced prosthetic limbs where there is an obvious incremental path of deconstraint from one degree of freedom (all but one joint locked) to many degrees.

9.3 Investigations of Further Network Parameters

It was discovered during the research that the neuron functionality is important in determining the success of the growth method. There are several different types of neuron model available including: Radial Basis, Leaky Integration, Non-Linear and Spiking types. Despite this, most widely used ANNs operate on a variation of the McCulloch-Pitts perceptron. An Evolutionary System capable of developing a neuron model which can evolve any reasonable neuron function is therefore required. This would be able to mimic the biological neuron and also be capable of producing a wide range of other behaviors.

However, the biological neuron itself is not well understood by theorists.

In the biological network, action potentials can be transmitted to other neurons either electrically or chemically. Electrical transmission is not as common as chemical, and its role in nervous system is not yet fully understood [Letivan 1997]. In chemical transmission, the action potential causes a neurotransmitter to be released. This neurotransmitter binds to the membrane of the next neuron. Different neurotransmitters have different effects on a neuron. Not all the neurotransmitters are known and, of the ones that are, it is not known what all of their effects are [Ganong 1995]. The neurotransmitters can be said to be controlling the amount and type of

signal transmitted to the following neurons. Such complexity and the number of unknown variables is the reason why a "universal artificial neuron" of the type mentioned above would be useful.

The ANN research team at RGU has produced a new neuron model based on the idea that a neural unit should be flexible enough to fulfil any differential mathematical function required of it [Capanni 2003]. In his work, Capanni used power series to represent the activation of the neuron. Figure 9-1 shows the possible setup. x, y and z would be the three inputs and bn, cn and dn would be the respective weights.

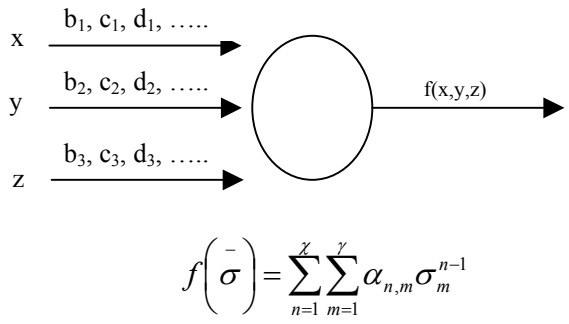


Figure 9-1 A polynomial neuron (Reproduced by permission of McMinn)

The most common artificial McCulloch-Pitts neuron is nothing more than a first order series. The above explanation is based on applying the power series neuron model to a simple pattern recognition system. The explanation can be expanded further by modeling the time response of the neuron. The resulting type of neuron is applicable to time dependent (locomotive) networks. The group is currently working on expanding the neuron model so that its time dependent response is also an evolvable time series.

The other aspect of the network, apart from neural functionality, which needs to be investigated, is learning. Of course, the networks used in this research do learn (optimize their weights) using an EA, but biological systems learn as their networks are operating (not off line, before operation starts). Online learning is therefore of topic for further research.

Traditional approaches to learning include Back Propagation (BP), Recurrent BP, Statistical Methods (such as Boltzman and Simulated Annealing), Reinforcement Learning, Competitive Learning, and Genetic Algorithms. The research group has developed an alternative to these which is described in [MacLeod 2002]. The paragraphs below are a brief explanation of this technique.

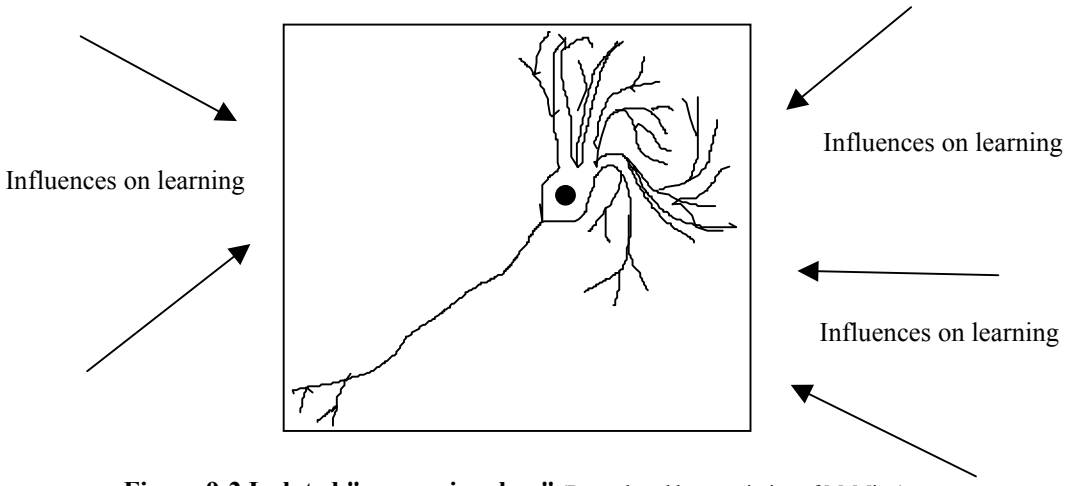


Figure 9-2 Isolated "neuron in a box" (Reproduced by permission of McMinn)

Consider a neuron in an isolated box as shown in Figure 9-2. Such a neuron can only be influenced by other neurons connected to it or the intercellular 'soup' that surrounds it. We can therefore start by listing all the possible parameters which could influence the network to learn.

Firstly, all the neurons in the brain are soaked in an intercellular fluid. Signals are transmitted chemically or electrically through this fluid-for example, by hormonal means. The result of this signal would affect the surrounding region and not an individual synapse.

Secondly, neurons may be affected by the activities of other neurons connected directly to them through their synapses.

Details of the possible parameters that could be used to model learning are outlined in the paper (Evolved and Devolved Action) in Appendix B.

This learning method is biologically realistic and highly dependent on the network topology; therefore, the learning algorithm is only suitable for networks whose topology is defined by an EA.

9.4 Other Ideas for Further Work

One of the aims of the research work beyond this project is to look at how intelligence might emerge from a complex network. Minsky [Minsky 1969] described a model that views the human brain as a collection of interacting modules called agents. In Minsky's model, each agent is capable of performing only a simple action, but intelligence emerges from their collective behavior.

It was emphasised in Chapter 5, that a major part of the modular evolution scheme is not the evolution of the neural network itself, but the evolution of the robot in terms of its body plan and the environment it is interacting with. Indeed, once the body has evolved to its fullest degree, then the system may continue to evolve the robot's mind by placing it in ever more complex environments. Therefore, if the robot is required to become 'smart', it needs to be introduced into a developing environment in which it can learn. Below is a list of progressively more complex environments for the robot to evolve in. The growth strategy would remain the same as used previously. It is hoped that intelligent behaviors might be observed as the network grows in terms of added modules.

Types of different environments:

- Add obstacles
- Add food/mate
- Add Predator
- Path Planning
- Add object which must be removed
- Tool skills

The issues described in the paragraphs above are subjects for future work. The final issue to be investigated is whether allowing some flexibility in previously evolved modules would make the evolution of later modules easier.

Chapter 10

Conclusions

10.1 Introduction to chapter

The purpose of this chapter is to summarise the project. In the first section, the objectives set at the beginning of the research are revisited in terms of what has been achieved. The following section describes the original contributions to the art of the research. The chapter concludes by commenting on the overall success of the project.

10.2 The project objectives revisited

The objectives, as originally stated at the start of the project, were:

1) Background reading and appropriate directed study
2) Literature search in field
3) Development of a basic Central Pattern Generator (CPG) network in a suitable format for Modular Evolution
4) The setting up of an experimental framework for the evolution of a sensory system
5) The application of the previous work to such a sensory system
6) The integration of these techniques into an overall algorithm which deals with the evolution of systems
7) Comparison with previously published results

Let us consider the objectives in terms of what has been achieved in the project.

1. Background reading and appropriate directed study

The initial work, in terms of background reading and study necessary to understand the project was undertaken at the beginning of the research. This included appropriate study as directed by the supervisors and the coding and testing of practical ANNs. Furthermore, McMinn's work [McMinn 2002] was examined and ANNs were evolved for the lower layers of the ANS (Reflex and Central Pattern Generator). Finally, the paper "Evolution and Devolved Action" (EDA) [MacLeod 2002] was studied and understood as it forms the basis of the research work.

2. Literature search in field

A literature review was undertaken during the first 6 months of the project and thereafter at a lower level all the way through until the end. The literature search has covered six different areas related to the research. The author has a high degree of confidence that all important work has been assessed. The outcome of the literature search is given in Chapter 4.

3. Development of a basic Central Pattern Generator (CPG) network in a suitable format for Modular Evolution

A framework was developed to investigate the evolution of Modular ANNs; this was successfully coded and implemented. The framework allows neural network modules to be added and deleted and also allows visualization of the growth pattern. Two different types of actuator for a quadruped robot body structure were used as a basis for the evolution of the ANS.

Modular Neural Networks were successfully evolved for control of locomotion in simulated Biped (walking and jumping) and Quadruped (pronk and trotting) robots. It was shown that modular evolution could evolve ANNs, adding more functionality (extra mechanical degrees of freedom) to their structure, by incrementally evolving single functions without retraining the whole network, provided that the functionality of the neuron is correct. This is an important result of the project.

The results from the Modular Evolution of ANNs for control of locomotion in simulated Biped and Quadruped robots were successful and different from the techniques developed by other researchers.

In the next part of the project, the growth techniques were explored more extensively, leading to some interesting findings. The outcomes are described below:

- In the growth method, connections to any of the previous neurons were allowed. However, when the technique was expanded to very large networks, the effect of connection area becomes a problem. The possible effects of adding a new network at the rear of previously evolved networks or in front of the initial module while preserving connections to the previous module only were investigated. It was found that adding the new network at the front end was more successful.
- It was found that allowing the algorithm to "copy and paste" previously evolved modules was often successful. For example, a biped was successfully evolved by taking two single leg sections and allowing an intermediate network to develop in between which interfaced the two sections.
- It was also found the flexibility of the system was such that it was capable of evolving two different gaits and switching between them.
- After evolving neural networks for a function (bipedal or quadrupedal single degree of freedom), an attempt was made to grow further networks on top of the previous function to modify the existing behavior (bipedal or quadrupedal with two degrees of freedom). This illustrates reusability of the existing networks.

Results to support all the points above are presented in Chapters 6 and 7 of this thesis.

4. **The setting up of an experimental framework for the evolution of a Sensory System**

The purpose of this experimental framework was to investigate the growth of networks designed for sensory input (upper layer of the ANS), using the example of vision systems. These networks are fundamentally different in nature from the locomotive nets developed in objective 3, above, and this proved the universality of the method. The vision framework starts with 1 by 1 grid (simple grid, 1 pixel) and this gradually evolves into a 5 by 5 (complex grid, 25 pixels) sensory system. The newer grids are added to the previously evolved vision system. Different ranges of patterns are available on the grid, from simple flight or fight responses, to the identification of obstacles in the field of view, as the grid evolves from simple to complex. Several different ANNs will produce the appropriate output pattern to control the robot's actuators based on the input from the vision grid.

5. **The application of the previous work to such a Sensory System**

The input sensor and the range of patterns to which it was exposed to, were allowed to grow from simple to complex. Modular neural networks were successfully evolved for different ranges of input patterns and responses.

6. **The Integration of these Techniques into an Overall Algorithm which Deals with the Evolution of Systems**

The issues of the evolution of systems, integrating both the locomotive and vision networks was considered. Both the vision and locomotion networks were successfully integrated by growing neural networks to map the different data sets into a single domain. Again, the ANNs have been grown using the method described previously. Finally, an Evolutionary Algorithm was developed for open-ended evolution of systems, without the need for human design or intervention.

7. Comparison with Previously Published Results from other Researchers

The results obtained were compared with appropriate related work. The comparison was presented along with the results in chapters (6, 7 and 8). Some of the results obtained from simulating the growth technique were not directly compared with other published results. This is because the growth technique used was different from those used by other researchers.

10.3 Novel Aspects of this Research

This project has several original aspects to it, all of which are a product of the work. These are:

- A unique and flexible method of evolving MANNs – the Direct Growth technique itself. The rigorous algorithm which controls the development and evolution of the network is presented in Appendix D of this thesis.
- Experiments showed that the neuron model used was very important in the success of the growth technique.
- Another significant finding was that the connections present from module to module play an important role. Instead of having a fully connected module, each neuron's connections and weights are determined by the evolutionary algorithm.
- It was shown that the growth technique could evolve ANNs for the extra added mechanical degrees of freedom to the robot's body structure. This result shows that the actuators and sensors can be added progressively and the ANNs which control them can be evolved incrementally, provided the neuron functionality of the neuron is correct.
- It was found that each module had to have a minimum number of neurons in order for the system to successfully evolve.
- It was also discovered that the success of the technique depends on where in the network new modules are added (permissible connections; at the end of the previously evolved network or before the previously evolved network) especially in large network.
- Several other applications of the growth technique are presented. These include the use of "Copy and Paste" method, networks which produce several gaits and can switch between them, and finally the integration of different networks to form a working system.

10.4 Summary of suggestions for further work

There are three main areas in which further work could be carried out to extend this research.

Firstly, in addition to evolving neural networks, the modular evolutionary algorithm has obvious applications in electronic engineering, mechanical engineering and mathematics. Refer to Section 9.2 of Chapter 9 for more information.

Secondly, the possible implementation of a more universal neuron which could potentially evolve more complex responses (Section 9.3) should lead to more evolvable networks. An on-line learning method would also be an important contribution to the research.

Finally (Section 9.4) the evolution of the mind for different behaviors, in much the same way as the human brain continued to evolve in our early ancestors, even after our body plan was essentially fixed, could be investigated.

10.5 Concluding Remarks

The project has been very successful in that all the initial objectives and more have been achieved. The growth technique is a powerful and useful method for evolving a modular system from simple to complex.

The author feels that the work is a useful contribution to the field of evolutionary techniques, allowing standard EAs like Genetic Algorithms to overcome some of the well known obstacles to their usefulness in complex systems.

It is hoped, that once the final issues (particularly neural functionality and learning) have been resolved and integrated into the growth technique framework, new and interesting intelligent behaviors will emerge out of larger and more systems-orientated networks.

References

Chapter 1

Minsky, M., and Papert, S., 1969, Perceptrons, MIT Press, Cambridge, MA.

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" Neuroinformatics and Neurocomputers, pp. 62-69, 1995.

Bonabeau, E., Dorigo, M. and Theraulaz, G., 1999, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press.

Warwick, K., Kelly, I.D., and Keating, D.A., 1997, "Mutual Learning by Autonomous Mobile Robots," in Proc. First workshop on Teleoperation and Robotics, Applications in Science and Arts, pp. 103-115.

McMinn, D., 2002, Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous System, PhD Thesis, The Robert Gordon University.

Arbib, M.A., 1995, Handbook of Brain Theory and Neural Networks, The MIT Press.

Happel, B.L.M. and Murre, J.M.J., 1994, The Design and Evolution of Modular Neural Network Architectures. Neural Networks, vol. 7, pp. 985 – 1004.

MacLeod, C., McMinn, D., Reddipogu, A., and Capanni, N., 2002, Evolution by Devolved Action: Towards the Evolution of Systems, in Appendix B of McMinn, D., Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous System, PhD Thesis, The Robert Gordon University, Aberdeen, UK.

Chapter 2

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

McMinn, D., 2002, Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous System, PhD Thesis, The Robert Gordon University.

Reddipogu, A., Maxwell, G., and MacLeod, C., 2002, "An Innovative Neural Network Based on The Toad's Visual System" in Proc. of ACIVS 2002 (Advanced Concepts for Intelligent Vision Systems), Ghent, Belgium.

Ewert, J. P.,1987, "Neuroethology of Releasing Mechanisms: Prey Catching in Toads," Behavioural and Brain Sciences, Vol 10:3, pp. 337-367.

Chapter 3

Schaffer, J.D., Whitley, D.L., Eshelman, J., 1992, Combinations of Genetic Algorithms and Neural Network: A survey of the state of the Art, COGANN-92(conference), IEEE Computer Society Press.

McMinn, 2002, Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous System, PhD Thesis, The Robert Gordon University.

Bentley, P.J., 2001, "Digital Biology", The creation of life inside computers and how it will affect us, Published by REVIEW, ISBN 0 7472 6654 9, pp. 209-217.

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

MacLeod, C., and Maxwell, G., 1997 "Using Embryology as an Alternative to Genetic Algorithms for Designing Artificial Neural Networks," Proc. ICANNGA, pp.361-366.

Gould, S. J., 2000, Wonderful Life. (London, England: Vintage).

Restak, R.M., 1979, The brain: The last frontier, Warner Books, pp 50-52.

Fritzsch, B., 1998, "Evolution of the Ancestral Vertebrate Brain", in The Handbook of Brain Theory and Neural Networks, M. A. Arbib, Ed, MIT Press, pp.373-376.

Chapter 4

Yao, X., 1997, A New Evolutionary System for Evolving Artificial Neural Networks. IEEE Transactions on Neural Networks, vol. 8, no. 3, pp. 694 – 713.

Happel, B.L.M. and Murre, J.M.J., 1994, The Design and Evolution of Modular Neural Network Architectures. Neural Networks, vol. 7, pp. 985 – 1004.

Holland, J. H., 1992, Adaption in Natural and Artificial Systems (1st MIT Press Edn.)The MIT Press, Cambridge, MA.

Goldberg, D. E., 1989, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA.

Back, T., Hoffmeister, F., and Schwefel, H.-P., 1991, A survey of evolution strategies, Proc. Of the Fourth Int'l Conf. on Genetic Algorithms, pp. 2-9. Morgan Kaufmann, San Mateo, CA.

Fogel, L. J., Owens, A. J., and Walsh, M. J., 1966, Artificial Intelligence Through Simulated Evolution, John Wiley & Sons, NY.

Sutton, R. S., 1986,Two problems with backpropagation and other steepest-descent learning procedures for network. In Proc. Of 8th Annual Conf. of the Cognitive Science Society, pp 823-831. Lawrence Erlbaum Associates, Hillsdale, NJ.

Bremermann, H. J., 1968, Numerical Optimization Procedures Derived from Biological Evolution Processes, In Cybernetic Problems in Bionics (1996 conference), Gordon and Breach, NY, 1968, pp. 543-562.

Fogel, D. B. and Angeline, P. J., 1995, A Review of Efforts Combining Neural Networks and Evolutionary Computation, In SPIE vol.2492, pp. 586 – 589.

Koza, J. R. and Rice, J. P., 1991, Genetic generation of both the weights and architecture for a neural network, In Proc. Of 1991 IEEE International Joint Conference on Neural Networks (IJCNN'91 Seattle), vol. 2, pp. 397-404. IEEE Press, NY.

Miller, G. F., Todd, P. M., and Hedge, S. U., 1989, Designing neural networks using genetic algorithms, Proc. Of the Third Int'l Conf. on Genetic Algorithms and Their Applications, pp. 379-384. Morgan Kaufmann, San Mateo, CA.

Kitano, H., 1990, Designing neural networks using genetic algorithms with graph generation system, *Complex Systems*, vol. 4, pp. 461-476.

Harp, S. A., Samad, T., and Guha, A., 1989, Towards the genetic synthesis of neural networks, Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications, pp. 360-369. Morgan Kaufmann, San Mateo, CA.

Oliker, S., Furst, M., and Maimon, O., 1991, Optimization and training of feedforward neural networks by genetic algorithms, In Proc. of the Second IEE International Conference on Artificial Neural Networks, pp. 39-43. IEE Press, London, UK.

Alba, E., Aldana, J. F., and Troya, J. M., 1993, Fully Automatic ANN Design: A Genetic Approach. In Proc. of Int'l Workshop on Artificial Neural Networks (IWANN' 93), vol. 686, pp. 399-404. Springer-Verlag.

Vonk, E., Jain, L. C., Johnson, R. P., 1997, Automatic Generation of Neural Network Architecture using Evolutionary Computation, Advances in Fuzzy Systems – Applications and Theory vol 14, ISBN 9810231067, World Scientific Publishing Co. Pte. Ltd.

Harp, S. A., Samad, T., and Guha, A., 1990, Designing application-specific neural networks using the genetic algorithm, In D. S. Touretzky, editor, Advances in Neural Information Processing Systems 2, pp. 447-454. Morgan Kaufmann, San Mateo, CA.

Alba, E., Aldana, J. F., and Troya, J. M., 1993, "Genetic Algorithms as Heuristics for Optimizing ANN design", In International Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93), pp.683-689, Innsbruck, Austria.

Doi, Y., 1988, Morphogenesis of Life Forms, Saiensu-sha.

Gruau, F., 1992, Genetic Synthesis of Boolean Neural Networks with a Cell Rewriting Developmental Process, In IEEE International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92), pp.55-74, Baltimore.

Gruau, F., 1994, "Genetic Microprogramming of Neural Networks", In Advances in Genetic Programming, MIT Press.

Boers, E.J.W. and Kuiper, H., 1992, Biological Metaphors and the Design of Modular Artificial Neural Networks, In Technical Report, Departments of Computer Science and Experimental and Theoretical Psychology, Leiden University, The Netherlands.

Merrill, J. W. L. and Port, F., 1991, Fractally Configure Neural Networks, Neural Networks, vol. 4, pp. 53-60.

Carpenter, G. A. & Grossberg, S., 1986, Neural Dynamics of Category Learning and Recognition. Brain Structure, Learning and Memory (AAA Symposium)

Carpenter, G. A. & Grossberg, S., 1987, A Massively Parallel Structure for a Self Organised Neural Pattern Recognition Machine. Computer vision, graphics and image processing 37, pp.54-115.

MacLeod, C., and Maxwell, G. M., 2001, Incremental Evolution in ANNs: Neural Nets which Grow, Artificial Intelligence Review 16, pp. 201-224, Netherlands.

Ash, T., 1989, Dynamics Node Creation in Backpropagation Networks, Connection Science 1, pp. 365-375.

Chakraborty, G., 1995, A Growing Network which Optimises between Undertraining and Overtraining, IEEE Conference on Neural Networks 2, pp. 1116-1120.

Vinod, V. V. & Ghoso, S., 1996, Growing Non-uniform Feed-forward Networks for Continous Mappings, Neurocomputing 10, pp. 444–452.

Ferran, E & Perazzo, R., 1991, Asymptotic Inferential Capabilities of Feed-forward Neural Networks, In *Europhysics letters* 14(2), pp. 175-180.

Mozer , M. C. and Smolensky, P., 1989, Skeletonization: a technique for trimming the fat from a network via relevance assessment, Connection Science, 1, pp. 3-26.

Anderle, M., Schweng, K, Kurten, K. E. & Kratky, K. W., 1995, Pattern Specific Neural Network Design, In Journal of statistical physics 81 (3/4), pp. 843-849.

Yao, X. and Liu, Y., 1996, Ensemble structure of evolutionary artificial neural networks, In Proc. of the 1996 IEEE Int'l Conf. on Evolutionary Computation (ICEC'96), pp. 659-664, Nagoya, Japan.

Gruau, F., 1992, Cellular encoding of genetic neural networks.In Technical Report 92-91, Laboratoire de l'Informatique du Parallelisme, ENS Lyon.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J and Hinton, G. E., 1991a, Adaptive mixtures of local experts, In Neural Computation, vol.3, pp. 79-87.

Jacobs, R. A., Jordan, M. I., and Barto, A. G., 1991b, Task decomposition through competition in a modular connectionist architecture: the what and where vision task, In Cognitive Science, vol.15, pp.219-250.

Battiti, R. and Colla, A. M., 1994, Democracy in neural nets: voting schemes for classification, In Neural Networks, vol.7, pp. 691-707.

Hansen, L. K. and Salamon, P., 1990, Neural network ensembles, In IEEE Trans. On Pattern Analysis and Machine Intelligence, vol.12, no.10, pp.993-1001.

Fukushima, K., 1980, Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybernet. 36, pp. 193-202.

Fukushima, K., 1987, Neural netwok model for selective attention in visual pattern recognition and associative recall, Appl. Optics 26, pp. 193-202.

Fukushima, K., 1988, Neocognitron: a hierarchical neural network capable of visual pattern recognition, Neural Networks 1, pp. 119-130.

Fukushima, K., Miyake, S., and Ito, T., 1993, Neocognitron: a neural network model for a mechanism of visual pattern recognition, IEEE Trans. System Man Cybernet, SCM-13, pp. 826-834.

Rueckl, J. G., Cave, K. R., and Kosslyn, S. M., 1989, Why are ‘What’ and ‘Where’ processes by separate cortial visual system? A computational investigation, J. Cognitive Neurosci. 1, pp.171-186.

Schmidt, A. and Zuhair, B., 1998, Modularity, A concept for New Neural Network Architectures, IASTED International Conference, Computer Science and Applications (CSA), Irbid, Jordan.

Yang, M. C., Mehrotra, K., Mohan, C. K., and Ranka, S., 1992, Partial shape matching with attributed strings and neural networks. In Proceedings of the Conference on Artificial Neural Networks in Engineering (ANNIE), pp. 523-528.

Jordan, M. I., 1994, Modular and hierarchical learning systems, In press: M. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks, Cambridge, MA, MIT Press.

Chu, K. L. and Mandava, R., 2000, Growing Multi-Experts Network, Tencon Proceedings, pp. 472-478, Malaysia.

Perez, C. A., Galdames, P.A, and Holzman, C.A., 1998, Improvements on handwritten digit recognition by cooperation of modular neural networks, IEEE International Conference on Systems, Man, and Cybernetics vol 5, pp. 4172-4177.

McMinn, D., 2002, Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous System, PhD Thesis, The Robert Gordon University.

Gomi, T., and Ide, K., 1998, Evolution of gaits of a legged robot, IEEE International Conference on Fuzzy Systems Proceedings, IEEE World Congress on Computational Intelligence, pp. 156-164.

Hornby, G.S., Fujita, M., Takamura, S., Yamamoto, T., Hanagata, O., 1999, Autonomous evolution of gaits with the Sony Quadruped Robot, Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1297-1304.

Takamura, S., Hornby, G., Yamamoto, T., Yokono, J., Fujita, M., 2000, Evolution of dynamic gaits for a robot, International Conference on Consumer Electronics, pp. 192-193.

Mezard, M., and Nadal, J.P., 1989, Learning in feedforward layered networks: The Tiling algorithm, In Journal of Physics A: Math. Gen., pp.2191-2203.

Gallant, S. I., 1990, Perceptron-based learning algorithms, In *IEEE Transactions on Neural Networks*, 1(2), pp.179-191.

Frean, M., 1990, The Upstart algorithm: A method for constructing and training feed-forward neural networks, In Neural Computation, vol. 2, pp.198-209.

Fahlam, S. E. and Lebiere, C., 1989, The cascade-correlation learning architecture, Advances in Neural Information Processing System 2. Proceedings of the 1989 Conference, pp. 524-532, San Mateo, CA.

Reilly, D. L., Cooper, L. N., and Elbaum, C., 1982, A neural model for category learning, In Biological Cybernetics, vol. 45, pp. 35-41.

Platt, J., 1991, A resource-allocating network for function interpolation, In Neural Computation, 3(2), pp.213-225.

Gallant, S. I., 1986, Three constructive algorithms for network learning, In Proceedings, 8th Annual Conference of the Cognitive Science, pp. 652-660.

Haris, de H., 1993, Incremental Evolution of Neural Nets, Genetic Programming in Incremental Steps, World Congress on Neural Networks, pp. 447-450.

Fritzke, B., 1994, Fast Learning with incremental RBF Networks, Neural Processing Letters, 1(1), pp.2-5.

Fritzke, B., 1995, A growing neural gas network learns topologies, In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), Advances in Neural Information Processing Systems (Vol. 7), Cambridge MA, USA: MIT Press.

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

Gruau, F. and Withley, D., 1993, Adding learning to the cellular development of neural networks: Evolution and the Balwin effect, In Evolutionary Computation, 1(3), pp. 213-233.

Gruau, F., 1994, Automatic definition of modular neural networks, In Adaptive Behaviour, 3(2), pp.151-184.

Nolfi, S., and Parisi, D., 1991, Growing neural networks, T.R. PCIA-91-95. Institute of Psychology. Rome.

Nolfi, S., Miglino, O., and Parisi, D., 1994, Phenotypic plasticity in evolving neural networks. Proceedings of the PerAc'94 Conference. IEEE Computer Society Press.

Vaario, J., 1993, An emergent modelling method for artificial neural networks, PhD thesis, University of Tokyo.

Vaario, J., 1994, From evolutionary computation to computational evolution, In Informatics, vol. 18, pp. 417-434.

Lindermayer, A., 1968, Mathematical models for cellular interaction in development, parts I, II, Journal of Theoretical Biology, 18, pp. 280-299 and 300-315.

Cangelosi, A., Parisi, D., and Nolfi, S., 1995, Cell division and migration in a 'genotype' for neural networks, In Network: computation in neural systems.

Dellaert, F. and Beer, R., 1994a, Toward an evolvable model of development for autonomous agent synthesis, Proceedings of the Fourth International Workshop on Artificial Life, The MIT Press/Bradford Books, Cambridge, MA.

Dellaert, F. and Beer, R., 1994b, Co-evolving body and brain in autonomous agents using a developmental model, In Technical Report CES-94-16, Dpt of Computer Engineering and science Case Western Reserve University, Cleveland, OH.

Lee, W.P., 2003, Evolving robot brains and bodies together: an experimental investigation, Journal of the Chinese Institute of Engineers, pp. 125-132, Taiwan.

Sims, K., 1994a, Evolving virtual creatures, In Computer Graphics Proceedings, Annual Conference Series, pp. 15-23.

Sims, K., 1994b, Evolving 3D morphology and behaviour by competition, Proceedings of the Fourth International Workshop on Artificial Life. The MIT Press/Bradford Books, Cambridge, MA.

Liu, Y., and Yao, X., 1998, Evolving Modular Neural Networks which Generalise well, Proceedings of the IEEE International Conference on Artificial Life and Robotics, AROBIII 98, vol. 2, pp. 736 – 739.

Chapter 5

McMinn , D., 2002a, "Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems, PhD Thesis, The Robert Gordon University, Aberdeen, UK.

Brodal, P., 1992, The Central Nervous System: Structure and Function, Oxford, pp.229.

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

Muthuraman, S., Maxwell, G. and MacLeod, C., 2003a, "The Evolution of Modular Evolutionary Networks for Quadrupedal Locomotion," in Proc. International Conference on Artificial Intelligence and Soft Computing (ASC), pp. 268-273.

Shigematsu, Y., Ichikawa, M. and Matsumoto, G., 1996, "Reconstruction studies on brain computing with the neural network engineering," in Perception, memory and emotion: frontiers in neuroscience, Elsevier, pp. 581-599.

Schwefel, H-P., 1995, "Evolution and Optimum Seeking",Wiley, New York.

Recenberg, I., 1973, "Evolution Strategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution.", Frommann-Holzboog.

Miller, G. F., Todd, P. M., and Hegde, S. U., 1989, "Designing Neural Networks using Genetic Algorithms", Proc. of the Third Int'l Conf. On Genetic Algorithms and Their Applications, pages 379-384, San Mateo, CA.

Yao, X., 1993, Evolutionary Artificial Neural Networks, International Journal of Neural Systems, Vol. 4, No.3, pages 203-222.

McMinn, D., Maxwell, G., and MacLeod, C., 2002b, "Evolutionary Artificial Neural Networks for Quadruped Locomotion," Proc. ICANN, pp. 789-794.

Muthuraman, S., Maxwell, G., MacLeod, C., 2003b, "The Evolution of Modular Artificial Neural Networks for Legged Robot Control," in Proc. ICONIP/ICANN, pp.488-495.

Sims, K., 1994, Evolving 3D morphology and behaviour by competition, Proceedings of the Fourth International Workshop on Artificial Life. The MIT Press/Bradford Books, Cambridge, MA.

MacLeod, C., McMinn, D., Reddipogu, A. and Capanni, N., 2002, "Evolution by Devolved Action: Towards the Evolution of Systems" in: Appendix B of D. McMinn, Using Evolutionary Artificial Neural Networks to Design Hierachial Animat Nervous Systems, PhD Thesis, The Robert Gordon University, Aberdeen, UK.

Chapter 6

McMinn, D., Maxwell, G., and MacLeod, C., 2000, "An Evolutionary Artificial Nervous System for Animat Locomotion," Proc. of the International Conference on Engineering Applications of Neural Networks, pp. 170-176.

McMinn, D., Maxwell, G., and MacLeod, C., 2002a, "Evolutionary Artificial Neural Networks for Quadruped Locomotion," Proc. ICANN, pp. 789-794.

McMinn , D., 2002b, "Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems, PhD Thesis, The Robert Gordon University, Aberdeen, UK.

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

Chapter 7

Shigematsu, Y., Ichikawa, M. and Matsumoto, G., 1996, "Reconstruction studies on brain computing with the neural network engineering," in Perception, memory and emotion: frontiers in neuroscience, Elsevier, pp. 581-599.

McMinn , D., 2002, "Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems, PhD Thesis, The Robert Gordon University, Aberdeen, UK.

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

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.

Chapter 8

Ewert, J. P., 1985, "Concepts in vertebrate neuroethology," Animal Behaviour, Vol 33, pp 1-29.

Ewert, J. P.,1987, "Neuroethology of Releasing Mechanisms: Prey Catching in Toads," Behavioural and Brain Sciences, Vol 10:3, pp. 337-367.

Arbib, M. A., and Liaw, J.S., 1995 "Sensorimotor transformations in the worlds of frogs and robots," Artificial Intelligence, Vol 72, pp. 53-79.

Reddipogu, A., Maxwell, G., and MacLeod, C., 2002, "An Innovative Neural Network Based on The Toad's Visual System" in Proc. of ACIVS 2002 (Advanced Concepts for Intelligent Vision Systems), Ghent, Belgium.

Shigematsu, Y., Ichikawa, M. and Matsumoto, G., 1996, "Reconstruction studies on brain computing with the neural network engineering," in Perception, memory and emotion: frontiers in neuroscience, Elsevier, pp. 581-599.

McCulloch, W., and Pitts, W., 1943, "A logical calculus of ideas immanent in nervous activity". Bulletin of Mathematical Biophysics, 5:115-113.

Lund, H. H. and Parisi, D., 1994, "Simulations with an Evolvable Fitness Formula", Tech. Rep. 94-01, Institute of Psychology, CNR.

Capanni. N., MacLeod. C., and Maxwell. G., 2003, "An approach to Evolvable Neural Functionality", International Conference on Artificial Neural Networks and Neural Information Processing, pp.220-223.

Chapter 9

Scientific American, February 2003, pp.40-47.

Cipriani, S. and Takeshian, Anthony. A., 2000, Conexant Systems, Newport Beach, Calif.

New Scientist, November 1997, pp. 30.

Keane, A. J. and Brown, S. M., 1996, The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques, in Proceedings of the conference on Adaptive Computing in Engineering Design and Control, pp.107-113.

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

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

Capanni. N., MacLeod. C., and Maxwell. G., 2003, An approach to Evolvable Neural Functionality, International Conference on Artificial Neural Networks and Neural Information Processing, pp.220-223.

MacLeod, C., McMinn, D., Reddipogu, A., and Capanni, N., 2002, Evolution by Devolved Action: Towards the Evolution of Systems, in Appendix B of McMinn, D., Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous System, PhD Thesis, The Robert Gordon University, Aberdeen, UK.

Minsky, M., and Papert, S., 1969, Perceptrons, MIT Press, Cambridge, MA.

Chapter 10

McMinn, D., 2002, Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous System, PhD Thesis, The Robert Gordon University.

MacLeod, C., McMinn, D., Reddipogu, A., and Capanni, N., 2002, Evolution by Devolved Action: Towards the Evolution of Systems, in Appendix B of McMinn, D., Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous System, PhD Thesis, The Robert Gordon University, Aberdeen, UK.

Appendix A

Papers produced during research.

Papers produced during the research program. These include published papers and papers awaiting publication.

Paper 1

The Evolution of Modular Artificial Neural Networks for Legged Robot Control

This paper describes the application of the evolutionary technique to control single degree of freedom legs of a robot. In the later body configuration, a second degree of freedom was added to the initial body plan. Initial results were presented to illustrate the successful operation of the technique in evolving networks to produce a bipedal walking gait.

Paper 2

The Development of Modular Evolutionary Networks for Quadrupedal Locomotion

In this paper the biological justification for the evolutionary technique was presented. Results were also presented which demonstrate the operation of the approach in the development of a quadrupedal gait for a simulated robot.

Paper 3

Unconstrained Incremental Evolution of Neural Networks for Robot Control

This paper outlines the evolutionary technique in more detail. Results were presented showing the technique in operation. There is also a discussion of other applications of the technique and related issues.

(Currently under review)

Appendix B

Evolution and Devolved Action: towards the evolution of systems

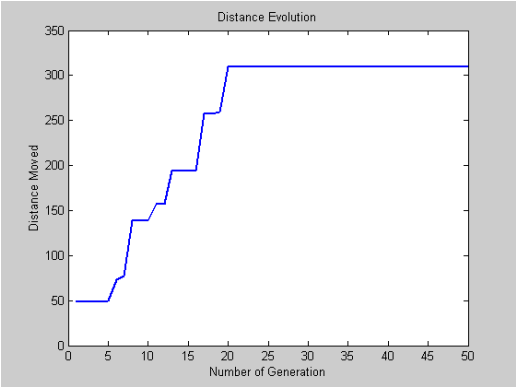
“Evolution and Devolved Action” examines the limitations of present Artificial Evolutionary Algorithms from a biological perspective and looks at how these limitations might be overcome. This report formed the basis for the research.

Appendix C

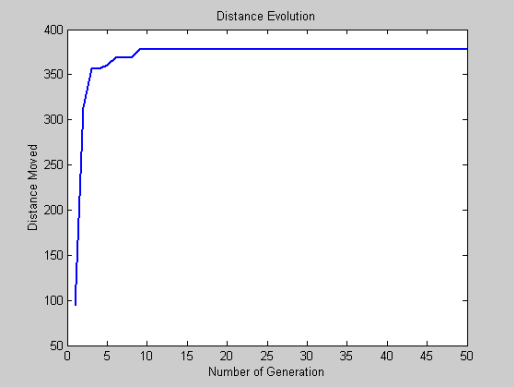
Further Results

Section C.1
Reference in Page 116

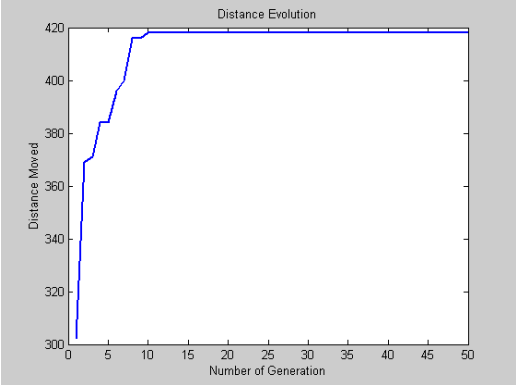
Note: x:y:z, where x,y,z… refers to number of neurons in a module



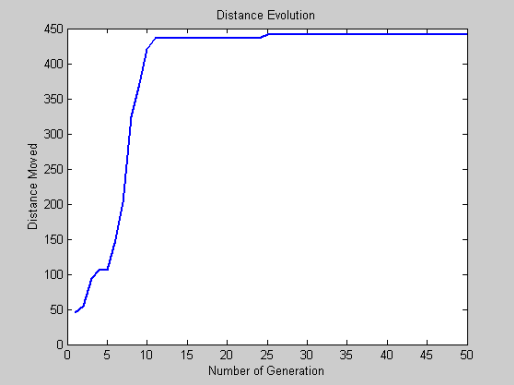
a) 1 module with 5 neurons



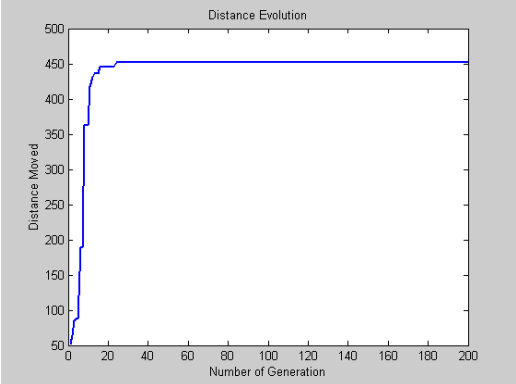
b) 2 modules, 5:3



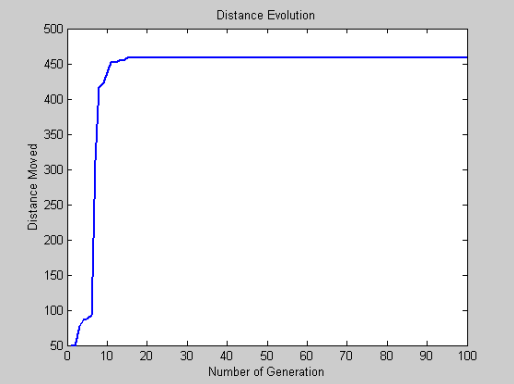
c) 3 modules, 5:3:2



d) 4 modules, 5:3:2:4



e) 5 modules, 5:3:2:4:4

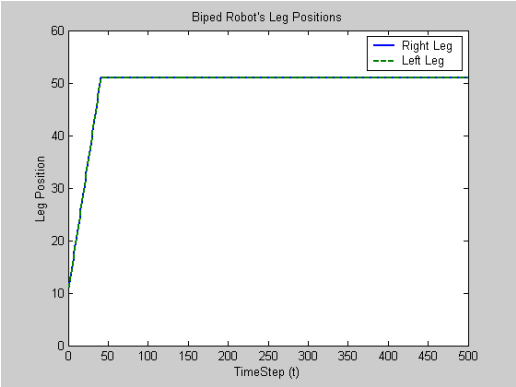


f) 6 modules, 5:3:2:4:4:5

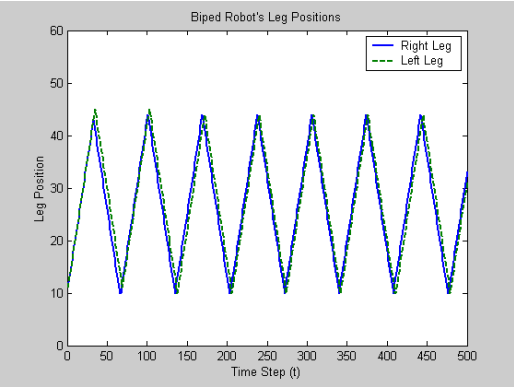
Section C.2
Reference in Page 99

David McMinn's Actuator Model (Bipedal Jumping Gait)

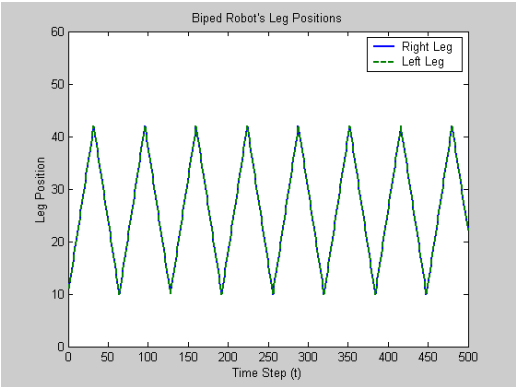
Note: x:y:z, where x,y,z… refers to number of neurons in a module



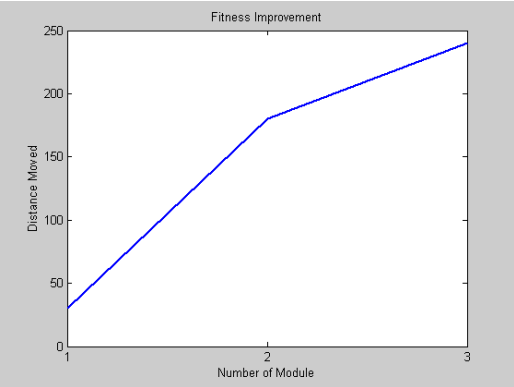
a) 1 module with 2 neurons



b) 2 modules, 2:2



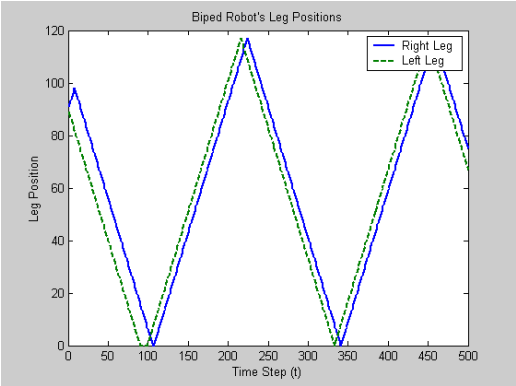
c) 3 modules, 2:2:2



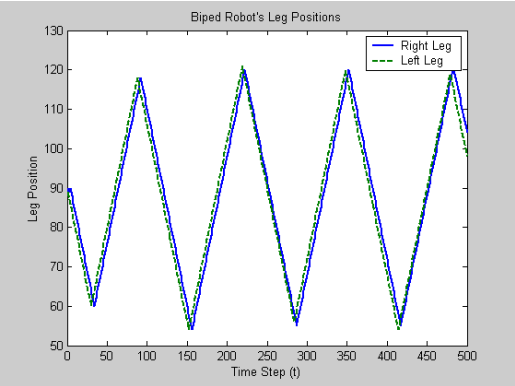
d) Fitness Improvement

New Actuator Model (Bipedal Jumping Gait)

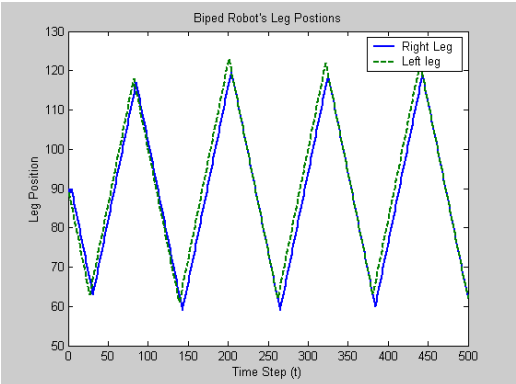
Note: x:y:z, where x,y,z… refers to number of neurons in a module



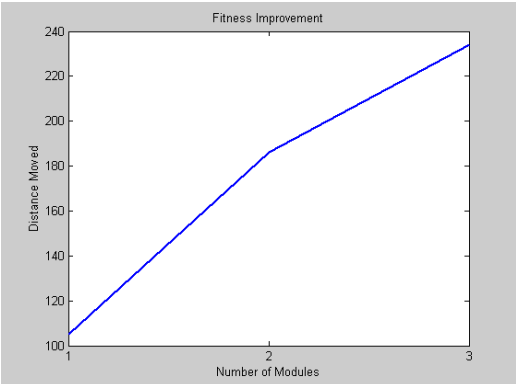
a) 1 module with 2 neurons



b) 2 modules, 2:2



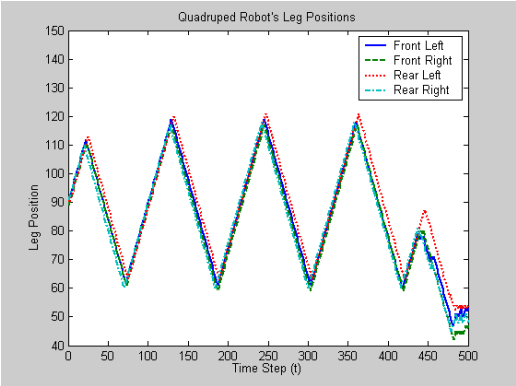
c) 3 modules, 2:2:2



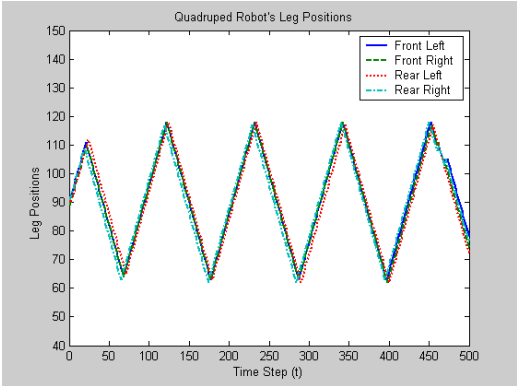
d) Fitness Improvement

New Actuator Model (Quadruped Pronk Gait)

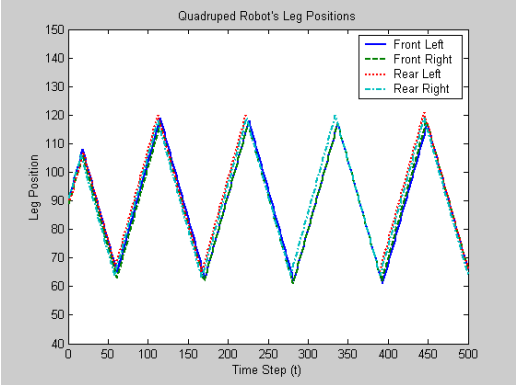
Note: x:y:z, where x,y,z… refers to number of neurons in a module



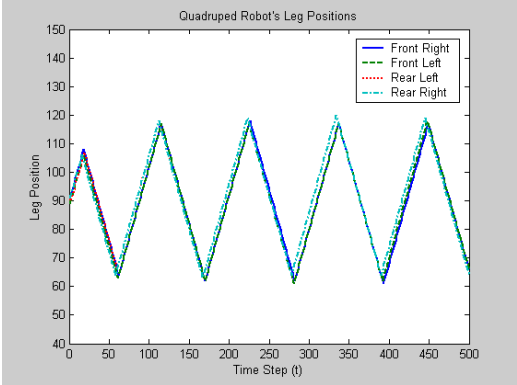
a) 1 module with 4 neurons



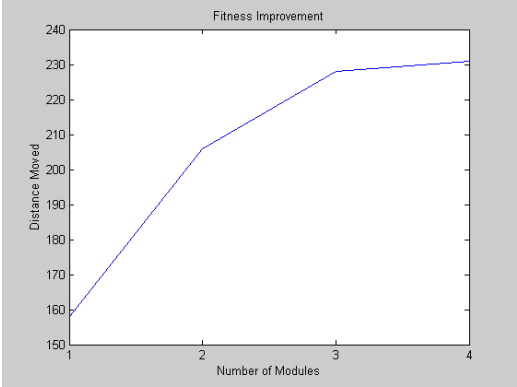
b) 2 modules, 4:1



c) 3 modules, 4:1:1



d) 4 modules, 4:1:1:1

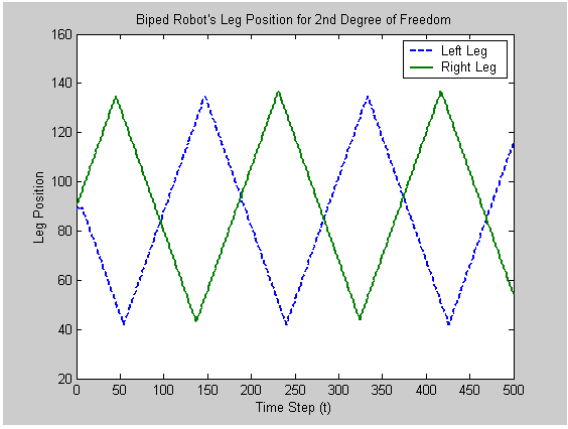


e) Fitness Improvement

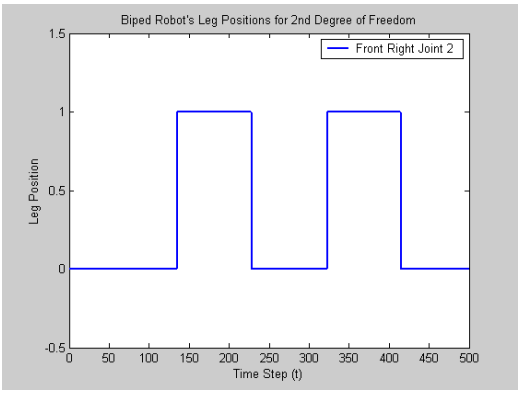
Section C.3
Reference in Page 103

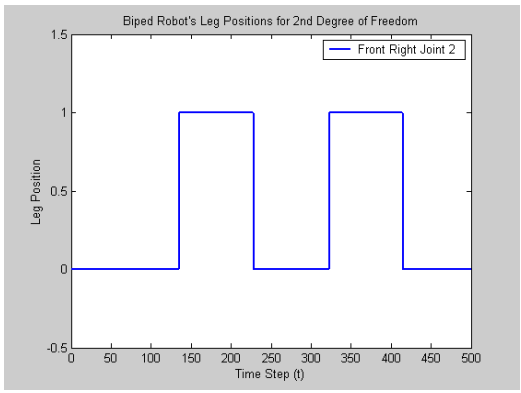
Bipedal walking with 2 active degrees of freedom (5th module output)

Note: x:y:z, where x,y,z… refers to number of neurons in a module



a) Leg Position for both first degree of joints:
5th module with 1 neuron (2:2:3:2:1)





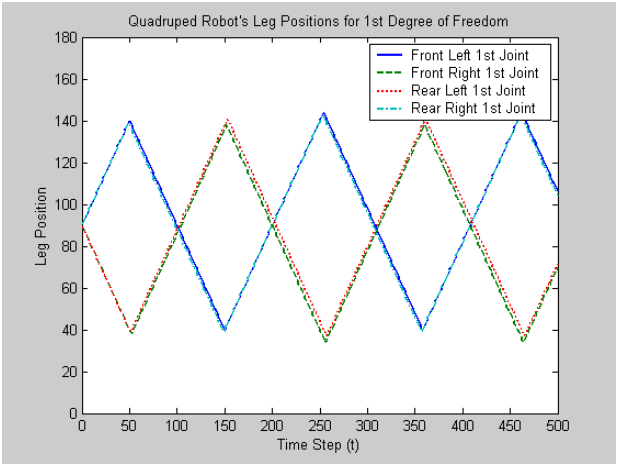
b) Leg Position for front right second joint c) Leg Position for front left second joint

Section C.4

Reference in Page 108

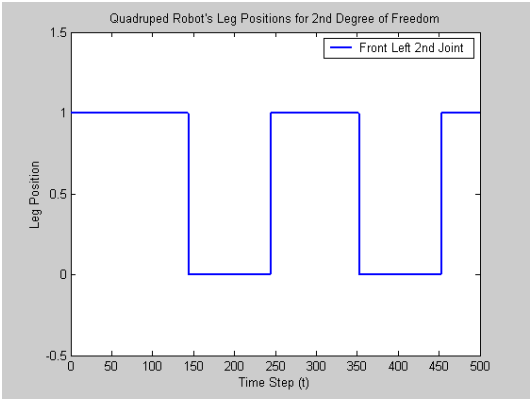
Quadruped trot with 2 active degrees of freedom (8th – 10th module output)

Note: x:y:z, where x,y,z… refers to number of neurons in a module

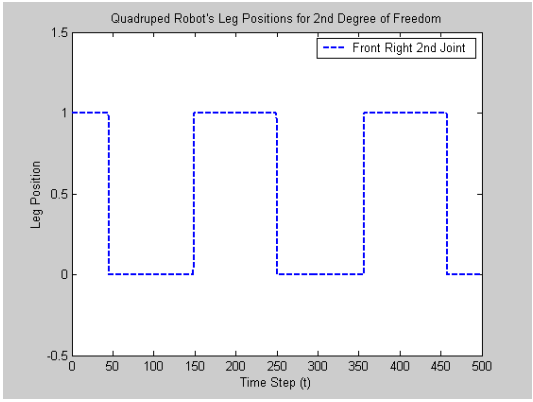


a) Leg Position for all first degree of joints:

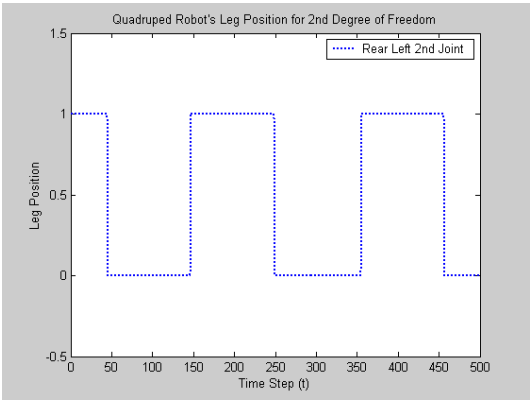
8th module with 3 neurons (5:3:2:4:4:5:4 in previous modules)



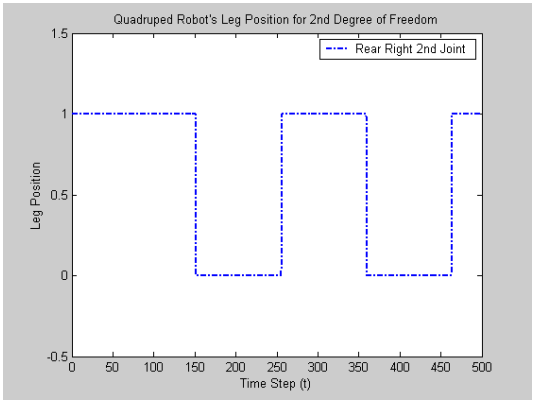
b) Leg position for front left second joint



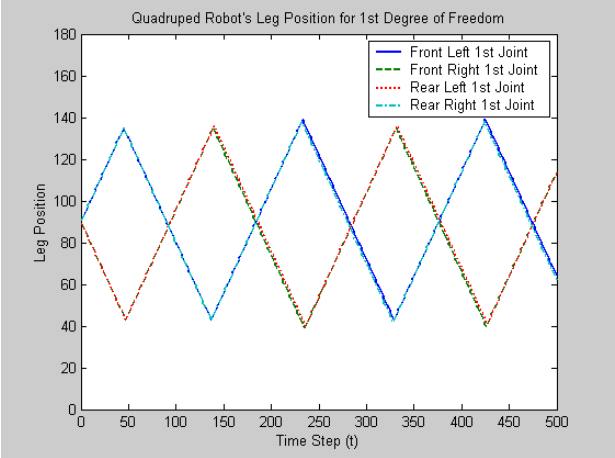
c) Leg position for front right second joint



d) Leg position for rear left second joint

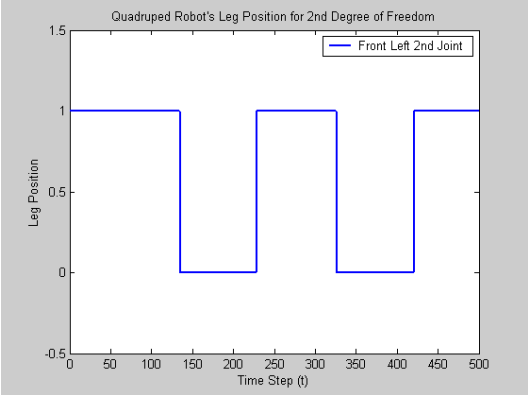


e) Leg position for rear right second joint

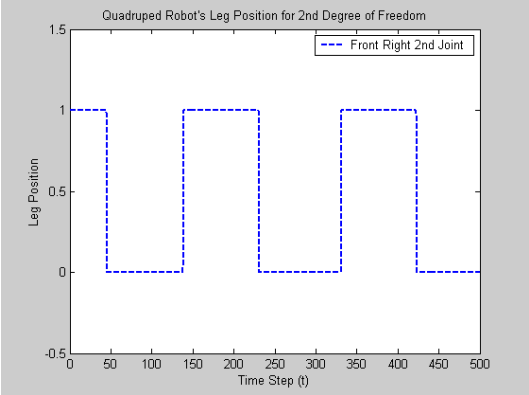


a) Leg Position for all first degree of joints:

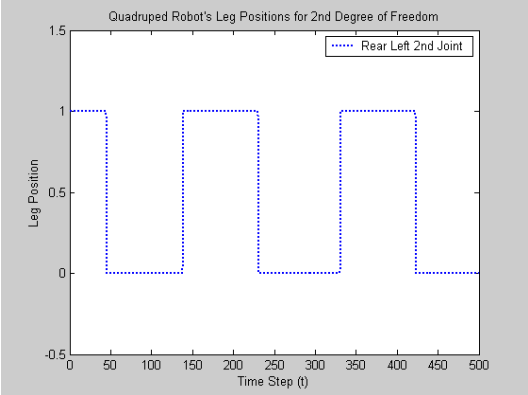
9^{th} module with 2 neurons (5:3:2:4:4:5:4:3 in previous modules)



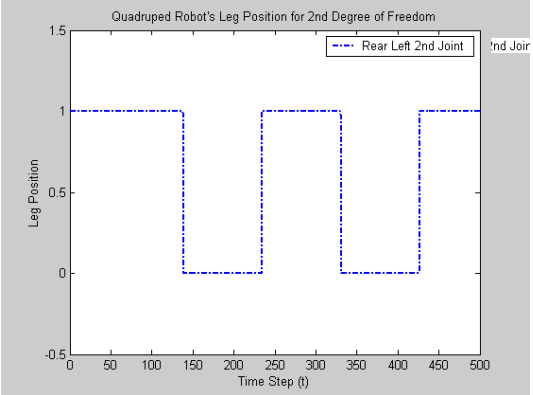
b) Leg position for front left second joint



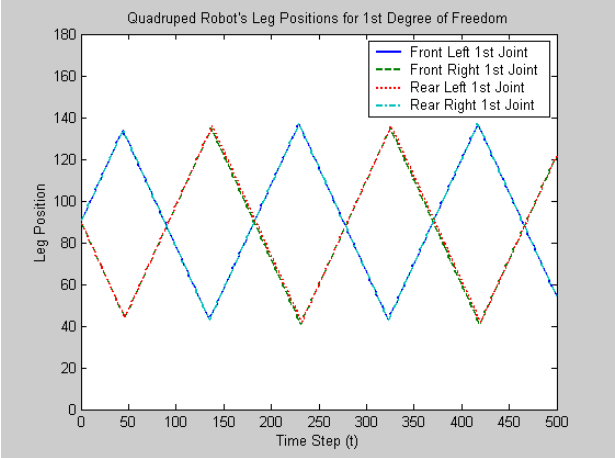
c) Leg position for front right second joint



d) Leg position for rear left second joint

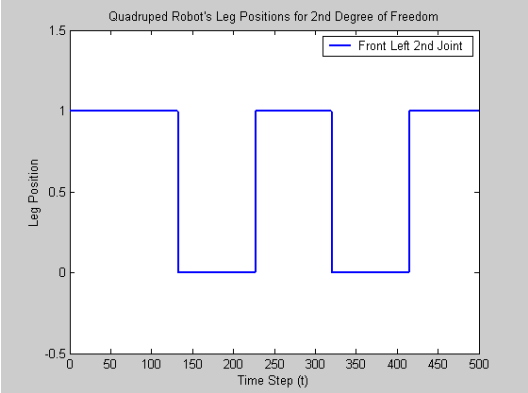


e) Leg position for rear right second joint

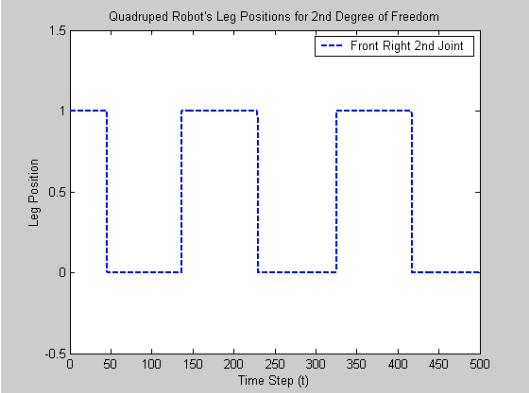


a) Leg Position for all first degree of joints:

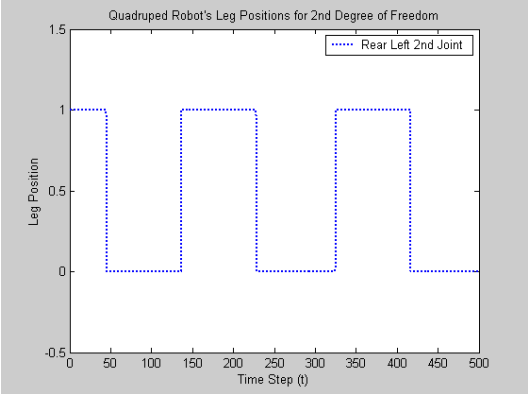
10^{th} module with 1 neuron (5:3:2:4:4:5:4:3:2 in previous modules)



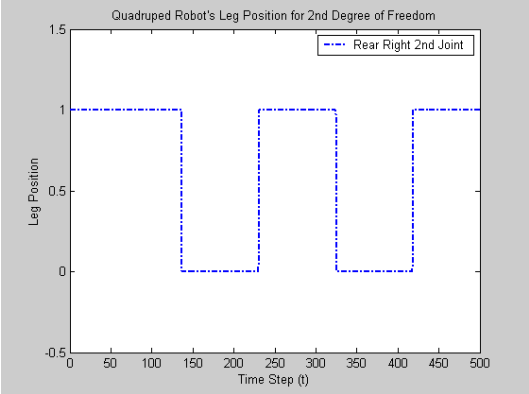
b) Leg position for front left second joint



c) Leg position for front right second joint



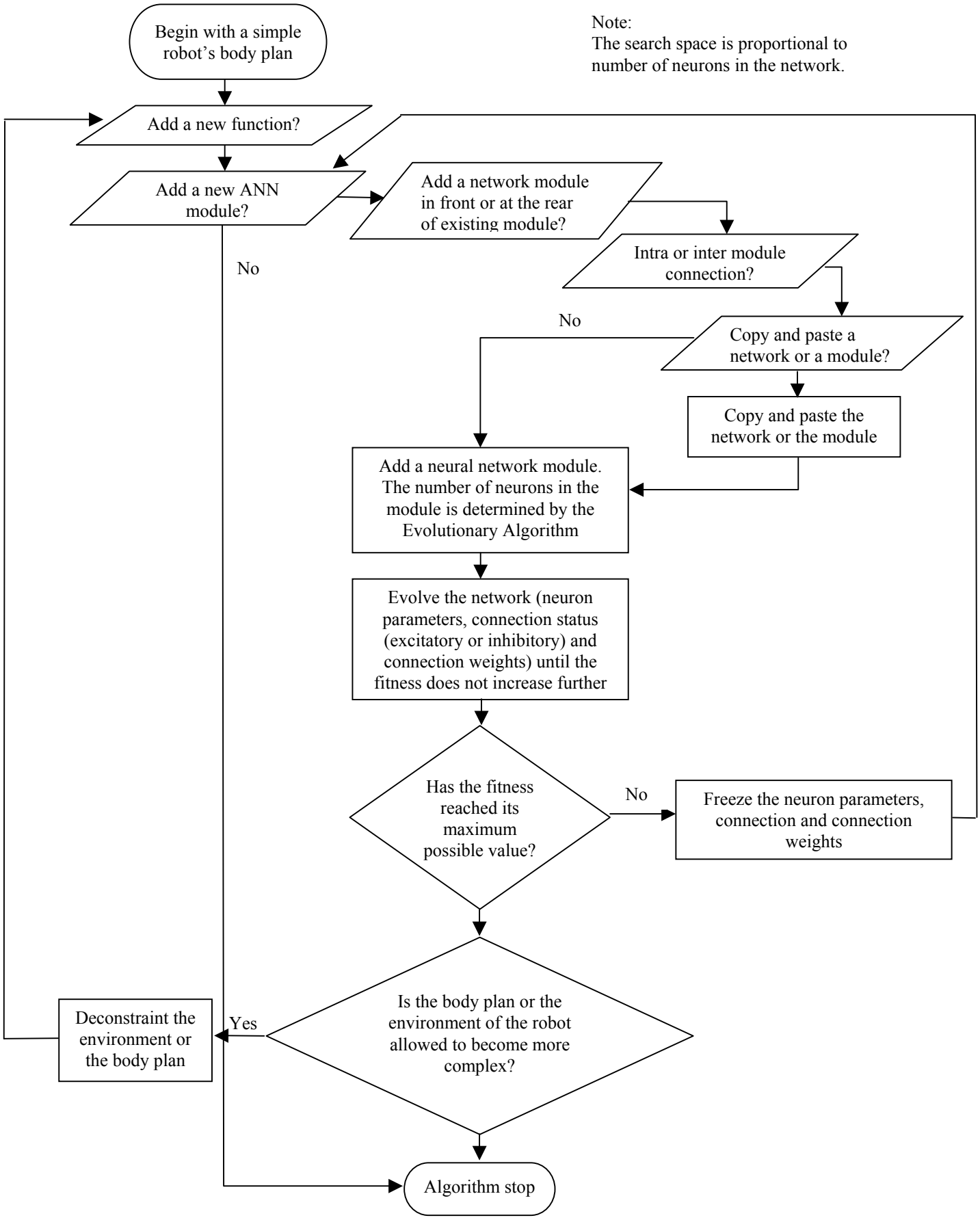
d) Leg position for rear left second joint



e) Leg position for rear right second joint

Appendix D

The flow chart illustrates the principal of the artificial evolutionary technique.


Begin with a simple robot's body plan
Note:
The search space is proportional to number of neurons in the network.
Add a new function?
Add a new ANN module?
Add a network module in front or at the rear of existing module?
Intra or inter module connection?
No
No
Copy and paste a network or a module?
Copy and paste the network or the module
Add a neural network module. The number of neurons in the module is determined by the Evolutionary Algorithm
Evolve the network (neuron parameters, connection status (excitatory or inhibitory) and connection weights) until the fitness does not increase further
Has the fitness reached its maximum possible value?
No
Freeze the neuron parameters, connection and connection weights
Is the body plan or the environment of the robot allowed to become more complex?
Yes
Deconstraint the environment or the body plan
Algorithm stop

Appendix E

This appendix contains a description of the software used in the project in implementing the evolutionary ANN.

The software allows modules of neurons and input sensors to be added or deleted from the network. Neurons in the network can be selected to be an output neuron. There are options to initialize or modify neuron and sensor parameters, connection status and associated weights. These parameters are subject to training when a new module is added. An Evolutionary strategy is used to evolve these parameters. Finally, the trained module can be retained and saved into a text file. Saved networks can also be reused as the network expands. All the results presented throughout Chapter 6 to 8 are obtained using this simulation software. The results presented are "averages" over several experiments and not "one-offs" test data.

The evolutionary technique was programmed using Borland C++ Builder Version 5. There are 70 functions associated with the software which manage the operation of the simulation. The software is divided into two different main windows. The layout of the first window (which handles the Evolution of Modular Artificial Neural Networks) is shown in Figure E.1.

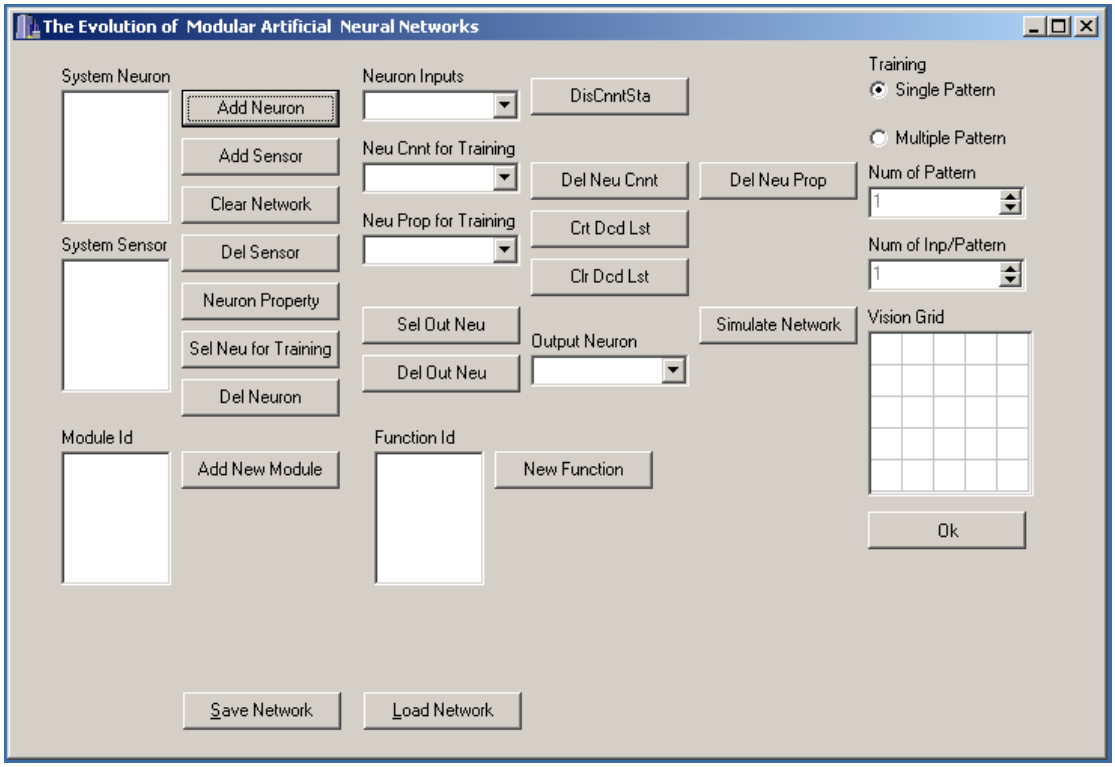


Figure E-1

The software initiates four different types of linked list. These are:

1) System Neuron List (**SysNeuLst**) – Linked list which stores the neurons in the network
2) System Sensor List (**SysSnrLst**) – Linked list which stores the sensor inputs to the network
3) Neuron Connection List (**NeuCnntLst**) – Linked list which stores the neuron connections to be trained
4) Neuron Property List (**NeuPropLst**) – Linked list which stores the neuron properties to be trained

Described below is the operation of the buttons on the layout (Figure E-1) above.

New Function – Assign an ID (N) for different functions added to the system

Add New Module – Assign an ID (N) for each new module added to the network

N is an integer from 1 to +∞, ID is the Identity and M is an integer from -1 to -∞

Add Neuron – A single neuron can be added at a time. The added neurons are assigned an ID (N) for example 1, 2, 3 etc. Firstly, a neuron structure as shown below is created. The data structure for a new neuron requires function ID, module ID, neuron ID, neuron parameters, training status and two linked lists. Then the system neuron list (**SysSnrLst**) is scanned and the last inserted neuron IDs are obtained. The function and module ID is obtained from the form (Figure E-1) above. The neuron parameters values are initialized to zero. The training status determines whether the neuron parameters, input connections and weights associated with the connection will undergo training. The two linked lists are neuron and sensor input list. These lists contain input information from other neurons and sensors in the network. The neuron and sensor input structure is shown below. Since recurrent connections are allowed in the network, a neuron can be connected to itself. Therefore, as soon as a new neuron is inserted into the network, a neuron input data structure is created and added to the input list. This new neuron will receive and make connections to and from other neurons in the network. The number of input data structures varies and depends on number of neuron in the network. The connections weight and status for the input neurons is initialised to zero. System sensor list (**SysSnrLst**) is also scanned and Input sensor data structures are created. The input value comes from the user while the other two parameters are set to zero. Finally, the neuron structure is added to the system neuron link list (**SysNeuLst**).

```
struct Neu              //neuron structure
{
   int fId;             //function ID
   int mId;             //module ID
   int id;              //neuron ID
   double dc;           //decay constant
   double isp;          //internal state parameter
   double th;           //threshold
   int st;              //training status
   TList *NeuInpLst;    //neuron input list
   TList *SnrInpLst;    //sensor input list
};
```

```
struct InpNeu     //neuron input structure
{
   int fId;          //function ID
   int mId;          //input neuron module ID
   int id;           //neuron ID
   double lb;     //connection weight
   int st;           //connection status (connected or not connected)
};

struct InpSnr     //sensor input structure
{
   int fId;          //function ID
   int mId;          //input neuron module ID
   int id;           //sensor ID
   double inp;   //sensor input
   double lb;     //connection weight
   int st;           //connection status (connected or not connected)
};
```

Del Neuron – Removes the selected neuron from the module. If neurons are not deleted from the list in sequence, a background function will then sort the neuron's ID in ascending order. This change is updated throughout the network.

Add Sensor – Add sensor function is very similar to **Add Neuron**. Firstly, when a new sensor is added to the network, a system sensor structure is created and added to system sensor list (**SysSnrLst**). Secondly, the sensor input list (**SnrInpLst**) of each neuron is updated. The data structure for the system sensor is shown below. The reason for having a separate list is to monitor and maintain the growth of the sensors in the network. Sensors can only be added when there is at least one neuron in the network. The button adds a single sensor. Each sensor is assigned an ID (M) i.e -1, -2, etc.

```
struct Snr  //system sensor structure
{
   int fId;   //function ID
   int mId; //module ID
   int id;    //sensor ID
};
```

Del Sensor – Removes the selected sensor from the **SysSnrLst** and **SnrInpLst** of every neuron. If sensors are not deleted from the list in sequence, a background function will then sort the sensors ID in descending order. This change is updated throughout the network.

Clear Network – Removes all the neurons and sensors for the selected function and module ID.

Neuron Property – List the selected neuron parameters from the System Neuron Listbox. There are options to enable and disable training neuron parameters. Figure E-2 below shows the layout for a selected neuron.

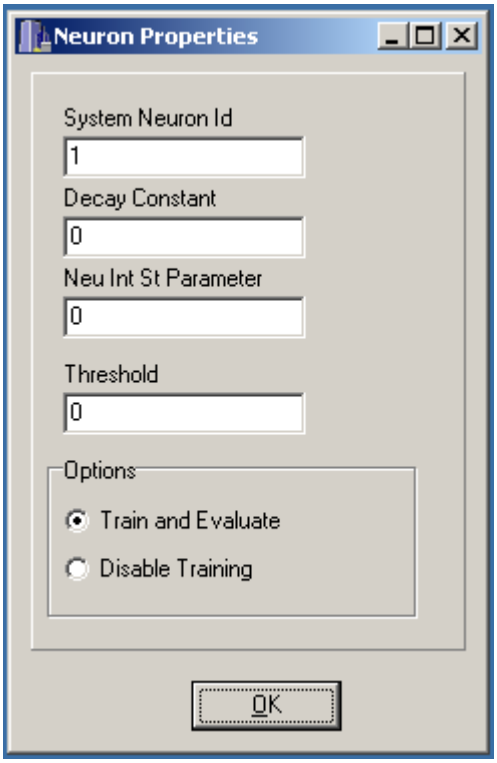


Figure E-2

Sel Neu for Training – This option enables the selected neuron's (of the System Neuron Listbox) inputs (neuron and sensor connections) to undergo training.

Neu Cnnt for Training – Display the neuron's ID whose input connections are selected for training.

Neu Prop for Training – Display the neuron's ID whose neuron properties are selected for training.

Del Neu Cnnt and **Del Neu Prop** – Remove the selected neuron.

Sel Out Neu – Selects the output neurons from the System Neuron listbox.

Output Neuron – Display the selected output neuron.

Del Out Neu – Deletes the selected output neuron from Output Neuron combobox.

Save Network – The network information (neuron parameters, input connections and associated weights, Evolutionary Strategy parameters) is written to a text file.

Load Network – Loads the saved network for evaluation.

Simulate Network – This will test run the loaded network for 500 time steps.

Crt Dcd Lst – Firstly, the Neuron Properties (**NeuProp**) and Connections (**NeuCnnt**) data structure is created as shown. Secondly, **SysNeuLst** is scanned to determine the training and evaluate whether the status of each neuron is enabled or disabled. Neuron input connections or properties of the enabled neuron will undergo training. The neuron connection data structure is added to neuron connection list (**NeuCnntLst**) and the neuron properties data structure is added to neuron properties list (**NeuPorpLst**). Figure 5-7 of Chapter 5 illustrates how the information is decoded into a chromosome.

```
struct NeuProp  //neuron  property structure
{
   int id;           //neuron ID
   double dc;    //decay constant
   double isp;   //internal state parameter
   double th;    //threshold
};

struct NeuCnnt  //neuron connection structure
{
   int NeuId;   //neuron ID
   int InpId;    //Inp neuron/sensor ID
   double st;   //neuron connections, status 0 = connected, 1 = not connected
   double lb;  //weight
};
```

Clr Dcd Lst – Clear decode list erases all the information stored on neuron connection and properties linked list.

DistCnntSta – Disable connection status disables the training status of a neuron. This means the selected neuron properties, input neuron and sensor connections will not undergo training.

Neuron Inputs – Enables the user to view the selected neuron inputs. There are two options. First option views all the neuron input connections. The second option shows all the sensor input connections to the neuron. The neuron Inputs form, as shown below, will appear if neuron is selected. Using this form, it is possible to edit neuron input connection weights and the connection status as shown in Figure E-3.

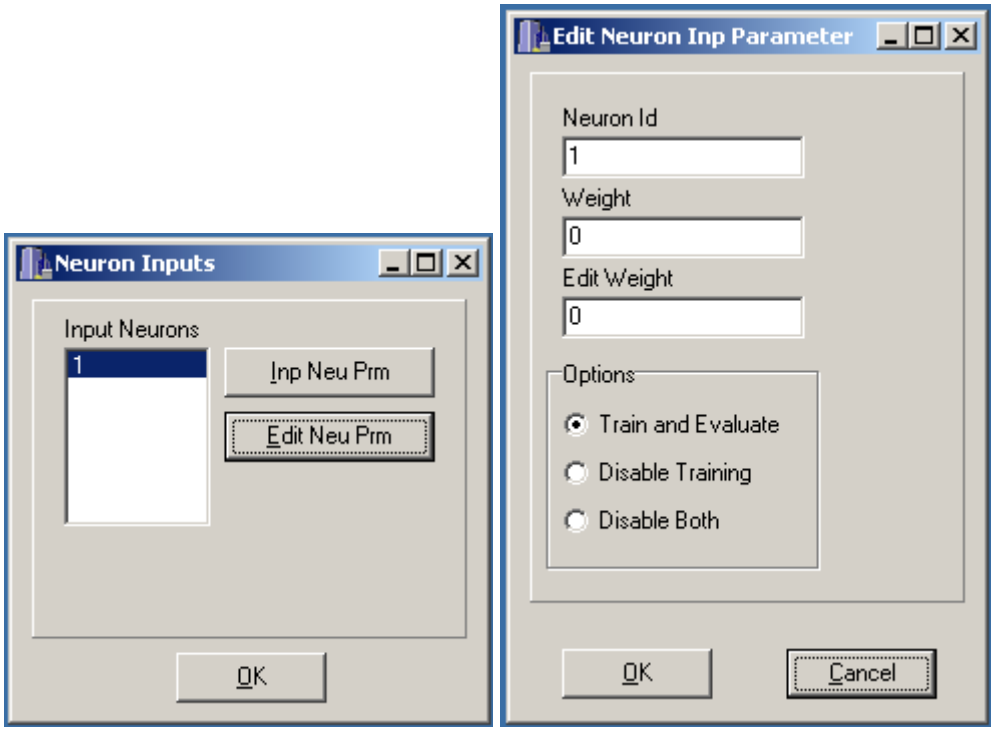


Figure E-3

Sensor Input form shown in Figure E-4 below will appear if sensor option is selected. Using this form it is possible to edit sensor input, connections weights and connection status as shown.

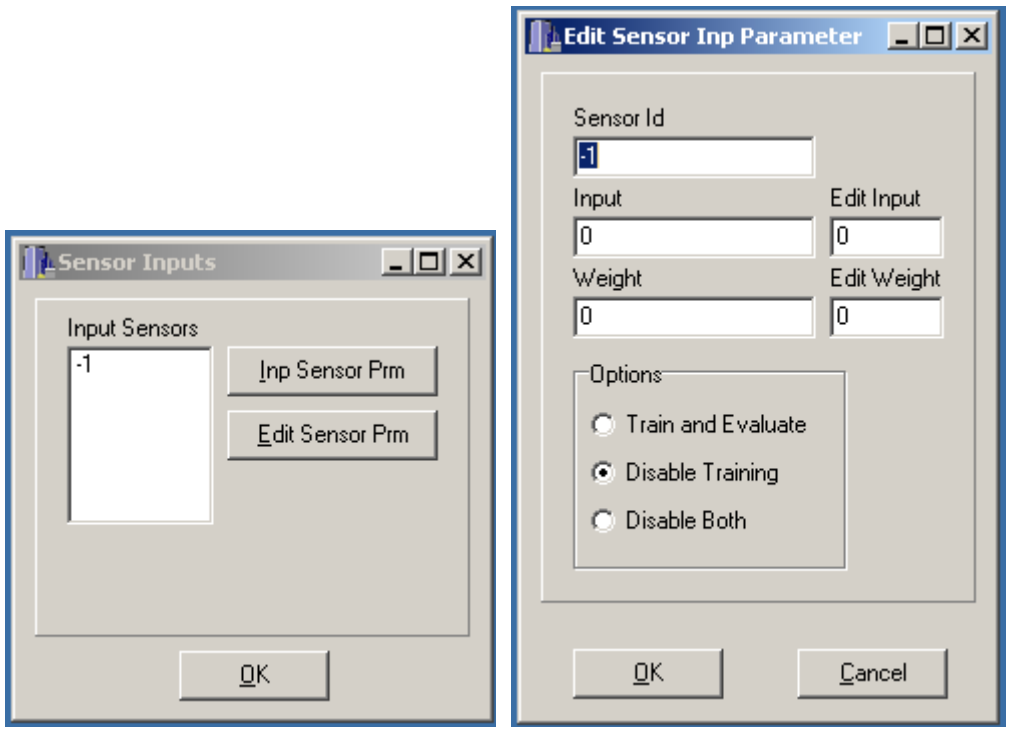


Figure E-4

The vision grid shows the evolution of the sensory system through three different stages. In the first stage, the pixel on the centre of the grid is selected. More details about the vision sensor evolution are given in Figure 8-2 of Chapter 8. There are 25 pixels on the grid. The centre grid is selected for single patterns and it has a predefined input value of 1 or –1. Numbers of patterns and inputs (sensors) per pattern have to be specified if the multiple pattern option is selected. The input sensors become unavailable after the patterns are trained. Clicking on the grid changes the input value and pressing the **OK** button inserts the input pattern.

The layout of the second window is shown in Figure E-5.

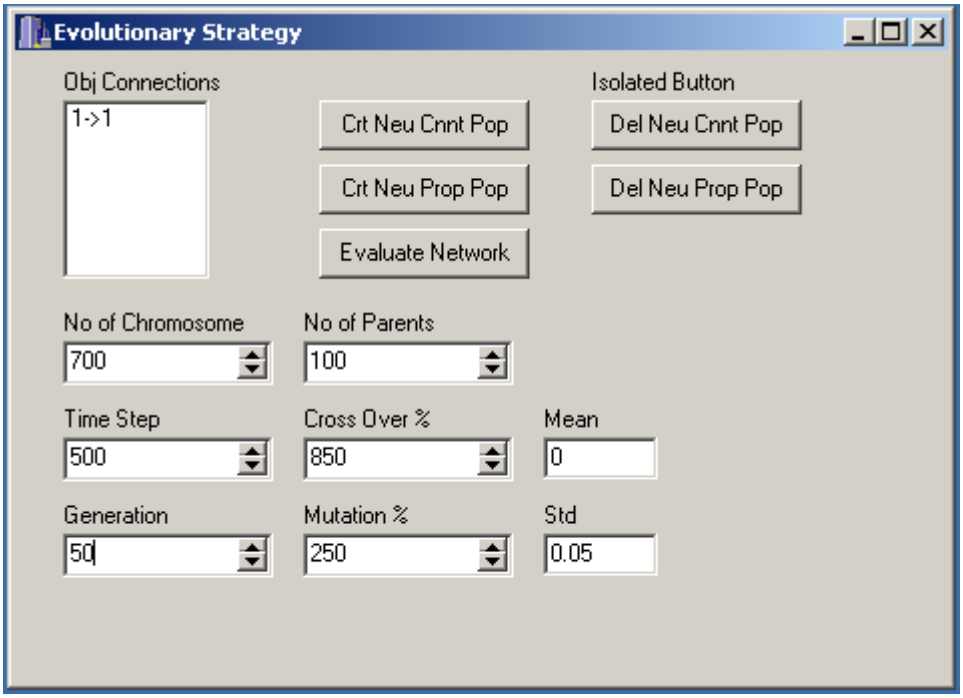


Figure E-5

Obj connections – The Evolutionary Strategy form will appear as shown above, when the create decode list (**Crt Dcd Lst**) button is clicked. The Object connections list shows all the connections that will be trained. For example 1-> 1 means connection from neuron 1 to neuron 1.

Crt Neu Cnnt Pop – Creates a population of neuron connection chromosomes. The Number of Chromosomes determines the size of the population. The information for the population is extracted from Neuron connection data structure. Figure 5-7 of Chapter 5 illustrates how the information is decoded into a chromosome.

Crt Neu Prop Pop – Creates a population of neuron properties chromosomes. The Number of Chromosomes determines the size of the population. The information for the population is extracted from Neuron properties data structure. Figure 5-7 of Chapter 5 illustrates how the information is decoded into a chromosome.

Del Neu Cnnt Pop and **Del Neu Prop Pop** – Deletes the created neuron connections and neuron properties population.

Evaluate Network – Trains the network and updates neuron properties, neuron connections and its associated weights for the specified number of generations.

The set-up of the Evolutionary Strategy genetic operators is explained in detail in Section 5.3 of Chapter 5.