Generating walking behaviours in legged robots

Richard Reeve

Ph.D.
University of Edinburgh
1999

Generating walking behaviours in legged robots

Richard Reeve

Ph.D.
University of Edinburgh
1999

Abstract

Many legged robots have been built with a variety of different abilities, from running to hopping to climbing stairs. Despite this however, there has been no consistency of approach to the problem of getting them to walk. Approaches have included breaking down the walking step into discrete parts and then controlling them separately, using springs and linkages to achieve a passive walking cycle, and even working out the necessary movements in simulation and then imposing them on the real robot. All of these have limitations, although most were successful at the task for which they were designed. However, all of them fall into one of two categories: either they alter the dynamics of the robots physically so that the robot, whilst very good at walking, is not as general purpose as it once was (as with the passive robots), or they control the physical mechanism of the robot directly to achieve their goals, and this is a difficult task.

In this thesis a design methodology is described for building controllers for 3D dynamically stable walking, inspired by the best walkers and runners around — ourselves — so the controllers produced are based on the vertebrate Central Nervous System. This means that there is a low-level controller which adapts itself to the robot so that, when switched on, it can be considered to simulate the springs and linkages of the passive robots to produce a walking robot, and this now active mechanism is then controlled by a relatively simple higher level controller. This is the best of both worlds — we have a robot which is inherently capable of walking, and thus is easy to control like the passive walkers, but also retains the general purpose abilities which makes it so potentially useful.

This design methodology uses an evolutionary algorithm to generate low-level controllers for a selection of simulated legged robots. The thesis also looks in detail at previous walking robots and their controllers and shows that some approaches, including staged evolution and hand-coding designs, may be unnecessary, and indeed inappropriate, at least for a general purpose controller. The specific algorithm used is evolutionary, using a simple genetic algorithm to allow adaptation to different robot configurations, and the controllers evolved are continuous time neural networks. These are chosen because of their ability to entrain to the movement of the robot, allowing the whole robot and network to be considered as a single dynamical system, which can then be controlled by a higher level system.

An extensive program of experiments investigates the types of neural models and network structures which are best suited to this task, and it is shown that stateless and simple dynamic neural models are significantly outperformed as controllers by more complex, biologically plausible ones but that other ideas taken from biological systems, including network connectivities, are not generally as useful and reasons for this are examined.

The thesis then shows that this system, although only developed on a single robot, is capable of automatically generating controllers for a wide selection of different test designs. Finally it shows that high level controllers, at least to control steering and speed, can be easily built on top of this now active walking mechanism.

Acknowledgements

Thanks to everyone who helped me take as long as I did over this thesis — they know who they are, but particularly my flatmates and the Mobile Robots group. Thanks to Louise for support and proof-reading above and beyond the call of duty, and to my supervisor and examiners for spotting the mistakes which would otherwise have littered the thesis — hopefully there aren't too many left. Finally thanks to my parents not hassling me too much for still not having a proper job, and supporting me through the years this has taken.

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

R E Reeve
Edinburgh
16th December 1999

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many legged robots have been built with a variety of different abilities, from running to hopping to climbing stairs. However, despite this there has been little consistency of approach to the problem of getting them to walk — every research group has followed its own instincts. For instance, some have broken down the walking step into discrete parts and then controlled them separately, some have used springs and linkages to achieve a passive walking cycle, and some have worked out the necessary movements in simulation and then imposed them on the real robot. All of these approaches have limitations, although most were successful on the robots for which they were intended.

The design of a controller for dynamically stable walking machines which can be used across as wide a spectrum of machines as possible would allow the field to progress beyond just building the robots and making them walk, towards actually putting the robots to use in real situations.

This has been the focus of my research, and inspiration on how to design controllers for a wide variety of different body conformations came from the best walkers, runners and stair-climbers around — ourselves — and as a result the control systems produced are based on the vertebrate Central Nervous System. The controllers are built by an evolutionary mechanism which adapts them automatically to any robot with which it is presented. These controllers then become part of the now walking robot, which is itself controlled, as to direction and speed for instance, by a higher level controller. This is an easier task than before because the robot is already walking, and so only has to have its gait modified rather than a single monolithic controller having to determine the position of limbs or torques in joints directly for all speeds and directions, as is normally the case.

1.1 Motivation

Some people believe that control systems give commands to mechanisms. But mechanisms have a mind of their own: they will obey physical laws. Control is not to compensate for the limitations of poorly designed mechanisms. The best systems will have mechanism and control designed to work together in harmony.

Marc Raibert (ISToMM'93)

Marc Raibert was acknowledged as one of the leaders in the field of legged robotics until he moved into industry in 1993, and the above extract from a talk he gave made me think about the mechanisms which people use when trying to make robots walk. It seems to me that almost without exception they are not designed for walking at all, but rather to be as general purpose as possible, and indeed that the whole purpose of the "controllers" is to get them to walk in the first place, not to control walking, as they have no innate ability. Exceptions to this include passive walkers (for instance the "biped glider" of McGeer, 1989) which are designed not to require a controller (or even power) for walking down gentle slopes, and spring actuated robots (*e.g.* Wadden *et al.*, 1993), all of which have springs and dampers built into the physical robot. However there is a significant problem with this approach: although the end result is generally a very competent walking robot, because all the springs and linkages which make the mechanism so effective are permanently in place the better the mechanism is at walking the worse it becomes at everything else — until in the limit it becomes as uncontrollable as the walking automata of the turn of the century whose intellectual successor it is, and it becomes impossible to get it to manoeuvre or climb over obstacles or achieve anything else which might be desirable in the context of a robot designed to carry out a task. However, the idea of the mechanism being inherently able to walk makes sense since it would make the control job so much more straightforward, if only there were a way of retaining the multifunctionality which the more basic robots have.

The answer in the end is clear — if you don't want the springs all the time, why not simulate them through the motors so that when you switch off the simulation you are left with your general purpose robot again? There was a clear precedent for this which encouraged optimism in the approach — the vertebrate spinal cord. Take the cat, for example. Clearly this is an extremely versatile animal capable of an enormous range of movements — yet it can be induced to walk very easily by sending a simple signal down the spinal cord which excites the Central Pattern Generators (CPGs) associated with walking (Grillner, 1985). These do not exactly simulate the springs and links mentioned above, but they have a similar effect — they actively alter the dynamics of the legs through the muscles to create a new dynamical system where walking is a stable attractor.

The significance of this point becomes clear when you realise that, both with the spinal cat[1] and with a robot with simulated springs and linkages, what you still have is a dynamical system which can be controlled: in the cat's case this is done by various higher centres in the brain, notably the cerebellum, but in the robot's case it can be done by a more conventional controller. This should be much simpler than most walking controllers, as it no longer has to "compensate for the limitations of a poorly designed mechanism", but rather controls an (active) walking mechanism and so only has to concern itself with maintaining the stability of the walking behaviour on rough terrain and perhaps during gait changes, as well as higher level concerns such as direction and speed of movement. The latter are very simple to control in vertebrates — for instance, the higher the excitation of the CPGs in the cat, the faster the cat will go, changing gaits automatically as it speeds up (Grillner, 1985), and in the lamprey, where Grillner and his colleagues have mapped the entire structure of the CPGs (Grillner *et al.*, 1991), it is found that exciting the CPGs on one side of the body more than those on the other side (which again is very easily done) causes the lamprey to move smoothly away from the excited side.

Strangely, Raibert's robots (Raibert, 1986, 1988) do not follow his own advice — neither the mechanism nor any individual part of the controller walks on its own, so the controller has to do the whole job in one go and make the robot walk as well as

[1] a cat whose spinal cord has been severed just below the brain

control the walking all at once. This has resulted in all of Raibert's controllers being carefully handcrafted, a time-consuming process, though, despite that, the controllers that he and his successors in the MIT Leg Lab have made have been the closest yet made to the general purpose controllers that we are looking for.

In fact very few people seem to have designed walking *mechanisms* at all since automata were replaced by controlled robots in the 1950s. All those that arguably have done so have two things in common (except the passive walkers which have no controller) — firstly they are controlled by neural networks (this is almost inevitable as they are the only well researched computational system which can be trained and have their own continuous dynamics), but secondly and more interestingly, although they alter the dynamics of the robots with neural networks to make them walk, they do not appreciate they have done this but rather describe their neural networks as controllers in their own right and do not go one step further to then design a (higher-level) controller for the walking robot. In trying to do it all in one go the results tend to be fairly poor, and certainly considerably less inpressive than Raibert's algorithmic approach.

1.2 Summary of Achievements

In this thesis I describe a new design methodology for creating legged robot controllers, where a low level controller entrains to the dynamics of the system, adapting it so that walking is a stable attractor, allowing a higher level controller to be very simple and easy to build when compared to equivalent controllers for the original robot. This is a general purpose design strategy which can build controllers for any robot which can be described in the simulation language provided. Indeed, even if this is not possible, the language is easily extensible to cover (for instance) new actuators, sensors, and even joint types, and the design strategy can then be applied to the new robot in exactly the same manner. The significant difference between this work and previous simulation work on walking such as that by Beer and Gallagher (1992) is that the simulator was a full 3D simulator, and dynamically stable walking was modelled.

In pursuing this goal a mechanism was built for measuring the ability of a neural network to control a legged robot in simulation, and as this proved to be a challenging real-world task a detailed comparison of a variety of evolutionary methods for building neural networks was carried out and an analysis done of the results to see how different approaches compared on a standardised but taxing problem.

The whole system consists of a mechanical simulator with its associated simulation language, a neural simulator capable of simulating a variety of different types of neurons as well as any network configuration, an evaluation mechanism for determining how well a specific network performs in controlling the robot, a genetic algorithm for evolving the controllers, and a series of implementations of the different neural network encodings used in experiments.

1.3 Organisation of this thesis

Chapter 2 discusses past and present work in walking research, looking mostly at robotics and its precursors but also, where appropriate, at locomotion in vertebrates and its neuromuscular control, showing how these strands tie into the research done in this thesis.

Chapter 3 contains a review of relevant neural network research, particularly evolutionary approaches, and describes in detail a variety of evolutionary encodings of neural networks, some of which are examined in this thesis.

Chapter 4 presents a detailed examination of the system used to build, model, and evaluate walking robot controllers, including details of validation experiments, robot models used, and how analyses were carried out.

Chapter 5 provides the initial results of the simulator on simple encodings, showing the potential of the system to learn appropriate behaviours, and compares a variety of different neural models.

Chapter 6 shows the detailed results of the selection of encodings implemented for comparison on the system, and then looks at how different fitness functions can help in evolving walking robots.

Chapter 7 uses the best system from the previous chapter to evolve controllers for a variety of different test robots to show that the algorithm developed is sufficiently general purpose. It then shows that it is simple to add a higher level controller to the active mechanism evolved which allows it to be steered and accelerated.

Chapter 8 examines what has been achieved in this thesis and suggests avenues for further research.

Chapter 2

Walking research

After a long time as a junior partner in robotics, walking has seen an explosion in interest and achievements in the last fifteen years as computational power unimaginable until recently has been brought to bear on its problems. At the same time we have seen advances in other related fields: non-linear dynamics has helped us understand and model the robotic systems that we are studying; zoology has provided details of the mechanics of animal locomotion to aid in the design of our robots; and neuroscience has developed a deeper understanding of how rhythmic movements like walking occur in the natural world, and this has helped us formulate new ideas for robotic controllers.

In this chapter I will give an overview of the history of walking research, following its progression from the study of body parts and the construction of clockwork automata to the understanding of neural rhythm generators and the control of robotic somersaulting[1]. We shall see that it has become a focal point of interdisciplinary research between the biological sciences and robotics, and we shall examine what can be gained from this work.

2.1 Early research

Fascination with walking goes back millennia, but research began in the 18th and 19th centuries. Interest arose for varying reasons, not all of them scientific.

[1] An earlier version of this chapter appeared in (Reeve, 1999b)

2.1.1 Studies of gait

Some of the earliest work involved the examination of cadavers to investigate how they were constructed (for instance work by J. C. Lavater in the late 18th century), but two significant studies of gait really started the ball rolling.

The first of these was by E. J. Marey who invented a pneumatic recording device in the 1870's which measured stepping patterns through sensors attached to the feet of subjects which moved the pen on a clockwork recorder (Marey, 1874). This allowed him to make records of step patterns in different gaits.

The second (and far more famous) was Eadweard Muybridge, a photographer who, initially spurred on by a bet to prove whether a horse lifted all of its legs off the ground simultaneously in a gallop, went on to record a huge collection of high-speed photographs of animal gaits (*e.g.* Muybridge, 1887).

These people took some of the first steps in researching the field and though they came from very different backgrounds, they were looking for the same information. This has always been a problem for walking researchers — it is an attractive subject to investigate, and it has no particular allegiance to any one field. As a result there is a danger of duplication of effort as new disciplines decide locomotion is a field worthy of study. Zoologists and physiologists were amongst the early researchers (*e.g.* Gray *et al.*, 1938): mathematicians found the apparent simplicity of these patterns interesting, and group theoreticians examined their properties (*e.g.* Collins and Stewart, 1993a,b; Collins and Richmond, 1994); neuroscientists began to study the structures which control locomotion (*e.g.* Grillner, 1985), and roboticists have tried to emulate the ability, as well as embodying their theories of intelligence in legged robots (*e.g.* Raibert, 1988; Brooks, 1989); non-linear dynamicists have examined the whole walking system to determine its dynamics and stability (*e.g.* Kelso, 1995); biomechanicists have looked at how to improve running performance, and are trying to stimulate nerves in paraplegics to allow them to walk again (*e.g.* Yamaguchi and Zajac, 1990); finally, computer graphics researchers have created the illusion of walking for our entertainment in the cinema or on our computers (*e.g.* Toy Story).

All of these people have different contributions to make, but it has become impossible for anyone to keep track of their different approaches and achievements. However, it is very important to legged robot research that some feel for the overall picture is maintained, and that is the purpose of this chapter.

2.1.2 Automata

In the mid 18th century, even before these early studies of walking in animals, automata were being constructed to mimic life as closely as possible and were being demonstrated at fairs and exhibitions. One of their greatest creators was J. de Vaucasson, whose aim was to produce the perfect artificial person (Elliott, 1997). He produced a flautist which could imitate the sound of the instrument and move in a lifelike fashion. It was incredibly intricate and was controlled by hundreds of bellows and levers. He went on to make a mechanical duck which could flap its wings, walk, and even eat, drink and defecate by means of a mechanical stomach. His project was continued in the 1770s by P.-J. Promond and H. Lois who made increasingly humanoid automata one of which (the Draftsman) could make writing movements and follow them with its head, and another could even reportedly play the harpsichord. Of course it is impossible to authenticate these reports and various automata were exposed as frauds, but incredibly complex machines were certainly being made around this time, and their makers boasted of mimicking life itself.

In the 1850s Chebyshev invented mechanical linkages which connected joints so that they moved together. This allowed walking to be developed much more easily as linkages could be designed which would make the body move horizontally by moving the feet and legs in a fixed pattern, and many walking automata were designed by this method (Raibert, 1986).

One of the difficulties with this early approach however was that the rigid mechanical linkages which made walking possible totally fixed the movement of the legs. This meant that no alterations in the gait could be made, for instance to change gaits and move over uneven terrain. Subsequent decades were spent designing better linkages in an attempt to produce suitable stepping motions to generate stable locomotion. However, to allow the gait to change some means of control would have to be devised, and there the problem lay for half a century.

2.1.3 Peripheral vs. Central Control

During these early years of the 20th century Neuroscience began to take a serious interest in locomotion, debating how control of locomotion and other rhythmic movements occurred in humans and other animals. Two competing hypotheses came to the fore: Peripheral Control and Central Control.

The former claimed that these movements were achieved through sensory feedback: a reflex chain existed where each phase of the motion cycle provided the sensory cues which triggered the correct timing of the next in a repeating loop. Thus the behaviour would be disrupted by a lack of sensory feedback.

The latter claimed that the Central Nervous System (CNS) does not require sensory feedback to provide the proper timing, but rather that there is a neural pacemaker providing the rhythm which, though it may be modulated by feedback, was essentially independent of it.

Both hypotheses had early supporters, but experimental evidence was scarce until the 1930's and 40's, when much support was found for the importance of sensory feedback.

Peripheral Control

Some of the best work supporting Peripheral Control was done by Sir James Gray and his colleagues (*e.g.* Gray *et al.*, 1938; Gray and Lissmann, 1940): they showed that there were behaviours in both invertebrates and vertebrates which seemed to consist of chains of reflexes. They also claimed that the same experiments, when repeated on deafferented[2] animals, produced no observable rhythmical motions. These last results have been largely refuted by more careful recent experiments, but the early behaviours they describe, such as reflex walking in the spinal toad when it is held against a moving surface, have been widely repeated.

Other work on a variety of animals showed that particular sensory inputs can disrupt or completely arrest normal motor output. For instance, in one experiment a bivalve scallop with its shell bound shut was found to completely stop contracting and relaxing

[2] sensory (afferent) nerves are cut so no feedback is received by CNS

its adductor muscle (Delcomyn, 1980).

Results like these were taken at the time to support the Peripheral Control hypothesis; however, in reality Central Control did not preclude them, except those which were later shown to be in error; it only said that sensory feedback was not necessary to create the rhythm, not that it did not play a significant part.

This became important when, in the 1960's and 70's, evidence came in which proved conclusively that rhythmical motions could occur without sensory feedback.

Central Control

A great body of evidence was gathered by various researchers (for a table see Delcomyn, 1980, page 494), which showed that creatures from right across the animal kingdom could carry out rhythmical actions when their nervous systems were completely isolated[3], deafferented, or when their muscles were paralysed. In all these cases there was no feedback to allow reflexes to generate the movement. That these results were not gathered before was in some cases a result of poorer experimental techniques and equipment which failed to pick up the rhythms, and in others because the experimental procedures caused too much extraneous damage for the nervous system to be able to continue to operate normally.

On the basis of this new evidence, there was no doubt that central mechanisms did generate many rhythmical motions, and the idea that all meaningful output had to be driven by specific sensory stimulation died. However, the central control hypothesis fell far short of explaining what was happening in these pacemakers that it proposed, and certainly observations like the reflex walking referred to earlier needed to be put into the structure of central control; indeed some more recent work had also shown results which did not fit in with a strong central theory. For example, Grillner and Wallén show that a spinal dogfish paralysed by curare will, if the tail is moved at a frequency different from its natural swimming rhythm, show co-ordinated bursts in its motoneurones at the imposed frequency (Grillner and Wallén, 1977). Also, as in scallops, the swimming movement can be suppressed by strong stimuli like holding the

[3] removed from the body

body tightly.

Experiments like this led to the realisation that a more sophisticated explanation would have to be found to replace the rather simplistic Central Control hypothesis.

2.1.4 Early Robots

Meanwhile, back in robotics, the first true walking robots had already been built. It was the arrival of logic circuits and later computers which made this possible by allowing a control mechanism to be built which would make variation in the walking patterns possible.

Nevertheless, one of the first ideas was to use a human as the controller, as this was still early days for computing. General Electric built one such vehicle in the 1960's (the "versatile walking truck" in Mosher, 1968) which, under the control of its operator, was capable of up to 5 mph and could climb over large obstacles. This was really ignoring the control problem however, and it was not until the late 60's that truly independent legged robots began to appear.

One of the first of these was built by Frank and McGhee and was called the Phony Pony (McGhee, 1976): each joint was controlled by a finite state machine made from digital logic circuits with each of four states triggering the next in a fixed loop. This was actually very restrictive, and made the robot behave in a fixed manner very similar to the automata it replaced. However it opened the door to the gait being computer controlled, and thus changeable in software.

After this, many research projects were started into computer controlled walking, and indeed the first commercial product, Big Muskie, which was a walking dragline used for strip mining, was produced by the Bucyrus-Erie company in 1969.

The late 1970's and 80's saw a succession of simple computer-controlled statically stable robots whose patterns of locomotion were very simple and inspired by insects (*e.g.* Gurfinkel *et al.*, 1981; Hirose and Umetani, 1980). They remained balanced in static equilibrium all the time, and moved surplus legs to new positions where they could in turn be used for support. They were the first computer controlled walking robots, and they moved very slowly (as they always had to keep 3 legs on the ground for support). Their controllers tended to be reactive (*i.e.* they would move to a new state based on the sensory feedback and the state they were in), and parallels can easily be drawn to the Reflex Controllers mentioned earlier. This strategy allowed fairly robust movements across terrain, and so long as a state was described in the controller (*e.g.* leg moving forward hits obstacle), it could be dealt with by the robot (move back, raise leg, and repeat). However, this approach does require the enumeration of possible states, which is far from ideal. These controllers were time-independent, in the sense that the system could be frozen and restarted at any point, or run at a different speed, and there would be no effect on the walking pattern, so it didn't matter if the computer had to sit and calculate for a bit to work out what to do next.

The next step forward was the realisation of dynamically stable (or actively balanced) walking, which arrived as computers became more powerful, and capable of dealing with the complexities involved in staying upright when not always in a stable posture (*e.g.* Matsuoka, 1979; Miura *et al.*, 1984; Raibert *et al.*, 1984). Controllers for this kind of robot were time-dependent, that is to say that there were points in the motion cycle when stopping or changing speed would be fatal (imagine stopping moving your legs in the middle of a fast step), so the controller had to be able to keep up with what was going on in real time.

2.2 Recent developments

At this point walking was still poorly understood. Physiologists had studied the bodies of animals, neuroscientists had argued about how they walk, and roboticists had created slow, clumsy walkers, but things had really yet to take off. The pace of change was soon to speed up however.

2.2.1 First steps with dynamic stability

The first attempt at a dynamically stable robot was by Ogo *et al.* (1980), where a biped with huge feet walked in a quasi-dynamically stable fashion. They avoided the problems of time-dependence by only having a small non-statically stable phase where the biped 'fell' from one foot to the other in a controlled fashion.

True dynamic stability was not long in coming however. At the same time as the above Matsuoka built a robot capable of running in a 2 dimensional world (Matsuoka, 1979), and not long after Miura and Shimoyama developed the first actively balanced dynamically stable walker, the stilt biped, which was supposed to model the behaviour of a person walking on stilts (Miura *et al.*, 1984).

Raibert and his team at Carnegie-Mellon University (it was later to move to MIT) then started to produce walking robots. Initially he designed a 3D one-legged hopping machine (Raibert *et al.*, 1984): shortly after this he progressed to bipeds and quadrupeds using the same basic algorithm for each (Raibert, 1986). They had finite state controllers, but within each state the controller was algorithmic, calculating the desired joint angle and the required joint forces. This was a more centralised control than the statically stable robots and the robots went at impressive speeds (for instance the one-legged robot had a top speed of 4.5 mph).

On a different front, McGeer built a dynamically stable passive walker which he called the "biped glider" (McGeer, 1989, 1990). This could walk stably down gentle slopes without any form of control, and could in theory be pumped to walk on the flat or on other terrain. This has been taken further by Goswani and colleagues at INRIA in (Goswami *et al.*, 1997) who looked at passive walking with a very simple gait, and then at the mimicking of passive control with an active mechanism to enlarge the natural basin of attraction of the passive limit cycles and to create new gaits.

2.2.2 Central Pattern Generators

Much progress has also been made by neuroscientists studying rhythmical controllers. Following on from the ideas of Central Control, during the 1970's the idea grew that the motoneurones (and hence rhythmical movements) in vertebrates were driven by central networks of interneurones that generated the essential features of the motor pattern, but also that sensory feedback signals played a crucial role in the control system, namely to turn a stereotyped unstable pattern into the co-ordinated rhythm of the natural movement. The networks were referred to as Central Pattern Generators (CPGs), and evidence showed that every part of the body which makes cyclical movements has its own individual CPG (for a summary, see Delcomyn, 1980). Experimental evidence has recently proven this to be true in the lamprey (Wallén *et al.*, 1992), where neurophysiologists actually mapped out the neurones making up the CPG, and it is accepted in other animals as well.

In vertebrates the spinal cord contains these neuronal networks. The CPGs, when stimulated, have their own dynamics which set up oscillations in outputs between the neurones. When they are connected to the motoneurones, these generate the characteristic rhythmical behaviour associated with the system. Central Pattern Generator is a general term: for instance, in walking each muscle can be considered to have its own CPG, but opposing muscles CPGs can be considered collectively to form a joint CPG, and similarly for limb CPGs. Feedback from the muscles keeps the CPGs in phase with the limb, so the pattern does not break down. In the same way, limb CPGs maintain the co-ordination between legs to generate the overall behaviour (*i.e.* walking) by their cross connections. The system is self-correcting, so that if any part loses synchronisation, the dynamics of the whole system forces it back into the original rhythm. Without the cerebellum, movements are coarser and there are some problems with equilibrium and co-ordination, but in essence the pattern remains unchanged: thus spinal mammals and birds have been shown to make walking movements very similar to those of intact animals (Grillner, 1985). On the other hand, without proprioceptive feedback the pattern can break down (Delcomyn, 1980), as it can be very important for adaptation to actual conditions. Even with this feedback the rhythm can break down though if it is knocked too far from equilibrium, as the movement now falls outside the basin of stability of the system.

As a result of work by Sten Grillner and his colleagues in examining and simulating CPGs in the lamprey (Grillner *et al.*, 1991; Wallén *et al.*, 1992), a great deal is now known about their structure and behaviour in this creature; however there is still much to be learnt in other animals — of particular interest to us are legged vertebrates: we know that each muscle has its own CPG and that this is essentially a very simple network, and that these are connected together inside each leg to generate a stepping motion, and between legs to generate a stable rhythm, and we also know that each of these rhythms (gaits) is stable over a certain range of speeds, so as the legs speed up, the basin of stability for one gait shrinks until a bifurcation occurs and that gait becomes unstable, and walking moves to a new gait with a different attractor (Kelso, 1995). We also know that the cerebellum and other higher centres are highly connected to the CPGs, receiving efference copies of the signals sent from the CPGs to the motoneurones, as well as the afferent feedback from the muscles (Grillner, 1985), and that this allows fine-tuning of the co-ordination without which the walking looks a bit rough and is more likely to break down. However, we do not know the details of how the neurones are connected, either inside a CPG or inside or between limbs, and certainly not how the cerebellum works (though theories have been put forward (*e.g.* Miall *et al.*, 1993)).

What we can deduce from this is that a low level dynamical system exists which links the muscles to produce walking and which has a highly regular structure consisting of several very similar neuronal networks (CPGs). These are strongly interconnected to maintain their rhythm and are controlled in turn by higher centres to maintain their stability.

Much of the discussion in this section has been about dynamical systems and it is the contributions of researchers in this field that we shall discuss next.

2.2.3 Group theory, non-linear dynamics and co-ordination

Researchers from various areas of mathematics have studied walking. Collins and Stewart used group theory to analyse the properties of various coupled non-linear oscillators. They predicted that fixed CPGs should be capable of changing between gaits by varying very few parameters (Collins and Stewart, 1993a,b). Collins then went on to test this with a selection of CPG models and found that it was generally possible to make simple CPGs produce different gaits by varying only a few internal parameters whilst leaving the connectivity unchanged (Collins and Richmond, 1994). This was a simpler solution than many previously proposed which suggested, for instance, that different co-ordinating neurones might be needed for different gaits (Grillner, 1985).

There has also been work by non-linear dynamicists like Kelso. He has studied how walking systems behave, what happens to the co-ordination between legs at gait transitions, and how the system converges to its stable attractor (a particular gait). He particularly stresses that gaits are selected (and are most stable) in animals when they are the most efficient for travelling at the desired speed, though some hysteresis stops switching back and forth at transitional speeds (Kelso, 1995). He suggests that we should build this kind of nonlinearity and multistability into robots to help eliminate problems with redundancy. He also points out that transitions normally occur very smoothly, but in animals with their higher centres removed instability and critical fluctuations occur and gaits switch back and forth near bifurcations.

Gallagher and Beer look directly at evolved locomotion controllers from a dynamical systems perspective, examining controllers evolved to be reflex chains, central controllers (without feedback) and CPGs (Gallagher and Beer, 1992). They examine the basins of attraction of the limit cycles and fixed points. They discover that the reflex chain controllers on their own have no limit cycles, but rather whatever the current state of the system, it is attracted to a fixed point forward in the walking cycle: this results in the whole (controller-body) system having a stable walking limit cycle, but this will break down if the feedback is interrupted. This behaviour is much like finite state machine walking controllers. The other two types of controllers both have inherent limit cycles which correspond to walking in the whole system, but whereas the central controller has a crude limit cycle which cannot adapt, the CPG is entrained to the frequency of the rhythmic feedback from the legs, so adaptation to different conditions does not have to be learnt, but rather emerges from the dynamics of the controller-body system. In a more heavily mathematical analysis, Cheng and Lin (1996) look at the stability of a biped using a linearised Poincaré map and discuss the robustness of the locomotion.

2.2.4 Evolving robot controllers

As more people started to built legged robots, they began to look for ways to automate development of the controller for them. The most popular method has been a form of simulated evolution[4].

As many different forms of evolution have been tried as there are researchers, but generally they tend to evolve parameters for some kind of neural controller for the robot: normally the strengths of connections, but sometimes other internal parameters

[4] for a description of Genetic algorithms, see Goldberg (1989).

of the neurons as well[5]. In general this has proved a step backwards for the sophistication of walking controllers with most of the controllers evolved being statically stable. However, a popular way of evolving controllers is to simulate the robot concerned and evolve a controller for it on the computer, and it is a very computer intensive process to model a robot dynamically, so it is possible that this is partly to blame. If so, as computer power increases and since fast dynamic simulation software is now available (*e.g.* McMillan, 1994), hopefully this problem will go away.

Lewis *et al.* (1992) evolved controllers for a hexapod robot (Rodney) which they had built. The controller was evaluated on the real robot and learnt to walk with insect-like gaits after a staged evolution where it was encouraged a bit at a time towards the final goal of walking.

Beer and Gallagher evolved the parameters of a dynamic neural network to control a (statically stable) insect walking in simulation (Beer and Gallagher, 1992). They then went on in (Gallagher *et al.*, 1996) to evolve a statically stable controller for a simulated robot, and then transferred the evolved controller onto a real robot with no problems. Spencer (1994) used Genetic Programming[6] to evolve the architecture as well as the parameters of a similar robot.

Karl Sims, on the other hand, evolved dynamic controllers for robots whose morphologies were evolved at the same time (Sims, 1994b). This produced arbitrary shaped robots with controllers which would allow them to perform behaviours like tumbling, sliding, jumping, and swimming. They had the advantage of not having to remain balanced as is usual for walking robots, and it is not clear how they would have performed if that had been what was evolved; but they are the most visually impressive result of this general approach.

2.2.5 Vertebrate locomotion and development

Meanwhile, zoologists have been looking for common ground between different animals when walking, examining which criteria appear to be being optimised, and how loco-

[5] for a general review of this, see Kodjabachian and Meyer (1995), but also see next chapter.

[6] see Koza (1992).

motion in vertebrates has developed over time. They call this first problem the inverse optimality problem — *i.e.* what was being optimised to produce this?

Alexander has written extensively about locomotion: in (Alexander, 1984) he looks at locomotion in reptiles, birds and mammals. He concludes that locomotion with a similar Froude number (a dimensionless measure: $speed^2/(gravity * hipheight)$) produces dynamically similar movements in general right across different species (0.1 walk, 1 trot/pace, 2-3 asymmetric). He also looks at the inverse optimality problem, and concludes that for turtles displacement (*i.e.* roll, etc.) is minimised, but humans minimise work. Tendon elasticity helps in this regard by storing strain energy rather than allowing it to be dissipated as heat. In (Alexander, 1990) he proposes the use of springs in robots to replace tendons in their job of minimising energy loss, and also on the feet to soften impact slightly to reduce 'chattering' and hence improve grip. In (Alexander, 1991) he looks at how energy is saved in terrestrial locomotion, through tendons, but also through aligning joints to minimise the amount of work that has to be done and the amount of conflict between muscles.

Eilam (1995) studies how movements change both across species and during ontogeny (development in individuals of one species), and points out that there is the same consistent progression in each from simple lateral movements of the trunk, to use of limbs, through to vertical movements of the trunk.

Work being done on development of locomotion includes that by Vaal *et al.* (1995), who detail a research agenda for studying human locomotion and gaining insights into its development. They claim that very little work has been done on the ontogeny of locomotion, and they want to identify the crucial subsystems and their interactions, and how these develop to produce an adult locomotory pattern. They discuss CPGs and the importance of feedback, and then move on to the development of walking, starting with precursors like reflex stepping going right through to integrated walking. Unlike most authors, they disapprove of the concept of optimality criteria as they consider them arbitrary, and believe the introduction of functionality constraints should be sufficient, namely that walking must work in a wide variety of situations. They also mention the usefulness of using muscles when modelling locomotion, as they claim that such models have inherently better stability properties than force control for joints.

van Soest and van Galen (1995) look at how animals reduce redundancy problems in multi-joint movements by imposing constraints. They divide these into physical and self-imposed constraints (they believe the latter are imposed specifically to help solve the problem). And finally, Assaiante and Ambland (1995) look at the ontogeny of balance control.

2.2.6 Functional Neuromuscular Stimulation

One of the most exciting developments recently has been work done to restore the ability to walk to paralysed patients by electrically stimulating their muscles (*e.g.* Marsolais and Kobetic, 1983). Research done by Yamaguchi recently shows that generating the appropriate patterns for walking is essential for optimal use of muscles which have been weakened by the paralysis, and that ankles are a particular weak spot which might well benefit from some kind of orthosis (Yamaguchi and Zajac, 1990).

2.3 Where are we now?

We have now looked at research up to the present day, and in this section we shall look at the commercial products which have come out of this, and at what ongoing research uses legged robots.

2.3.1 Commercial robots

In recent years various projects have been proposed for which walking robots would be 'invaluable'. However, the number which have actually been built and used for anything other than walking research is much more limited.

Walking machines have long been proposed for travel over rough terrain that even a tracked vehicle could not navigate, and for dangerous environments where human would be at risk.

To that end, Dante and Dante II were designed to descend into volcanoes to study conditions in an environment too dangerous for humans and too rough for other robots (Wettergreen *et al.*, 1993). They were not unqualified successes. Dante II was the more successful of the two. It was an extremely stable eight legged robot which moved slowly down into Mount Spurr in Alaska. Unfortunately on its way back the ground collapsed under it, and it was not able to cope and had to be airlifted back out.

NERO was a climbing robot funded by Nuclear Electric in Britain and designed for work on a nuclear reactor pressure vessel where humans have huge safety problems (Luk *et al.*, 1994). It was designed to climb the outside walls of the pressure vessel and inspect its condition, clean it or even install equipment on it, but was too light too do heavy work. It had drawbacks (it travelled at a speed of 0.1 m/min, and had to be placed on the wall of the pressure vessel, it couldn't walk there), but it was successfully used for these purposes. Nuclear Electric together with Electricité de France, CERN and the Italian electricity board then funded work on a next generation of climbing robots which would be more versatile, stronger, and capable of walking to the reactor vessel and performing the floor to wall transition itself (called Robug, see Luk *et al.*, 1993), but there have been problems getting these to work to specification (for instance Robug III is too heavy to support its own weight for long when climbing).

Other commercial robots include a "walking harvester" built by PlusTech Oy. in Finland (see Plustech Oy., Finland). This robot has been built to replace wheeled vehicles for forestry work as it doesn't damage the ground and hence minimises the risk of soil erosion on steep slopes. It also has a high and variable ground clearance and so can move over obstacles which would block most other vehicles. It is driven by a driver in a cabin on top of the robot.

Much more recently some Japanese firms have come into the market. Honda recently revealed their biped robot (which they have been working on for many years). This can walk semi-autonomously or be teleoperated and can carry out simple tasks (Honda Motor Company Ltd., Tokyo). This is a very impressive robot, and presumably further developments will come from them in the future.

In a completely different field, Sony have now released their robot puppy, Aibo (Sony Corporation, 1999), which is being marketed as a toy. This can perform a variety of stereotyped actions like rolling over and standing on its hind legs and boxing the air, as well as being remote controlled by its owner.

Some other fun robots, which are currently being finished, are the walking dinosaurs being constructed for use in museums round Europe. The project is called Palaiomation and is funded by the EU[7].

2.3.2 Research Robotics

There is a lot of research currently studying the problems of coping with difficult terrain, and avoiding falling and recovering afterwards. For instance, Boone and Hodgins have recently been looking at how bipeds can recover from slipping or tripping (Boone and Hodgins, 1995, 1997) despite having little information about what is going wrong. Yoneda *et al.* (1996) have built a robot (Titan VI) capable of travelling at a reasonable speed on flat ground (over 2 mph), but which can move over obstacles, and of course Honda's robot (Honda Motor Company Ltd., Tokyo) can walk up and down steps and avoid obstacles very effectively at a fairly slow speed.

2.3.3 Modelling humans and other animals

The other area where legged robots are being used is in modelling work, examining how animals move by modelling them in simulation (or sometimes on robots), and also just modelling their behaviours to create impressive graphics for films.

Modelling roughly divides itself between creating computer graphics and simulating real locomotion. In the former, animating robots, people and animals is becoming more involved with simulation as the computational expense of doing the extra modelling becomes less important and techniques become more sophisticated. In the latter, simulating real animal and human locomotion helps us gain insight into how control of movement happens in the body and provides a source of ideas for robot controllers.

Simulating human and animal locomotion

Roboticists have collaborated a great deal with zoologists looking at the energetics and common principles of locomotion, as well as with neuroscientists, and with ethologists studying animal behaviour.

[7] Brite/Euram Craft CR 1651

Grillner and colleagues did some very careful modelling of CPGs in lampreys to see whether what they found could explain the behaviour they saw (it could). The simulation work is described by Ekeberg (1993). Other work includes that by Cruse and others at Bielefeld in modelling stick insects using a real robot (Cruse *et al.*, 1995), and comparing its behaviour to their neural network based model of the insect controller; Taga (1995) has looked at how using a sufficiently sophisticated neural model can allow behaviour which can be quantitatively compared to human locomotion to emerge on a simple simulated biped; and Lewis (1996) has looked at how the transition from swimming to primitive walking gaits may have occurred using real and simulated robot models.

Research in Edinburgh has examined CPGs to see how well they can be modelled artificially for controlling swimming movements (Ijspeert *et al.*, 1997), and simulating a variety of legged robots to see what general principles can be used to build controllers for them (Reeve, 1999a).

One new element may be work on Functional Neuromuscular Stimulation, where researchers like Yamaguchi and Zajac (1990) admit that their current methods for selecting stimulation patterns to create gaits in paraplegic subjects are definitely suboptimal. It is quite possible that this biomechanics work can be combined with artificial intelligence optimisation techniques, which are already applied to designing controllers for walking robots, to improve the efficacy of this technique.

Human modelling has become an attractive target, and several people have investigated it. For example, Hodgins has simulated a 30 degrees of freedom humanoid figure, and got it to run in a fairly biomechanically accurate fashion (Hodgins, 1996). Playter has looked at gymnastics, and has modelled different manoeuvres to see how stable they are, and explore the best way of stabilising them (Playter, 1994). He even managed to get a unpowered robot to do a layout somersault! Jalics and others at Ohio State University have started investigations into dancing with a simple planar model of a biped, examining how to keep track of the rhythm of the music and move the body in time with it (Jalics *et al.*, 1997).

Modelling creature dynamics

This is moving more towards modelling for the fun of it, and much work has been done in this field (Jurassic Park for instance!). Until recently most graphics for games and films were produced using kinematic techniques, and simulation has been largely ignored as a waste of time, especially as the results were much cruder than could be achieved by kinematic techniques. However, researchers are beginning to realise that it is no longer quite so time consuming, and work has been done to combine the techniques, tuning gaits which were created in other ways in a simulator so that they are dynamically plausible, with the expectation that this will improve the appearance of the movement (*e.g.* van de Panne, 1996). Work like (Ko and Badler, 1996) looks at how to generate the right movements to stay stable whilst also maintaining the realistic gait produced by the animation, a crucial point as the gait produced must be perceived as a normal walking pattern for it to be useful for their purposes.

2.4 Conclusions

There are many areas where an automated process for designing controllers for legged bodies would be very useful. These include such diverse uses as producing believable walking in computer graphics applications and virtual reality, modelling human locomotion to develop activation patterns for Functional Neuromuscular Stimulation, as well as the more obvious application of making the choice of using walking robots, for any task, that much more practical and straightforward (consider that the designers of Dante II, who are planning on landing the first private expedition on the Moon in the near future, have decided to use a wheeled robot for the task). In the next chapter we will look at the methods which could be used to automatically build such controllers.

Chapter 3

Neural Networks

In the last chapter we looked at the history and state of the art in walking research, and we concluded that a method for automatically generating low-level controllers for legged robots would be very useful for the development of the field. Now we will examine the kind of methods which are appropriate for this task.

3.1 Building controllers

We will look at what kind of controllers are appropriate for this problem and then at what generation methods may be feasible for these controllers; we will then assess a variety of systems against the criteria we have produced and select those which seem the most appropriate for further investigation.

3.1.1 Types of controllers

Generating controllers for legged robots is a difficult task, but several points are clear:

- There is a lot of symmetry in robot design, and identical joints are used in more than one place on any given robot (*e.g.* the hindlimbs of a robot are almost invariably mirror images of each other). Consequently an ideal controller is likely to be highly degenerate, and any method for building controllers should take account of this.

- In vertebrates, control of the legs to generate a basic walking pattern is a function of the spinal cord, where Central Pattern Generators actively alter the dynamics of the legs through the muscles to create a new dynamical system where walking is a stable attractor (Grillner, 1985); this creates a walking animal which is in turn controlled by the higher centres in the brain. Because there is a detailed understanding of Central pattern Generator (CPG) control of locomotion (see Section 2.2.2), it would be possible to transfer a lot of knowledge from neuroscience into a controller like this for walking robots. A basic requirement for this is a basic building block with its own continuous dynamics like neurones[1] in animals.

- Artificial Neural Networks (NNs) are used for control in a broad range of walking robots (see Section 2.2.4 for examples), and indeed throughout robotics and even in control engineering (*e.g.* Narendra and Parthasarathy, 1989); because of this they are a very well researched computational system and there are a large number of methods available for training them. Although most neurons used in these fields do not have their own internal dynamics, some are particularly designed with this is mind, and so would be appropriate for this kind of task (*e.g.* Wallén *et al.*, 1992; Taga *et al.*, 1991; Beer and Gallagher, 1992; Kodjabachian and Meyer, 1998)

For these reasons we will use Neural Networks as controllers for our walking robots. Two issues now arise: what kind of neurons to use in the network, and how to train them.

3.1.2 Types of Neurons

Traditionally, neurons in Artificial Intelligence have been idealised for the sake of mathematical tractability to produce threshold and sigmoidal neurons, where the output of a neuron S_i is a simple function of its inputs; the sigmoidal neuron, for instance, is governed by the following equation:

[1] I refer to biological *neurones* and artificial *neurons* throughout this thesis.

$$y_i = \sum_{j=1}^{n} w_{ji} S_j \tag{3.1}$$

$$S_i = \frac{1}{1 + e^{-y_i}} \tag{3.2}$$

where: w_{ji} is the weight connecting neuron j to neuron i, and
y_i is the internal state of neuron i

The analytical tractability of this model has made it particularly easy to train feed-forward networks using gradient descent algorithms like backpropagation (Rumelhart and McClelland, 1986) or more sophisticated ones such as conjugate gradient descent (Bishop, 1995, ch.7), and indeed they have been used in the past for control of insect-like walking robots. However, neural networks based on this have no internal state or continuous dynamics, and training recurrent networks (which could offer some kind of memory indirectly) is much more difficult, and so they may not be appropriate for the task of altering the dynamics of the legged robot to make walking a stable attractor.

Many other neural models exist with continuous dynamics, however, and they are capable of a much richer variety of activation patterns than these stateless neurons. For instance Continuous Time Recurrent Neural Networks (CTRNNs) were devised by Beer (1995) (although they are based on the common leaky integrator model) for exactly this purpose, as controllers for a (statically stable) walking robot: indeed, they were chosen for this by Beer precisely because they showed a much richer behaviour than discrete neurons — they are governed by the following equations:

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{n} w_{ji} S_j \tag{3.3}$$

$$S_i = \frac{1}{1 + e^{(\theta_i - y_i)}} \tag{3.4}$$

where: θ_i is a bias term, and throughout these equations
τ_i is the adaptation rate of the neuron

This neural model is governed by a simple first order differential equation but is nonetheless capable of a surprisingly rich variety of behaviours when connected in a network (*ibid.*). However, we can see that in the specfic case where $\tau_i = 1/\delta t$, δt being the timestep of the integration, we get the same behaviour as for the sigmoidal neuron, and indeed in general these are a continuous time version of the sigmoidal neurons above. More complicated models exist, including this one from Taga (1995):

$$\begin{aligned}
\tau_i \frac{du_i}{dt} &= -u_i - \beta_i \max(0, v_i) + \sum_{j \neq i} w_{ji} S_j + u_0 && (3.5)\\
\tau'_i \frac{dv_i}{dt} &= -v_i + S_i && (3.6)\\
S_i &= \max(0, u_i) && (3.7)
\end{aligned}$$

where: u_i and v_i are the internal state of neuron i

Again this is a fairly simple neural model, governed by two coupled first order differential equations (effectively a second order ODE, and so capable of more interesting behaviour like oscillations), and again it has been used for control of a walking robot, this time a dynamically stable but two dimensional biped: indeed, stable walking was achieved, so this seems to be a promising model. Another step in this direction is a model by Wallén *et al.* (1992), described below:

$$\begin{aligned}
\tau_i^D \frac{d\xi_i^+}{dt} &= -\xi_i^+ + \sum_{j \in \Psi_+} w_{ji} S_j && (3.8)\\
\tau_i^D \frac{d\xi_i^-}{dt} &= -\xi_i^- + \sum_{j \in \Psi_-} w_{ji} S_j && (3.9)\\
\tau_i^A \frac{d\theta_i}{dt} &= S_i - \theta_i && (3.10)\\
S_i &= \max(0, 1 - e^{(\Theta_i - \xi_i^+)\Gamma_i} - \xi_i^- - \mu_i \theta_i) && (3.11)
\end{aligned}$$

where: $\xi_i^{\pm}$ and θ_i are the internal state of neuron i, and
$\Psi_{\pm}$ is the set of all excitatory (inhibitory) inputs, and
Γ_i and μ_i are bias terms.

This is a third order model, and has an even richer behavioural repertoire, with individual neurons being capable of a variety of different oscillatory responses to tonic excitation. Indeed neurons of this type were successfully used to model the CPGs in a lamprey spinal cord in an investigation of the neuronal networks controlling swimming (*ibid.*).

Many other increasingly sophisticated neural models exist, including extremely realistic multi-compartmental models, but these are likely to be too computationally intensive for use in this project. Overall, it is unclear which neural model will be the most effective: all except the last have been used for this type of task before, but the ability of neurons to generate a greater range of behaviours individually seems useful, and so we will investigate them all in our experiments.

3.1.3 Methods for training Neural Network controllers

In order to make a NN controller, or indeed any other controller, we must first define how the controlled system should behave. This means that we know in advance what we want the NN to do, so we can use a supervised learning algorithm to teach it. In cases where we know *quantitatively* what is required we may be able to use inductive learning with a gradient-descent type algorithm such as Backpropagation (Rumelhart and McClelland, 1986) which will allow us to train the network very effectively. Unfortunately, for a task such as walking there is no definitively correct answer for the question of how to walk (or at least not one that we can determine), so we have to rely on more qualitative measures: for instance a robot might be judged according to how far it travels whilst keeping its body off the ground, or how little energy it expends to move a certain distance.

This more difficult problem requires a reward based approach, like Reinforcement Learning where NNs are trained by allocating blame when something goes wrong, and rewarding correct actions (Sutton and Barto, 1998). One difficulty with this type of approach is to determine which part of controller is doing well or badly, and hence which to reward or punish — this is the Credit Assignment Problem (*ibid.*). There are many different solutions to this, but one which is useful for particularly intractable cases are the family of reward based approaches collectively known as Evolutionary Algorithms[2]. These avoid the credit assignment problem altogether by dealing with populations of controllers, and rewarding or punishing individuals as a whole rather

[2] see Goldberg (1989); Koza (1992) for example.

than the more common approach of dealing with a single individual and assigning credit to its components. They are by definition a cruder technique than than either Inductive or Reinforcement Learning, but in this case they offer the only practical solution, and so it is these which we shall use in our experiments.

Evolutionary Neural Networks

The common thread running through all of these methods is that they use Genetic Algorithms (or a relative such as Genetic Programming) with some encoding of a NN as the genotype to evolve the desired NN. These NNs are then tested for suitability and more individuals are created from the most able genotypes whilst the least able are discarded.

The genotypes can encode either or both of the architecture and the weights of the network. Those that do not encode the weights have to learn them separately however, and since this will require another learning technique, typically a gradient-descent type method, and since we have already said that this is inappropriate for the problem we are tackling here, we will only concern ourselves with those genotypes which *at least* encode the weights of the network. Those that do not encode the architecture must rely on it being fixed, for instance as a fully connected recurrent network, or as a specific hand-crafted layout. We will discuss the advantages and disadvantages of this later.

3.1.4 Selection criteria

There are many criteria which should be considered in choosing an encoding for the NN controllers:

Reusability As was mentioned above, it is clearly desirable for one leg of a robot to be controlled in a similar or indeed identical fashion to another leg of the same robot, so it should be possible for subnetworks to be reused in different parts of the controller to avoid wasted effort building different subnetworks to achieve the same task.

Modularity In animals we know that CPGs are associated with individual muscles and joints and limbs, and are tightly coupled to the sensors and muscles which they control. In our robots it is reasonable that there should be more direct connections between sensors and actuators on the same joint than with anything else. This can easily be achieved for instance by assigning CPGs to joints and making more local connections (inside CPGs and to local sensors and actuators) than distal connections (to other joints or limbs). This might also be desirable in our controllers because it would tie in well with the previous item, providing modules which can be easily reused.

Bias It is inevitable that any encoding will show bias towards certain types of network — it should be possible to examine the kind of networks which are likely to be created and eliminate or alter those encodings which tend to produce configurations which are unlikely to succeed.

Chromosome Size The space to be explored increases exponentially with the size of the chromosome, thus potentially making the problem much harder, so more compact encodings may be more desirable than larger ones.

Completeness Some encoding schemes are not complete (*i.e.* there are networks which cannot be encoded), and it is possible that the solution required is one of these networks, so in the absence of other considerations this should be taken into account.

We will now examine a variety of encodings and consider how they stand up to these criteria.

3.2 Evolutionary Neural Networks

There are many ways of evolving neural networks, but the most significant difference between different methods lies in how the NN is encoded into the genotype. There are basically two ways of doing this — directly, so that every weight and connection in the NN is recorded explictly in the chromosome (and the subset of these where the network architecture is fixed, and only the weights are recorded), and indirectly, where commonly some grammar or developmental rules are used to translate from the chromosome to the NN. Many of these latter are production rule systems where each rule describes what one symbol (the left hand side) becomes after another developmental step (for instance two new symbols). This goes on, depending on the system, until all the symbols are terminal symbols (*i.e.* neurons or connections), or for a fixed number of steps, after which all the symbols are translated into terminals according to some separate translation scheme. The latter encodings are often called L-systems, named after Lindenmeyer, a biologist who first used them to describe the development of artificial plants (Lindenmeyer, 1968).

Both types of encoding have problems — Direct encoding methods suffer for two main reasons:

Modularity and Reusability Groups of neurons (subnetworks) which have a useful function are likely to be spread across the whole chromosome, so crossover will tend to break them up; also because each connection and weight is recorded separately, when a useful subnetwork does form, there is usually no mechanism for duplicating it when the problem is degenerate.

Chromosome Size Because every weight and connection is recorded separately, the chromosomes become impractically long: n neurons require n^2 weights if they are fully connected, and these weights are often real numbers.

This first problem would be extremely serious for my purposes, because as we have said before, these are primary considerations; also as we discussed in Section 3.1.1, the solution is likely to be highly degenerate. The chromosome size may also be a problem, due to practical time constraints. If we are to use a direct encoding, we must therefore come up with a solution which at least gets around these problems of modularity and reusability. Indirect encodings tend to be designed specifically to avoid these problems (though some still suffer from the first). As a result other problems occur:

Completeness Some of the encoding schemes are not complete, and networks which are not representable might be better than those which are.

Bias Even schemes which are complete bias the networks greatly in one direction or another: after all they contain exactly the same information when they have been decoded as the direct schemes, so there would be no advantage in using them if they did not do something extra such as make it easier to encode modular networks. Generally the bias is towards some kind of regular structure to the network, but whatever it is, it may be just as damaging as incompleteness if the optimal structure becomes very difficult to express.

These seem on the whole to be less serious problems, but it is clearly very important to choose the right encoding scheme for the problem to be solved, since what is a good scheme for one problem may make another impossible to solve.

3.2.1 Direct Encodings

Fixed Architectures

There are several, mostly old, experiments which have been done with fixed architectures. They tend to try to solve very simple problems such as XOR and n-bit adders, for instance those by Whitley and Hanson (1989). An example of a fixed architecture encoding is shown in Figure 3.1. They were found to be faster than Backpropagation for large NN problems at that time, but the architecture does have to be hand-coded, which is a problem. This seems to be a very primitive solution, and suffers from problems of reusability, modularity and chromosome size, and bias towards massively interconnected networks in its most primitive form (a fully connected network): however, if it were possible to automatically generate an appropriate architecture there is no reason why this should not work. This transfers the problem to one of generating a network architecture which is modular and involves reuse of subnets: we will discuss this shortly.

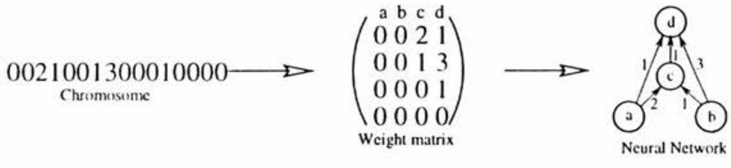

Figure 3.1: A standard direct fixed architecture encoding

De Garis evolved controllers for a simulated legged robot (LIZZY) by designing control structures by hand, and then evolving the connections of small parts of them to achieve desired subtasks (de Garis, 1990a,b). This avoids a lot of the issues discussed here through detailed hand design, but this would not be possible for a general purpose system for arbitrary robots.

One more sophisticated fixed architecture NN was developed by Lewis, Fagg and Solidum — it was a controller for a real hexapod robot which was evolved by Staged Evolution (Lewis *et al.*, 1992). Staged Evolution involves intermediate products being evolved on the way to the desired goal: in this case first the weights for a 2-neuron oscillator were evolved, then one was put on each joint of the robot, and then the inter-joint connections were evolved to make the leg move correctly, and then the inter-leg connections were evolved to make the legs coordinate properly. This was fine for the problem of generating a controller for this particular robot, solving all of the problems mentioned above at a stroke, but it involved a detailed knowledge of exactly how a joint or a leg should move, which will not be known in the general case.

The common thread among these more sophisticated solutions is that considerable work went into hand-coding the architecture of the networks so that it would be possible to evolve the appropriate controller. This is a fatal disadvantage for our task: it relies on work being carried out on each robot to determine, usually by trial and error, what the best design for this particular network would be; this is something we are explicitly trying to avoid, and so it will not provide the general purpose tool we desire.

However, it may be possible to take some of the ideas from this and put them into a more general framework: what we require is knowledge about the robot itself to determine the architecture for the network. This approach is used by Kodjabachian and Meyer (1998), where he uses information about the design of the robot to calculate on the structure of the network. However, if we can extract this information from the description we are given of the robot automatically (*e.g.* from the simulator description of a robot being modelled), then we may be able to generate a network which reuses, for instance, the same subnetworks on each leg of the robot, or only connects small groups of neurons to each sensor/actuator pair, without requiring the user to have detailed knowledge of how the system works. This would potentially eliminate the problems of reusability and modularity which we have discussed, and because we would still be producing complete controllers, it would avoid problems we discussed with staged evolution.

Variable Architectures

A lot of direct encodings with variable architectures only encode the architecture and not the weights, in a very similar fashion to figure 3.1 but using connection matrices with 0's and 1's for no connection and learnable connection respectively instead of weight matrices. For instance Miller *et al.* (1989) use a GA to evolve the architecture and then Backpropagation to learn the weights. Unfortunately, as we have already said, we cannot use this, but there are several which learn both architecture and weights (though many have a fixed network size).

Maniezzo (1994) manages a direct encoding of architecture and weights in the simplest way possible by combining the two previous techniques (see Figure 3.2) with each position on the chromosome containing a connection bit to say whether the link exists as well as a weight for it if it does. As with previous techniques there is no way for subnetworks to retain their integrity or to be reused, and the network size is fixed. This a very similar case to the fixed architectures, and suffers from the same problems. However, it may be possible to use the same technique of automatically decomposing the problem to force reuse of components to avoid some of these difficulties.

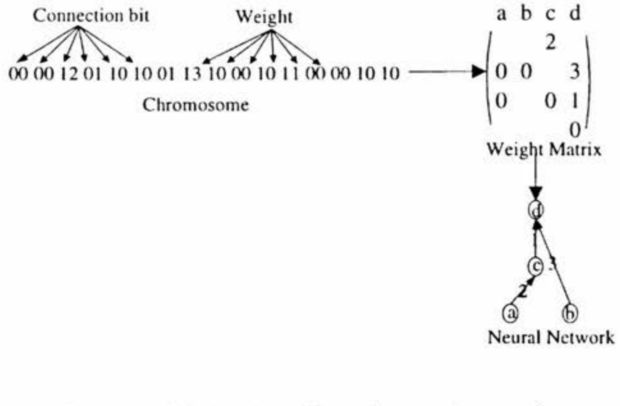

Figure 3.2: Maniezzo's weight and connection encoding

Torreele, on the other hand, has a relative connection structure which could allow reuse of subnets — the neurons are ordered on a grid and connections are allowed to a (prespecified) set of neighbours (Torreele, 1991). Each neuron has an associated bit string which determines whether connections exist, and if so, whether they excite or inhibit. Because the addressing is relative subnets can be reused by copying to a different position on the grid (although Toreele has no mechanism for achieving this). However, there are strong restrictions on the connectivity imposed by the encoding — there can be no long distance connections — which may cause problems. Although this seems superficially to be biased towards the kind of networks we are interested in (locally connected neurons, some possibility of reuse of subnets), the impossibility of distal connections make it a very incomplete encoding, and as it is a distinct possibility that some long distance connections may be very useful in locomotion (*e.g.* in bipeds it is important not to bend one knee when the other leg is off the ground), this is too serious to overlook.

Collins and Jefferson take another approach and treat the connection as the basic element rather than the neuron, and directly encode them in the form *From:a To:c Weight:-1*, which they call K connection descriptors (Collins and Jefferson, 1990). This has the potential to be the first of these techniques to have a variable number of neurons, by allowing the connections freedom to mutate out from their original limits for instance: however, they do not take advantage of this, and indeed even keep the number of connections fixed as well so that they can use straightforward crossover. The result of these decisions is that the architecture is quite badly constrained, but this is not necessary. An advantage of this encoding in general is that subnets can be encoded in very short strings and thus are less likely to be broken up by crossover. However, there are disadvantages — it is still not possible to reuse subnets, and because there is no physical location on the chromosome for any particular neuron, it is likely that different chromosomes will have the same nodes' connections in different places, so even if a subnet remains intact there is a much greater chance of interference with its operation during crossover from other connections being added. This is caused partly by the massive redundancy in the encoding which is on top of the competing conventions problem already inherent in NNs (several different networks can have the same structure by just changing which node has which name).

However, Wieland has evolved pole-balancing controllers (including multiple and jointed poles) using exactly this technique, at least some of the time with considerable success (Wieland, 1991). He also considered the 2-legged walker as an extension of the jointed pole, but unfortunately only had limited results.

Angeline uses a very different approach in (Angeline *et al.*, 1993; Angeline, 1993) to evolve recurrent neural networks. He uses mutation alone in a quasi-Simulated Annealing technique he calls GNARL. Since only mutation is used encoding is almost irrelevant, because changes are done all the time on the phenotype itself. Various different mutation operators are used (weight change, connection addition/removal, node addition/removal), the severity of which depends on the temperature which cools as the fitness increases. Some of the operators are shown in Figure 3.3. The technique is also closely related to constructive/destructive NN algorithms but claims superiority over them because they are monotonic, only allowing 1 neuron to be added or removed at a time, whereas his can do many — the advantage of this is that it can get out of deeper local minima (in the same way as Simulated Annealing). It was found to be effective on a variety of problems. Overall the technique avoids the problem of losing subnets by not allowing crossover, and is complete with regard to network architectures, allowing any number of nodes/connections (although you can bias the starting population), but again it cannot reuse existing subnets. However, with the system discussed in section 3.2.1 to force reuse of components, this could be very promising.

Utecht and Trint (1994) describe and compare a variety of other mutation operators which could perhaps be used in this method to enhance it further.

3.2.2 Indirect Encodings

For the most part the indirect encodings contain both the architecture and the weights, but, as with fixed encodings, some of them use Backpropagation to learn the weights; for instance Harp, Samad and Guha use "Blueprints" in (Harp *et al.*, 1989), and Boers and Kuiper use an L-system to develop the architecture in (Boers and Kuiper, 1992; Boers *et al.*, 1993).

However, most indirect encodings do encode both, using generally either a grammar

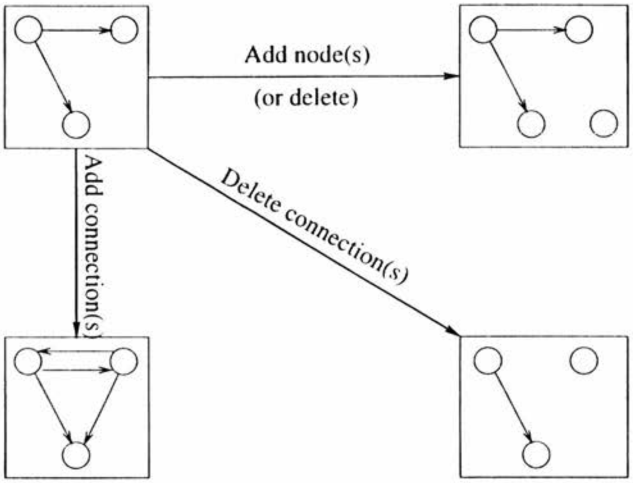

Figure 3.3: A network in the GNARL encoding and the effects of some mutation operators

or developmental encoding of some form. I shall go onto those after I have described another technique which is less popular.

Fullner and Miikulainen used a Marker-based encoding scheme (Fullner and Miikulainen, 1991), loosely based on the marker structure of biological DNA. Each node has a key with which it is associated, and a series of other keys which it is connected to (or the closest match if that key is not present) — each of these keys has a weight associated with it; this allows arbitrary networks to be described. The markers referred to are start and end markers on the chromosomes which border the segments where neurons are defined. In other words there is no fixed point where the neuron definitions are, and there is even unused genetic material between end and start markers.

The rest of the encoding can however be separated from these markers, and Michel and Biondi have done just that in proposing a very similar scheme with a more straightforward chromosomal representation (Michel and Biondi, 1995a,b) which is inspired by protein synthesis regulation. Each node now has a set of signals with which it is associated and inhibitor and activator signals which will connect to it if they are produced. The connection weights are now only ± 1 and neurons can have a greater variety of inputs as they can have more than one input signal, but otherwise it is very similar. Neither of these methods have any possibility of reusing subnets because the keys/signals would all have to be changed if the new network were to not interfere with the old.

These two techniques have exactly the same expressive powers as GNARL, described in the previous section, but are much more complicated. The advantages they claim are in robustness under crossover, which GNARL does not employ, but it is unclear whether their results justify the increased effort in using them.

A successor of this approach which is at the same time very different is the symbiotic evolution approach by Moriarty and Miikulainen called SANE — Symbiotic Adaptive Neuro-Evolution (Moriarty and Miikulainen, 1994). Here labels (keys again) determine the connectivity, but each chromosome represents only one neuron, and collections of chromosomes are put together at random to form a network to be tested for fitness. This very cleverly stops convergence on the gene pool, since diversity is essential as a variety of different neurons are needed to make an effective neural network. However, the experiments done with it were on feedforward NNs with only one hidden layer, so individual neurons could realistically have separate domains of expertise: it is difficult to see how the experiment could be expanded to multi-layer or recurrent neural nets as it stands because few neurons in these nets can be of any use in and of themselves.

Development encodings and grammars

Mjolsness *et al.* (1987) and Kitano (1990) came up independently with similar L-system schemes: each symbol produces a $2 * 2$ matrix of symbols. This is iterated a number of times until there is a $2^n * 2^n$ matrix which is then translated by a separate mechanism into a weight matrix. Kitano calls this a graph L-system. This scheme tends to produce extremely regular NN structures, and thus constrains the network a lot, although, if there are enough symbols, the scheme could theoretically be complete. The process is also highly epistatic and changing any of the rules is likely to change the fitness significantly, because symbols which appear early control the eventual shape of very large sections of the NN, and ones which occur late are likely to appear in a lot of places, and thus again change a large section of the networks.

Both Mjolsness and Sharp, and Kitano have since moved on from this to much more complicated methods, both involving very sophisticated development schemes. The former (Sharp *et al.*, 1991) is still in this field, and involves a cell division process which eventually produces neurons with individual state vectors which are much like the keys above, and must be matched to make connections, but the latter (Kitano, 1995) is more computational neuroscience, with cell division occurring according to how high the metabolic rate is in the cell, axon growth determined by Nerve Growth Factors, and many other equally complex effects.

These are all interesting ideas, although none of them produce bias in exactly the direction desired for this control problem, they do solve problems of modularity and reuse, and so it is entirely possible that a different L-system based grammar would be capable of generating controllers for legged robots.

Gruau has a grammar scheme based on Genetic Programming rather than GAs which he calls cellular encoding since the rewriting process happens to cells rather than symbols (Gruau, 1994a,b), and which he uses for feedforward boolean[3] neural networks. In the model each cell has a copy of the chromosome which codes the process, and reads it from a different position. The chromosome is a tree with ordered branches whose nodes are labelled with instructions. The instructions act on a cell or an input to the cell. Each step in the process involves a cell reading an instruction and moving down the tree to the next node. The instructions can be for instance to divide, change its links, or become a neuron. For example when a cell is given the instruction S — sequential division — it divides, with the first child getting the input links, and the second getting the output links from the mother cell, with a connection between them. The nodes have 0, 1 or 2 branches depending on whether the command was to make a neuron, to alter an existing cell, or to divide into two cells, the branches being followed by the new cells. In this fashion branches of the tree form mostly separate subnetworks and crossover can move them about. Gruau also proposed using Koza's ADFs (Koza, 1994) to reuse useful subnets. This seems a very ingenious system, but its tree structure makes it difficult to see how to get it to cope with recurrent neural

[3] Connection weights are ± 1

networks, and it is unclear that a boolean network will suffice for this problem. This work is currently being extended by Rotaru-Varga (1999) to allow more modularity in the networks.

Nolfi and Parisi also produce feedforward networks with their encoding method, but it would be possible to make the system recurrent. It is based around neurons which are physically situated in two dimensions, and which then grow axons to connect to other neurons which branch and lengthen all according to the instructions on the chromosome (Nolfi and Parisi, 1992, 1993; Nolfi *et al.*, 1994a,b; Nolfi and Parisi, 1995). In (Nolfi *et al.*, 1994a) they evolve controllers using this for autonomous (wheeled) robots first in simulation and then on the actual robots with marked success. They have also added a developmental encoding to this as well (Cangelosi *et al.*, 1994), which allows the cells to divide and migrate about the space whilst altering their parameters (*e.g.* axon shape and weights). This system is probably complete, and biases towards local connections. In its original form it did not allow the possibility of reuse of subnets, because the position of every node would have to be altered; however, with the developmental encoding this is now possible as a split of the original node which produced the subnet will produce two identical ones physically removed from each other. Unfortunately it is an extremely complex scheme, but it could potentially be altered to generate desirable network structures.

Very recently work has been carried out by Kodjabachian and Meyer (1998) which extends Gruau's work to allow cells to grow in a space similar to Nolfi and Parisi's. This is a promising approach, and has been used to control statically stable walking in two dimensions as well as higher level control including gradient following and obstacle avoidance. However, it would be very interesting to see how this approach could deal with the much more difficult problem of dynamically stable walking.

Finally Karl Sims had a more complicated development process which he used to evolve various robots and their nervous systems (Sims, 1994a,b) to do jobs like walking, swimming and jumping. This process evolved the morphology of the robot as well as the controller, and produced very impressive simulation of robots crawling, swimming, and walking, with the bodies being more snake-like for swimming, and legged for walking, *etc.*. The nervous system was not in fact a straightforward neural network, but rather contained a variety of different nodes including thresholds, integrators, and oscillators amongst others. Each body part has a piece of the nervous system associated with it so that changes in the morphology of the robot change the controller as well. This technique worked well, and some very entertaining virtual robots were evolved: although none of them were very conventional walkers (they tended to tumble and roll, since this is easier to evolve), this provided a very impressive example of what should be possible.

This kind of coevolutionary approach, whilst interesting, is not directly applicable to the problems being solved here as we have fixed robots we wish to build controllers for. However the idea of associating a part of the controller with each joint is one which accords well with our thinking on CPGs, and could be incorporated into the system for automatically decomposing the problem which we have discussed before. As we have said before this is the kind of approach used by Kodjabachian and Meyer (1998), though in that paper the adaptation to the specific robot was done manually.

3.3 Summary

Because of constraints on the amount we know about the details of walking, the most appropriate method for training controllers for legged robots is a reward based approach, and we have chosen to use evolutionary algorithms to evolve neural networks. There are many different encodings, but the most appropriate to test seem to be:

- A simple direct encoding of weights, both fully connected and with the network architecture determined by an automatic analysis of the structure of the robot (which is described in Section 5.2).
- A direct encoding of the weights and architecture similar to above, but allowing the connectivity to vary, perhaps favouring denser local connections and sparser distal connections.
- An encoding similar to Angeline *et al.* (1993), adapted to allow reuse of subnetworks as above. This will be useful as it is a complete encoding which will allow us to see what kind of structures are useful so that we can consider designing a

more indirect encoding to produce these structures automatically. Its weakness of not allowing reuse of subnetworks is overcome by the automatic symmetry which will be built into the system by the analysis of the robot.

We will also investigate a variety of different neural models to see which seem most appropriate for this task.

Chapter 4

Architecture

In order to evolve the networks described in the last chapter it was necessary to measure how well they were capable of controlling legged robots. There are two possible approaches to this — either run the controllers on a real robot and evaluate their effectiveness *in situ* (as was done by Lewis *et al.*, 1992), or build a simulator which will mimic the behaviour of real or putative robots, and evaluate the performance of the simulants in the expectation that this will approximate to that of the real robots (as was done by Nolfi *et al.*, 1994a). There are drawbacks to both positions — for instance:

- Simulated robots can never be the exactly the same as real robots, so you never get a truly accurate picture of how your controllers would behave in the real world.

But:

- Real robots are expensive and fragile, needing constant maintenance and supervision, whereas simulations can run indefinitely with neither.
- There is an upper limit (in terms of both time and cost) on the number of different real robots it is possible to experiment with, but there are less problems with computers to run simulations on.

This last point was crucial — since the aim is to build a system capable of generating controllers for an arbitrary walking robot, it would be impossible to test this using actual physical robots, as only a very small number could be examined, whereas as many computer models as necessary can be devised. It is nonetheless a serious issue that simulated robots cannot guarantee to behave in the same fashion as a real robot should. However work by Nolfi *et al.* (1994a) and more recently by Perkins (1999) amongst others shows that controllers evolved in simulation can run effectively on real robots: Nolfi compares this transition to nothing more than a change in environment for the robot.

As a result it was determined that a full three dimensional dynamic simulator should be built with the ability to model as wide a range of legged robots as possible. Together with this it was necessary to create a neural simulator capable of modelling the behaviour of recurrent dynamic neural networks, and an evolutionary engine to evolve the networks. These then formed the basis for all of the research in this thesis, and this chapter will describe the system which was built and such details of the implementation as are necessary.

4.1 Design Criteria

The most important part of the system from a computational viewpoint is the mechanical simulator; it was obvious from the very beginning that the vast majority of the processor time would be spent here, so optimising this was a first priority.

4.1.1 Mechanical simulator

Initially many different simulators were tested, from our own (Reeve, 1994; Reeve and Hallam, 1995), which was implemented for another project based on work done in Edinburgh by Featherstone (1984), to various other simulators written by other robotics researchers and available on the web. However none of them were sufficiently robust or efficient for our purposes. It soon became apparent that there were several key criteria would have to be satisfied. The dynamical simulator would have to:

- be a full three dimensional dynamic simulator. Otherwise real robots could not be simulated effectively in the computer.

- be capable of easy extension to incorporate any elements which might be present on a robot it was decided to model (*e.g.* a new type of joint motor).

- work as efficiently as possible — since this module would do the bulk of the processing in the final program, the faster it could run the more experiments it would be possible to carry out.

- allow easy designing of and switching between different robot bodies.

Eventually the framework for a simulator based on the PhD work of McMillan (1994) was found, called DynaMechs. This satisfied all of the criteria as it could model an arbitrary tree shaped robot (*i.e.* no closed loops, see Figure 4.1 for examples) much faster than any other system tested. It was also modular, easily extensible and free. Work was done to incorporate a new simple joint motor and sensor design into the framework, and it was packaged in a simple interface which would allow it to be replaced easily with a different simulator at a later date if this proved desirable (for instance for modelling humans[1]). The simple language which was used to describe robots was also expanded to allow further information to be entered about the robot (an example file is shown in Appendix D.1).

The new motor was direct drive, offering torque or force control depending on the joint. A parameter in the motor definition in the robot description file determined the maximum torque (or force) individual motors could generate, and the motor was controlled by a single input between +1 and -1, for maximum forward drive and maximum reverse respectively. This made control by the neural networks less complicated as they would not have to generate different ranges of values for different motors.

The new sensors were equally simple, producing outputs proportional to the joint angles of the robots, but scaled to between +1 and -1 again, which were the front and back joint limits respectively. This simplified the inputs to the neural networks, as the inputs would vary over a simple range similar to that of the neurons themselves, so weights could be uniform across all connections. A further set of sensor connections, provided for joints on the legs, gated the first sensor readings depending on whether

[1] See section 8.2 for details

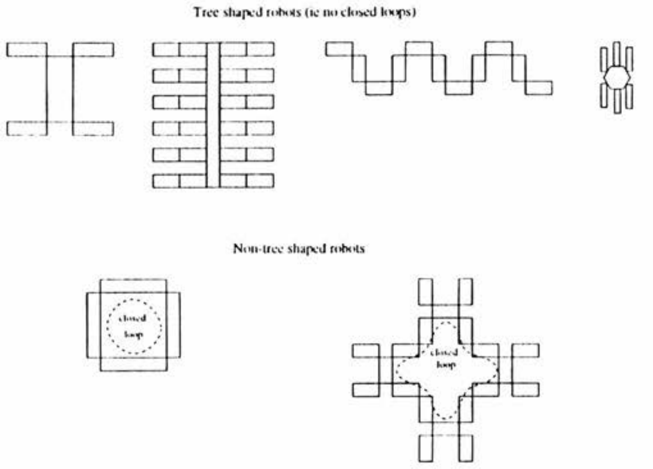

Figure 4.1: Tree shaped robots

the foot on that leg was touching the ground or not (producing a normal reading if they were, and 0 if not). This was provided to give some information about foot contact.

The interface, which was implemented as a C++ superclass of the mechanical and neural simulator classes, specified a set of methods which allowed the details of the specific simulator to be ignored. These are detailed in Appendix C.

4.1.2 Neural Simulator

The neural simulator was always an easier problem, but some factors were important — it should:

- be modular to allow experimentation with a variety of different neural models.
- permit easy replacement of one network configuration with another for repeated experiments.
- be as efficient as possible.

Examination of a few available neural network simulators showed that they tended to be too general purpose for my needs, so a simple one was built, and packaged in the same interface used for the mechanical simulator, allowing new neural types to be easily plugged in, but was otherwise as uncomplicated as possible.

4.1.3 Evolutionary Algorithm

As far as the Evolutionary Algorithm was concerned, the constraints were very straightforward. It should be as unrestrictive as possible, allowing any kind of population (*e.g.* pannictic, island, finegrained), selection criteria (*e.g.* tournament, proportional, *etc.*), and crossover and mutation operators that might be desirable.

This proved very easy to satisfy as the Edinburgh Parallel Computing Centre had just created exactly such a system, called RPL2. This had a built-in Basic-like language in which reproductive plans were written, and libraries could be easily added to the language to add new encodings or operators as desired.

This then completed the design of the system, which is detailed in figure 4.2.

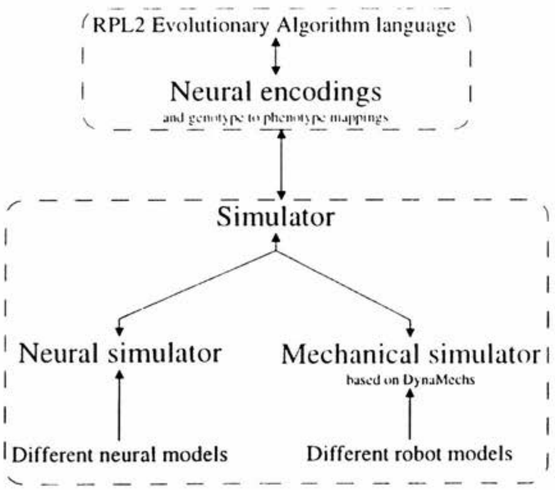

Figure 4.2: The basic program architecture

4.2 Verification

It was important to ensure that all of the components of the system were working as required This was done by testing each component separately and then in combination.

4.2.1 Mechanics

The mechanical simulator was difficult to validate as the calculations it was performing were extremely complex, so two approaches were taken:

1. Simple objects (*e.g.* cubes) were dropped and thrown in the simulated world and calculations were made to see what their trajectories should be. These were then compared.

2. More complex segmented objects (*e.g.* snakes and legged robots) were dropped and shaken, and visual observations were made to judge whether the simulated behaviour looked like the behaviour one would expect in the real world.

Although the latter seems a fairly unsatisfactory test, humans are in fact extremely good at distinguishing natural behaviour from artificial, and it was surprisingly easy to spot problems with the simulator when adding new components by this method.

4.2.2 Neurons

The neural simulator was relatively simple by comparison, and it was possible to construct simple neural networks and send them to the simulator and compare the neural activity to those generated by other methods. It is also the case that there was less need for accuracy in these tests as it was sufficient that the networks produced similar types of activations and that these were consistent between runs, as the layout of the neurons should be altered by the genetic algorithm to whatever was specifically required so long as they were capable of the right sorts of behaviour.

4.2.3 Evolution

Little testing was required here as RPL2 is now a commercial product, and validation was largely an excuse to familiarise ourselves with the software by running a variety of standard optimisation problems and checking that RPL2 was capable of evolving solutions to them.

4.2.4 In Concert

Connecting up the various parts and testing them together involved writing the first neural encodings for the GA. In fact, the most validation work (which carried on right through the experiments) went into checking the individual encodings and their respective mutation and crossover operators and genotype to phenotype mappings. This was done by examining individual chromosomes and checking that the decoding of them into networks for the neural simulator was correct, and that crossover and mutation were creating children that were derived correctly from their parents. An example of what information is passed back and forth in a typical experiment is shown in Appendix D.2.

Having done this, it was possible to carry out simple GA runs, first examining single chromosomes, checking that the GA had extracted the correct information about the robot to build appropriate controllers (*e.g.* number of actuated joints, sensors, *etc.*), checking that the neural simulator was building the correct networks, and then checking that the communication between the mechanical simulator and the neural simulator was working correctly, with the outputs from the neurons activating the motors and the feedback from the joints coming back into the network as input.

Finally, the first experiments were carried out to check that the fitness of the phenotypes was being measured correctly and the GA was carrying out the selection and breeding correctly to produce offspring from the fitter adults in the population. Regrettably these were not spectacularly successful at evolving walking controllers.

4.3 Summary

The simulation environment was constructed and validated to satisfy the following constraints:

- The mechanical simulator should be capable of accurately modelling the three dimensional dynamically stable movement of arbitrary legged robot (this is provided by the DynaMechs simulation code)
- It should be easy to transfer new robots to the simulation environment (a simple modelling language is used to describe robot in terms of its components, which is designed to be fairly intuitive).
- It should be possible to expand the simulation environment to allow for new motors, sensors, and types of neurons, *etc.* if a new robot requires it (this is built into DynaMechs and the neural simulator).
- Having put a new robot into the simulator, it should be trivial to evolve controllers for it (this is ensured by the GA and the neural simulator querying the mechanical simulator directly for information about the robot instead of asking the user).
- It should be possible to replace any component of the environment fairly painlessly if this is necessary, for instance to simulate a different type of environment (all components are connected by a simple interface which should be easy to implement on any replacement component).

Chapter 5

Symmetric controllers and neural models

In Chapter 3 we discussed the necessity of having degenerate controllers in our robots, so now we will consider how this will be implemented, and check that it really is effective by comparing results with those of a simpler implementation which does not take advantage of symmetries in the robots.

We also discussed the variety of different neural models which it is possible to implement, and we will investigate these thoroughly to determine which will be the most appropriate for the rest of our experiments.

5.1 Experimental design and results

All of the experiments in this thesis are evolutionary runs and as such the results are stochastic in nature. Consequently many repetitions of each experiment have to be carried out to get an accurate estimate of the effectiveness of any particular experimental setup. One of the main factors limiting the number of repetitions of each experiment was the amount of time available. Because of the complexity of the algorithms involved, the dynamic and neural simulators could only run at 60-80% of real time (*i.e.* taking 1.2 to 1.7 seconds to generate 1 second of simulation). This was almost entirely due to the mechanical simulator, although for very large heavily interconnected networks, the neural simulator did begin to have a marginal effect. Experiments were set up with 50 individuals per generation and 100 generations per experiment, and it was determined by trial and error that about 5 seconds of simulated time was the minimum necessary to determine whether the simulant was performing satisfactorily — if less time was allowed it was difficult to distinguish robots which had just thrown themselves forward in a single movement from those which had actually developed a repeating pattern of some kind. Likewise 50 individuals and 100 generations were the minimum found necessary to generally evolve as good controller as possible from the population. This was determined by the fact that the fittest member of the population stopped improving significantly for a number of generations, and while this is not conclusive, it indicated at least that further improvements were likely to take prohibitively long. Even stopping at 100 generations meant that each trial took almost 10 simulated hours, or between 12 and 17 cpu hours. This severely limited the number of trials that were possible for each experiment.

It was calculated during the experiments that 50 repetitions of each were sufficient to give statistically significant results in comparisons between most experiments whilst not being prohibitively slow (taking 25 to 35 cpu days). However, from a practical point of view it is clear that this would be too long for an end user of the system to wait for results, so a comparison was made between expected results from only 4 and 9 repetitions, which could be calculated with some confidence based on our larger sample. These numbers of repetitions were chosen as the 80^{th} and 90^{th} percentiles respectively in a uniformly distributed set of 50 trials, but in the end as the distributions were highly non-uniform (being generally unimodal but with significant tails and occasionally skewed as well — see Figures 5.5 and 5.9 for example), it was necessary to estimate these values directly by repeated subsampling from our population of results. A more detailed analysis of the statistical techniques used is to be found in Appendix B.

In fact, this proved to be a very satisfactory way of comparing different experiments, as although it is possible to compare expected mean or median performance of the evolutionary algorithm on a single trial or the best found across all runs, neither are usually a useful measure due to the stochastic nature of evolutionary algorithms; the former because the variability of individual trials means that more than one repetition is always done, and the latter because comparing the best trials achieved over the whole 50 runs opens up the possibility of having purely by chance succeeded in finding an abnormally good result in one of the experiments. Also there is no measure for the standard error of the best result and so no way of expressing confidence in the results obtained, and the standard error for the mean or median is generally larger than for a best of four or best of nine sample, so results are less likely to be significant.

Consequently all experiments are compared by considering their estimated best performance over 4 and 9 trials, and using the standard errors of those figures to determine whether the results were significant. However, other measures will be mentioned when they are seen to be important.

5.2 Degenerate controllers

It seems intuitively obvious that controllers for symmetric legs should be the same, but it is less obvious what exactly we might wish to class as symmetric in this context. However, as we said in Section 4.3, it is important for it to be easy to implement new robot models (otherwise the system will not be used), so if the controller is to be broken down into symmetric parts, this should mostly be done automatically from an analysis of the robot, and not require expert intervention from the user.

It is theoretically possible to analyse the actual structure of the leg models and determine whether they are identical (or mirror images of each other), and so require identical controllers, but it is prohibitively difficult to implement this in practice, and also might not achieve the desired effect (*e.g.* arms and legs could be designed the same for simplicity, but would still need different controllers). A much simpler scheme was therefore implemented where each articulation from the main body is described as a leg or part of the body (arm, head, *etc.*), and then the side of the body it is on is described (Left, Right, Centre), and then whether it is the same as any other legs is indicated by grouping them by number. The four legs on a simple quadruped are described as follows for example:

```
Leg Left 1
Leg Right 1
Leg Left 2
Leg Right 2
```

Indicating that the first two (fore) limbs are the same, as are the last two (hind) limbs. It would have been equally possible to define all four legs as the same, but the fore and hind limbs on this robot were sufficiently different that this was not appropriate. This is all the information that is used by the evolutionary algorithm to generate the degenerate controllers: it just uses topological information about the number of articulations (or limbs) — 4 here — the number of unique articulations (2), and how the sensors and actuators are distributed in the articulations. No information about physical dimensions of the robot is used. This is discussed in more detail in Appendix D.2.

In a degenerate controller, the neurons and connections are defined for only one articulation of each group, and are then duplicated in the others. In Figure 5.1, we see Leg Left 1 has one neuron on its shoulder joint (n1), with two connections (c1, c2). Consequently, Leg Right 1 has an identical neuron (n1') on its shoulder with two matching connections (c1', c2'). Similarly Leg Left 2 has 2 neurons (n2, n3) with a connection between them (c3), so Leg Right 2 has a matching set (n2', n3', c3').

The initial experiments were carried out with a very simple encoding where every joint had the same number of neurons and every neuron was connected to every other neuron and every sensor and actuator (the network is *fully connected*). In this case all possible neurons and connections always exist, so only the weights on the connections are being replicated between symmetric neurons (so the weight on the c1 connection is the same as that on the c1' connection). See Figure 5.2 for a simple example of this connectivity (only one sensor per actuator is shown). In the encodings in Chapter 6 on the other hand, the existence of connections and even neurons will be duplicated in the same way.

The non-degenerate controller is also fully connected, but symmetries are disregarded and so separate weights are encoded for neurons in each of the robot's articulations.

Full details of the setup of the Genetic Algorithm which remains the same throughout this chapter are in Table 5.1, and the robot is shown in Figure 5.3. The fitness measure used throughout this chapter is the simplest imaginable — just the average speed of the

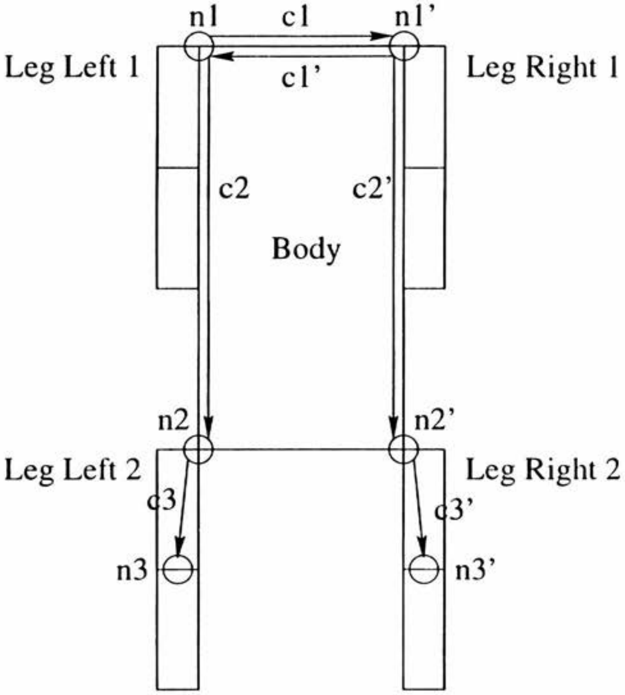

Figure 5.1: Replication of neurons and connections in a quadruped

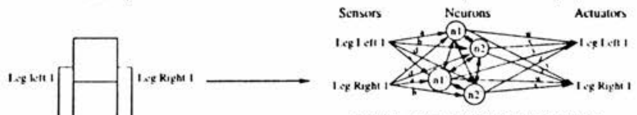
Robot with two joints
Leg left 1
Leg Right 1
Neural connectivity with 2 neurons per joint
Sensors
Neurons
Actuators
Leg Left 1
Leg Right 1
n1
n2
n1'
n2'
a
b
d
w
x
a - d and w - z are examples of the weights which are copied symmetrically from left to right and vice versa in the degenerate controller

Figure 5.2: Connectivity of neurons in a simple example

robot along its principal axis; we will investigate the usefulness of more sophisticated fitness measures in subsequent chapters.

Robot model:	quadruped with two hinge joints per leg with knees bending inwards
Fitness measure:	speed along principal axis (ms^{-1})
Population size:	50
Generations:	100
Selection type:	Tournament
Tournament size:	3
Prob. crossover:	0.8
Crossover type:	1 point
Mutation rate:	0.1

Table 5.1: Details of Genetic Algorithm parameters for Chapter 5

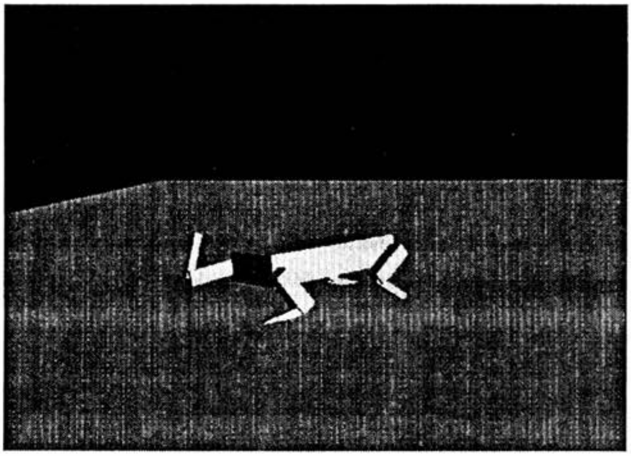

Figure 5.3: Simple quadruped with two hinge joints on each leg, knees bending inwards

Both experiments were repeated 50 times, both used the third order neural model described in more detail in Section 5.3.4 because this was a model which had not been used before for this class of problem (walking control), and it was our expectation that more sophisticated models might perform better; also every actuator was given 6 neurons; that is to say that there are 48 neurons in all, as there are 8 actuators. In this encoding there is no significance to the neurons being related to individual actuators, but in more sophisticated encodings in Chapter 6 the association will determine which connections can be made by the neuron. An example of this connectivity on a simpler robot was shown in Figure 5.2. Closer analysis later in the chapter will show these were reasonable parameters to get good results from the system. The results can be seen in Figures 5.4 and 5.5, and more discussion of the statistical techniques is found in Appendix B.

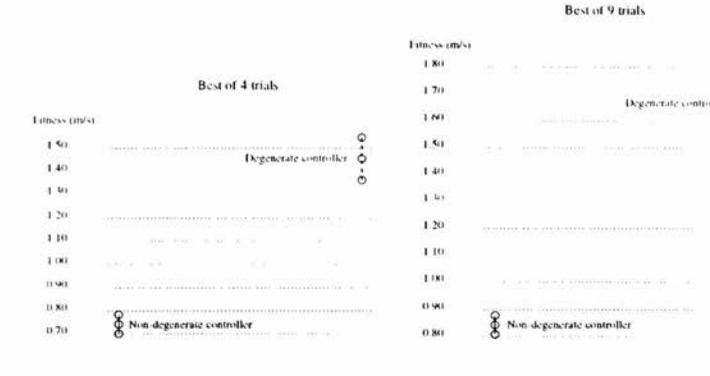

Figure 5.4: Expected values of best of n trials with 95% confidence intervals

There was a very large difference between the two experiments, with the degenerate controllers being significantly better even at the 0.1% level.

As we can see from Figure 5.5, the highest probability density for the non-degenerate controller is around a fitness of $0.5(ms^{-1})$ or roughly $3.5m$ travelled — this is generally a result of the robot learning to throw itself forward a couple of metres and then not moving again. Sometimes it did continue to move, but the legs tended to act independently and indeed no controller learnt to use all of its legs. The best of them is shown in Figure 5.6, the only one to keep a fairly stable rhythm going, and it still only uses 2 1/2 legs to do it.

The degenerate controllers, on the other hand, produced repeating patterns with all four of their legs over 90% of the time, with the mode around a fitness of $1.1(ms^{-1})$, or about 8 metres travelled in an average run. Many of these patterns were recognizable as stable mammalian gaits — Figure 5.7 for example shows a robot using an ambling gait, albeit mostly on its knees.

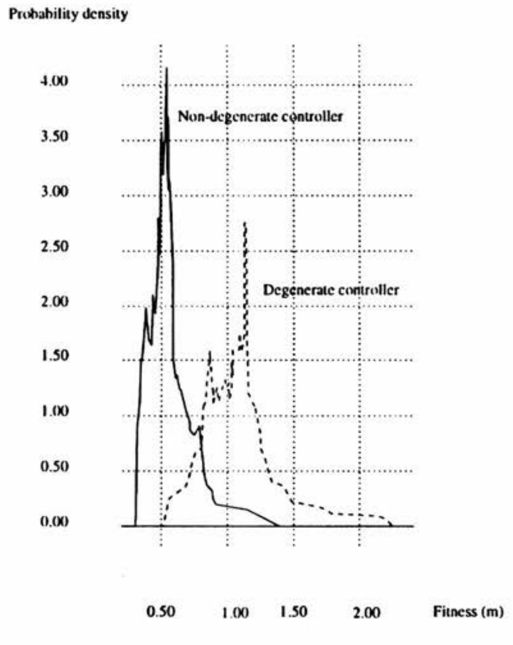

Figure 5.5: Bootstrapped probability density function of controller fitness estimated from results

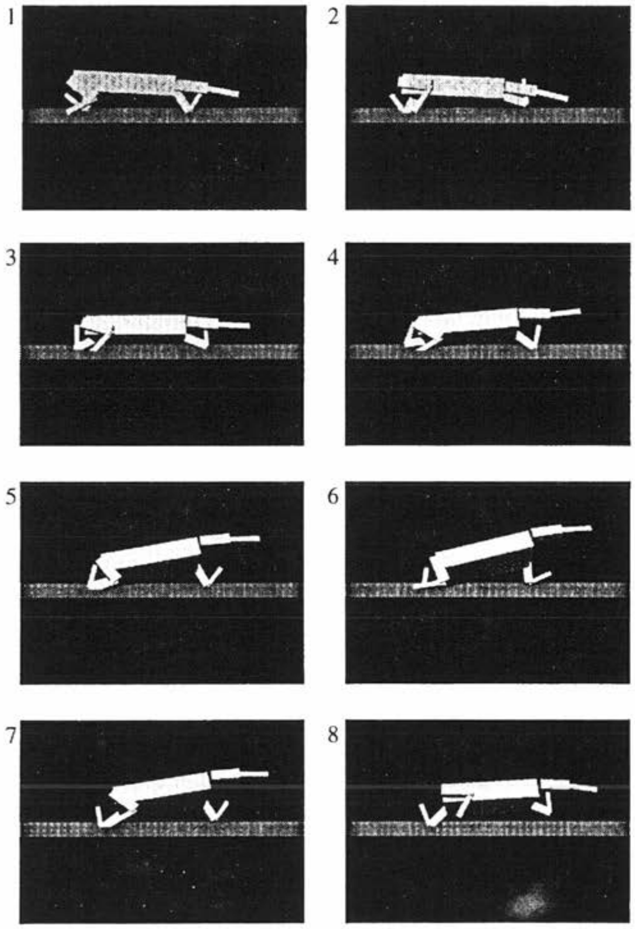

Figure 5.6: The best robot with a non-degenerate controller (viewed left to right, then top to bottom)

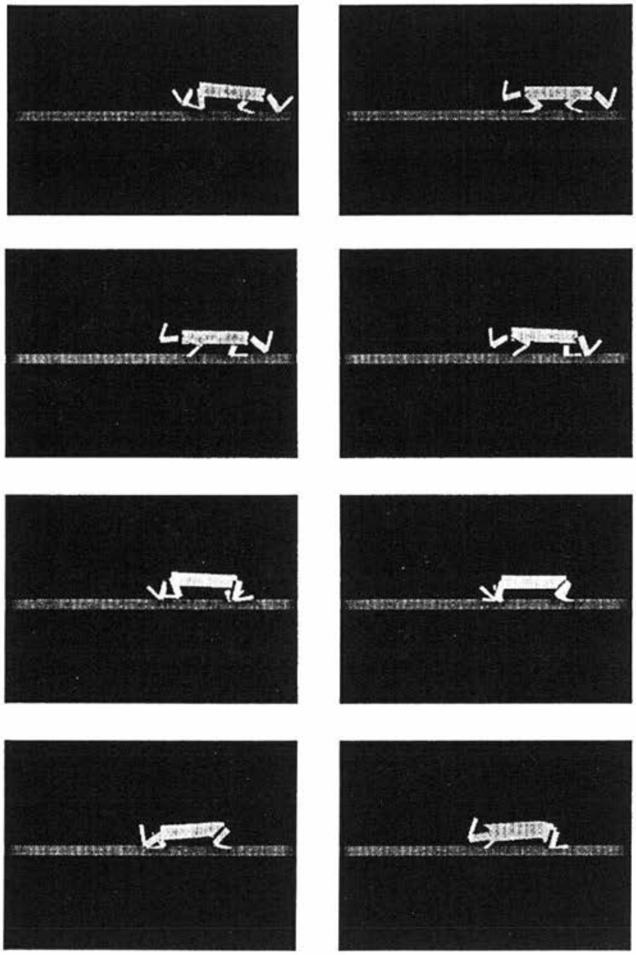

Figure 5.7: One of many stable robots with a degenerate controller (left to right, top to bottom)

5.2.1 Discussion

Building symmetry into the controllers had an enormous effect on the fitness of the controllers evolved by the system. It was to be expected that it would allow legs with the same controllers to perform similar tasks, thus helping to avoid the possibility of only three legs learning useful functions for instance, but what was not so obvious is that it also allowed the robots to coordinate much better between legs to develop stable gaits. This in itself is a significant result as we are not aware of any other evolutionary system evolving dynamically stable gaits in three dimensions; it seems to be a consequence of the symmetrical cross-connections between legs building up, so that a neuron activating the near forelimb of the robot when the near hindlimb is fully extended, will make its symmetrical partner do the same on the off side, for instance, and just a few of these types of connections will build up a simple rhythm between the legs. It was nonetheless surprising both how bad the asymmetric and how good the symmetric controllers were. It is possible that allowing the asymmetric controllers to evolve for longer would have produced better results as the genome had more degrees of freedom to explore, but it is not clear that this would have helped as the evolution had stopped at local minima which the system would then have to have broken out of, and in any event time constraints were imposed on the problem which necessitated stopping when it did.

The other significant feature of these runs was that the majority of the degenerate controllers moved at least partially on their knees and lower limbs. In retrospect this is not very surprising, as it is undoubtedly easier to balance when closer to the ground (children learn to crawl before they walk, for example), and using feet, which were otherwise absent from the model, also helps stability — since there was no penalty for doing this, it is in retrospect reasonable to find them using what were initially perceived as parts of the leg as modified feet.

It would be interesting to be able to examine how the neurons in these networks are controlling the walking behaviour of the robot, but unfortunately the large number of neurons (48), all of which are fully connected to all of the other neurons and all of the sensors and actuators, make it impossible to see any patterns in the neural oscillations. This is very frustrating, and affects the usefulness of the results, as it is impossible to show how reliable the network is. However, a robot model from a later chapter (Quad_same from Section 7.1.1) was sufficiently simple that it was possible to do a limited amount of analysis, and this is shown in Appendix A.

5.3 Neural models

Section 3.1.2 discussed a variety of different types of neurons which might be appropriate as a basis for a NN controller for legged robots. This section will now briefly recap each neural model, and then test its efficacy thoroughly in order to determine which model should be used for the rest of our experiments.

5.3.1 Sigmoidal

The sigmoidal neuron is by far the best known of all neural models used in Artificial Intelligence and many techniques exist for training it because of its mathematical tractability. There is however a strong drawback in using it here — it is not a continuous time model — and since this is a continuous time dynamic system we are trying to control, it seems likely that a network which can entrain to the frequency of the movements will be able to control it better than one which has its rhythm imposed from without.

However, it is the simplest neural model available, so we will investigate it first. The equations used are exactly as seen before, with y_i being the internal state of neuron i, and S_i being the output.

$$\begin{aligned} y_i &= \sum_{j=1}^{n} w_{ji} S_j \\ S_i &= \frac{1}{1+e^{-y_i}} \end{aligned} \tag{5.1}$$

I investigated a range of sizes of neural network, making sure to investigate enough to determine the optimal value assuming there was only a single peak in the values produced. The same technique was used throughout these experiments. In this case it was necessary to look at 6, 8, 10 and 12 neurons per actuator. The results are shown in Figures 5.8 and 5.9, and are compared with those from the 3rd order neuron examined in the previous section.

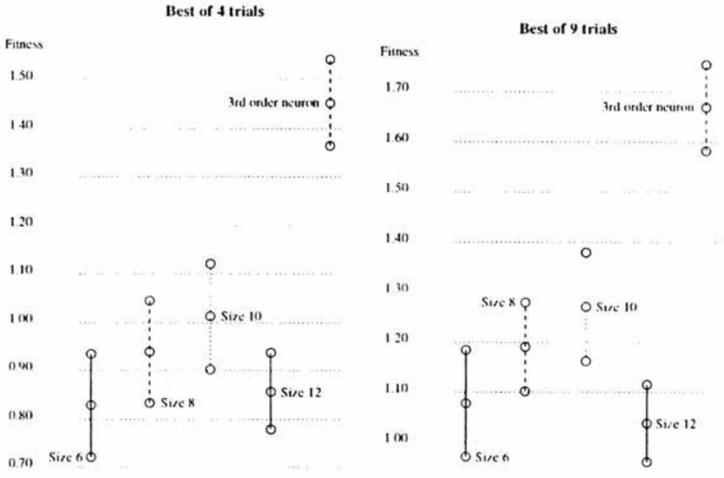

Figure 5.8: Results showing expected fitness with 95% confidence intervals for sigmoidal neurons

All of these results are significantly worse than the 3rd order neuron at the 0.1% level, and indeed the modal fitness is around 0.4 which is even below that of the non-degenerate controller examined before. However, the distribution is strongly skewed, with a few very fit specimens, and so the expected values of the best of 4 and best of 9 trials are better than might be anticipated. Though the difference is not significant, 10 neurons per actuator seems to be the best size of network for this type of neuron. Looking at individual controllers, many of them are learning repetitive movements, but the vast majority of these are not effective as gaits, consisting of dragging movements with one pair of legs, or some similar action. Only a very few in the upper tail of the distribution evolve any kind of recognizable gait, and even these do not seem to be very stable.

I will leave further discussion until we have examined all of the neural models.

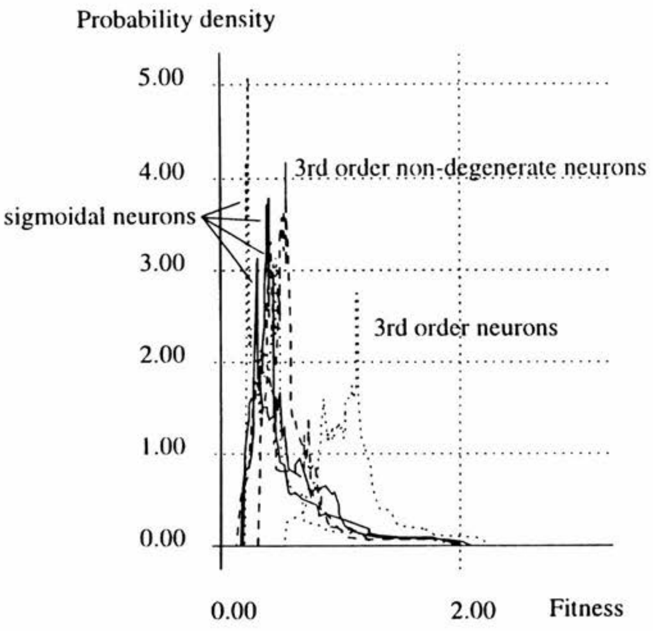

Figure 5.9: Bootstrapped probability density function of sigmoidal neuron fitness

5.3.2 First order

The next model is the CTRNN model of Beer (1995), which is a continuous time version of the sigmoidal neuron. Equation 5.2 describes the model, exactly as used by Beer.

$$\begin{aligned} \tau_i \frac{dy_i}{dt} &= -y_i + \sum_{j=1}^{n} w_{ji} S_j \\ S_i &= \frac{1}{1 + e^{(\theta_i - y_i)}} \end{aligned} \tag{5.2}$$

where: θ_i is a bias term,
τ_i is the adaptation rate of the neuron, and
y_i is the internal state of the neuron.

This time there are parameters θ_i and τ_i for each neuron, for which Beer suggests values based on his studies of the dynamic behaviour of the neurons (*ibid.*): $\theta_i = -2$ and $\tau_i = 1$. First we will compare the behaviour of the neurons with fixed θ_i and τ_i with the behaviour when co-evolving them with the weights, and then we will look at the number of neurons which produce the best controllers in this model.

The co-evolved parameters in Figure 5.10 prove to be significantly better than the fixed parameters at the 5% level for both 6 and 8 neurons per actuator, so we will now examine co-evolving the parameters more fully in Figures 5.11 and 5.12.

Surprisingly the 1^{st} order neural models do significantly worse than even the worst sigmoidal neuron. The best size for the networks remains 10 neurons per actuator, but the results are significantly worse even at the 1% level. However, as before the distribution of results shows that even the best sigmoidal neuron leaves something to be desired. The modal fitness for all of the 1^{st} order neurons is at least as good as that for the best sigmoidal one, and, as can be seen from the cumulative probability distribution, the medians are all roughly the same; again it is the skewed distribution of the sigmoidal neurons which makes it better in practice. Examining the first order controllers in detail, it is clear that very few of them even learn a repetitive movement, and those that succeed do not necessarily learn inherently stable ones — like that

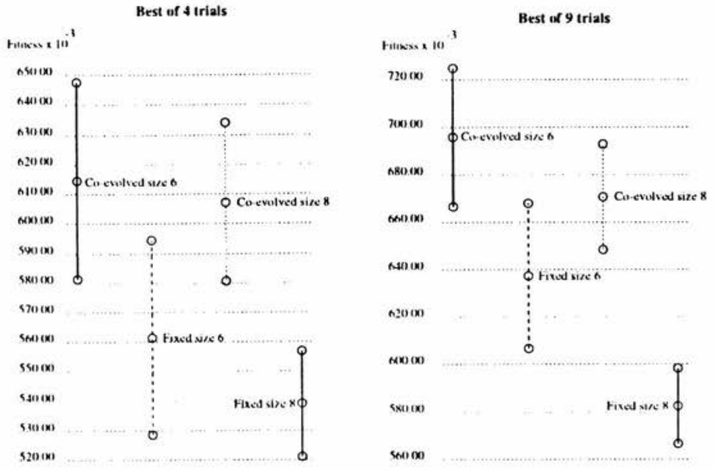

Figure 5.10: Testing co-evolved against fixed parameters for first order neurons

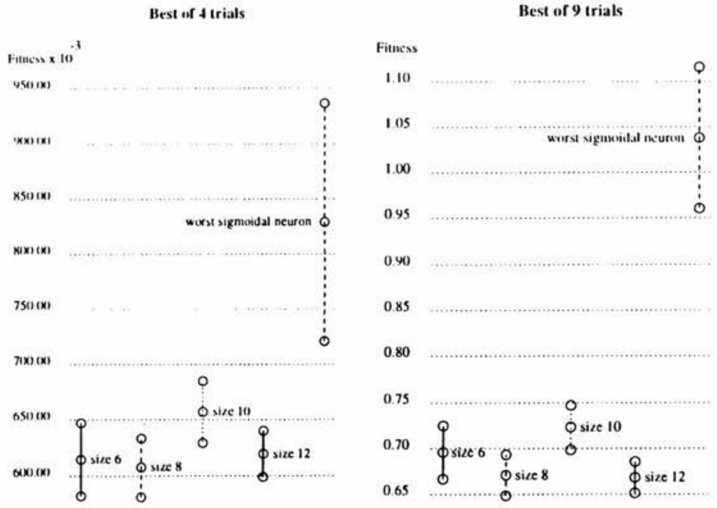

Figure 5.11: Comparison of different network sizes for first order neurons

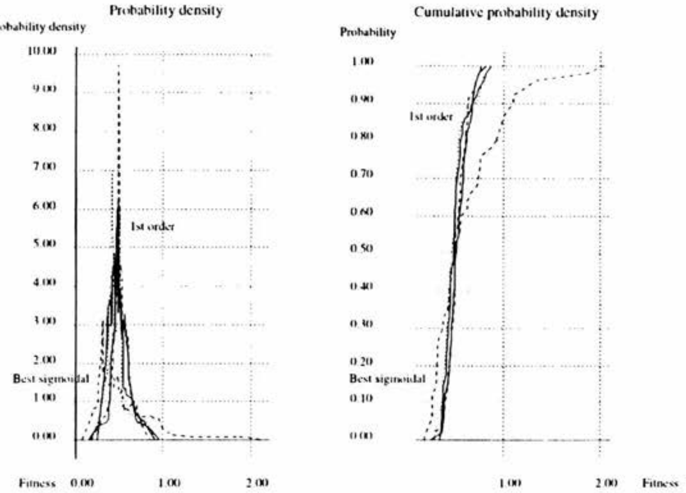

Figure 5.12: Bootstrapped probability density and cumulative probability density of first order neuron fitness

in Figure 5.13! It is entirely possible that a more sophisticated fitness function or encoding might have made a difference here, but this was not investigated.

5.3.3 Second order

The second order neural model is that of Taga (1995). It is governed by two coupled first order differential equations (5.3 and 5.4), which have been modified very slightly to simplify their use in this system.

$$\tau_i \frac{dy_i}{dt} = -y_i - \beta_i \max(0, y_i') + \sum_{j=1}^{n} w_{ji} S_j + k_i \qquad (5.3)$$

$$\acute{\tau}_i \frac{dy_i'}{dt} = -y_i' + S_i \qquad (5.4)$$

$$S_i = \min(\max(0, y_i), 1)$$

where: k_i and β_i are constants

We have allowed self connections, set a maximum output of 1, and Taga's global

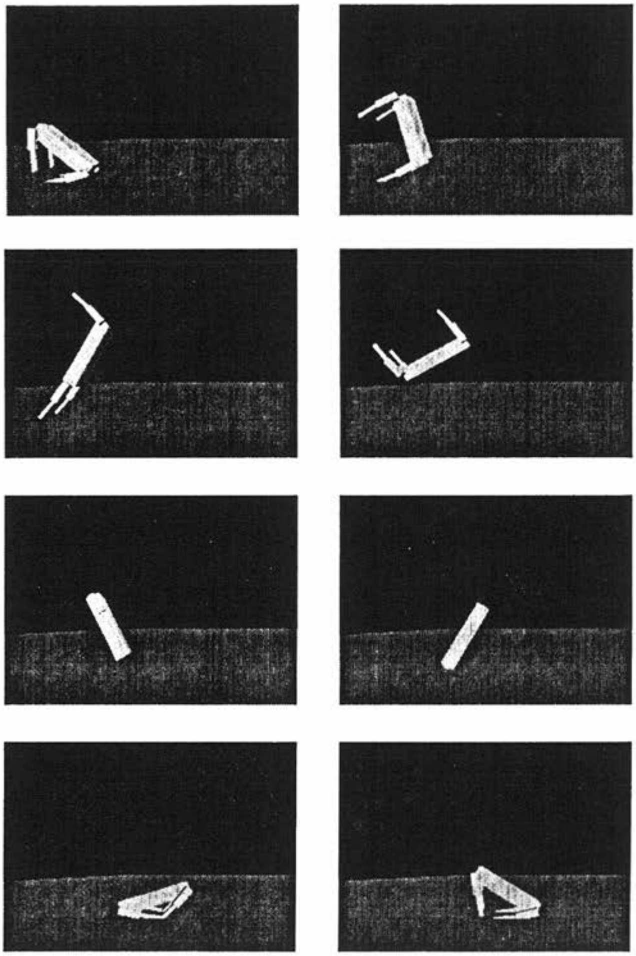

Figure 5.13: One of the more enterprising first order controllers!

constant u_0 is now a per-neuron constant k_i. Again Taga provides a set of values for τ_i, $\acute{\tau}_i$, β_i and k_i (1, 1, 2.5 and 1), so we will begin by testing these values against co-evolving the parameters.

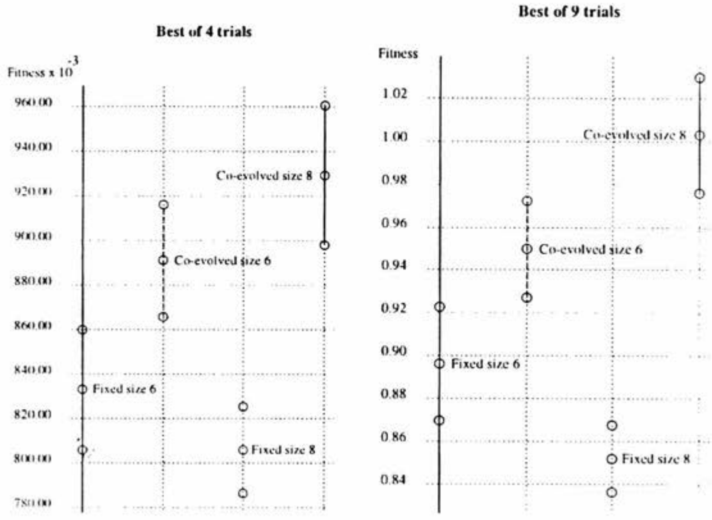

Figure 5.14: Testing co-evolved against fixed parameters for second order neurons

The co-evolved parameters in Figure 5.14 prove to be significantly better than the fixed parameters even at the 1% level for both 6 and 8 neurons per actuator, so we will now investigate co-evolving the parameters more fully.

The results of using second order neurons in Figures 5.15 and 5.16 are more equivocal. The best number of neurons to use seems to be 8, but the result is not significant. Looking at the best of four trials, the second order neurons seem to lie in the middle of the sigmoidal results. The best sigmoidal result is slightly better and the worst is slightly worse than all of the second order results, but again the result is not significant. However, looking at the best of nine trials, all of the sigmoidal results are better than the second order ones, although this is only significant for the best of them. Overall, the second order neurons are significantly better than the first order neurons, but are probably slightly worse than the sigmoidal neurons for practical purposes (looking at

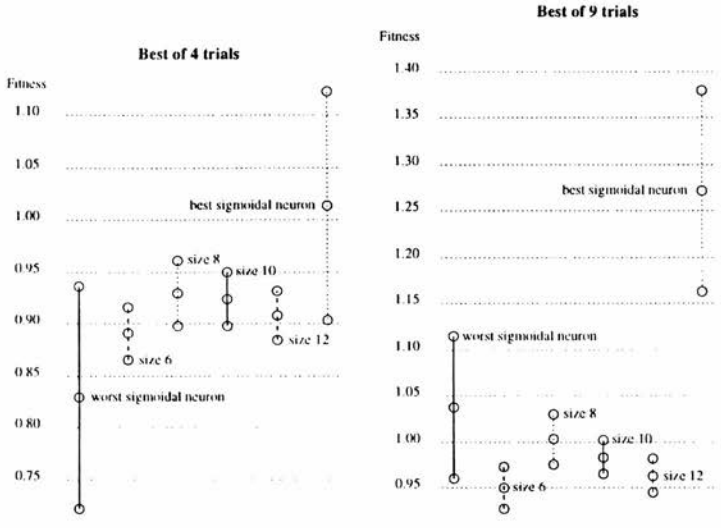

Figure 5.15: Comparison of different network sizes for second order neurons

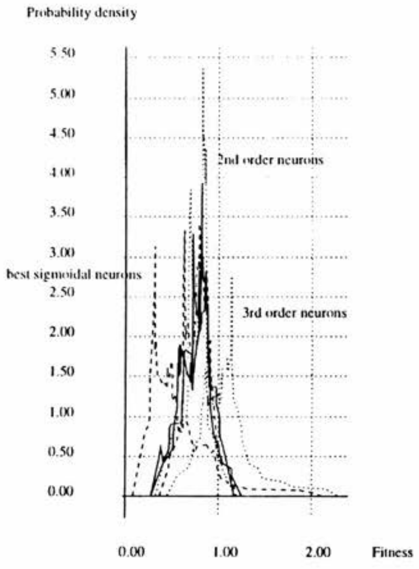

Figure 5.16: Bootstrapped probability density of second order neuron fitness

the distribution in Figure 5.16 it should be no surprise that the mean, median and modal values of all of the second order neurons are significantly better than the best sigmoidal one, but this is not really relevant). Examining the controllers individually, it is immediately obvious that even the worst controller has learnt some kind of repeating movement, albeit a dragging one, and indeed every controller seems to have evolved some kind of dragging or tumbling motion (like that in 5.13), although some of these are too unstable to continue indefinitely. However, none have learnt any recognizable gait.

5.3.4 Third order

We have already seen in Section 5.2 that these neurons can produce controllers significantly better than anything else that we have seen so far, but we will now look at them in more detail. They are based on work done in modelling neurones in a lamprey spinal cord, and are described in (Wallén *et al.*, 1992). Again there are some very small modifications to their equations which are shown in Equations 5.5 to 5.7.

$$\begin{aligned}
\tau_i^D \frac{dy_i^+}{dt} &= -y_i^+ + \sum_{j \in \Psi_+} w_{ji} S_j && (5.5)\\
\tau_i^D \frac{dy_i^-}{dt} &= -y_i^- + \sum_{j \in \Psi_-} w_{ji} S_j && (5.6)\\
\tau_i^A \frac{d\acute{y}_i}{dt} &= S_i - \acute{y}_i && (5.7)\\
S_i &= G_i^{\pm} \min(\max(0, 1 - e^{(\theta_i - y_i^+)\Gamma_i} - y_i^- - \mu_i \acute{y}_i), 1)
\end{aligned}$$

where: $G_i^{\pm}$ is ± 1 depending on whether the neuron is excitatory or inhibitory, and
$\Psi_{\pm}$ is the set of all excitatory (inhibitory) inputs, and
Γ_i and μ_i are bias terms.

The neurons now have a maximum output of 1, and whether they are excitatory or inhibitory is now explicitly stated in the equations instead of being defined indirectly through what weights the neurons are allowed to have. Wallén *et al.* also describe a set of parameters for four different types of neurons, which are listed in Table 5.2.

First we compare these parameters (choosing each neural type with a probability of 0.25) with co-evolved parameters.

θ	Γ	τ^D	μ	τ^A	$G^\pm$
-0.2	1.8	0.030	0.3	0.400	1
0.1	0.3	0.020	0.0	0.0	1
0.5	1.0	0.020	0.3	0.200	-1
8.0	0.5	0.050	0.0	0.0	-1

Table 5.2: Parameters for four different lamprey neurons

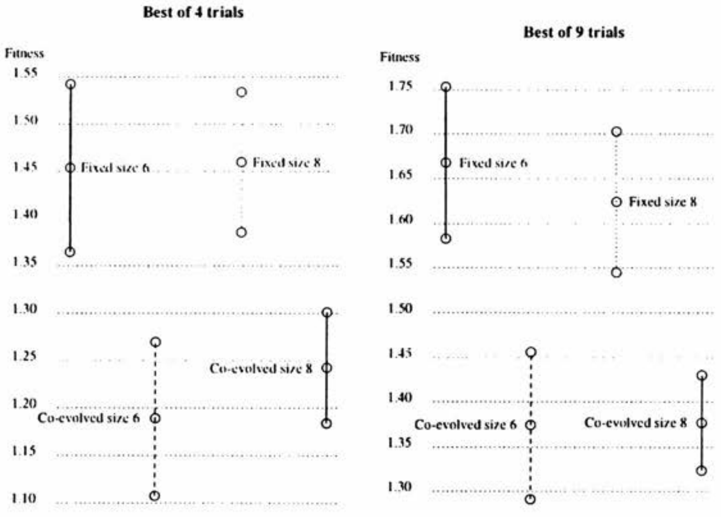

Figure 5.17: Testing co-evolved against fixed parameters for third order neurons

Surprisingly the co-evolved parameters in Figure 5.17 proved to be significantly worse than the fixed parameters even at the 1% level for both 6 and 8 neurons per actuator, so we will now investigate the third order neurons with the fixed parameters from Wallén *et al.* (1992) more fully in Figure 5.18.

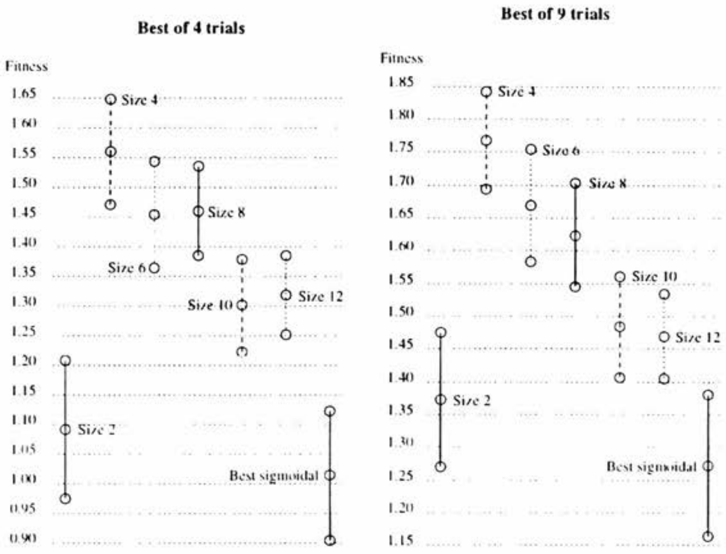

Figure 5.18: Comparison of different network sizes for third order neurons

All of these results are significantly better than any of the sigmoidal, first or second order results, and the best number of neurons per actuators has come down to 4, although this result is not quite significant at the 5% level. The controllers themselves all seem to induce some kind of oscillatory motion in the robots, though for the less fit individuals this tends to only be enough to catapult them onto their backs where they lie with their legs waving in the air. As the robots get fitter, gaits begin to appear, first dragging and tumbling moves we have seen before, and then with increasing fitness hops and more recognizable ambles and trots, and some even stranger (and faster) gaits like that in Figure 5.19 which moved the robot along at 2.5 ms^{-1}.

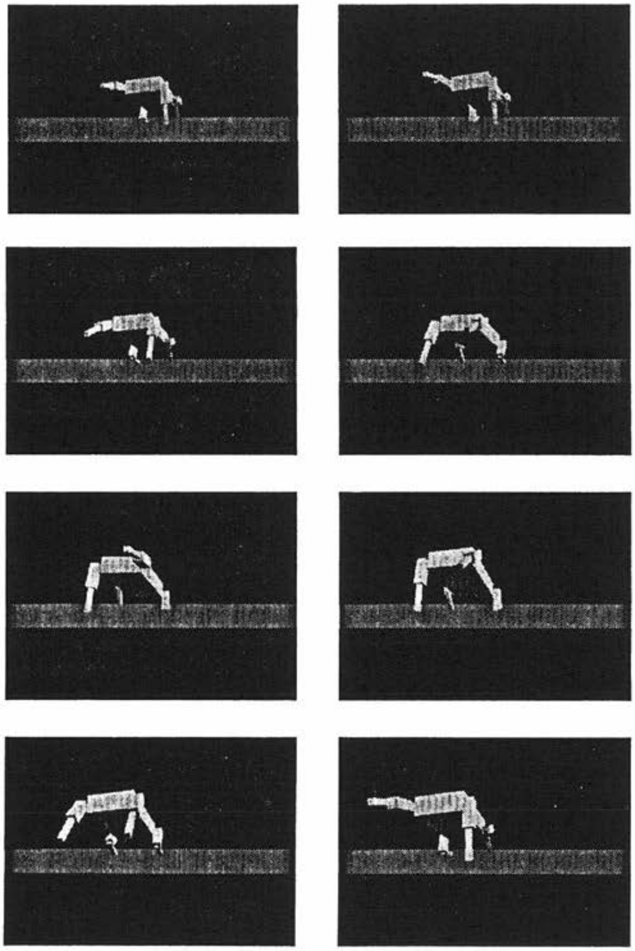

Figure 5.19: A typical third order controller

5.3.5 Discussion

Amongst the continuous neural models there was a clear progression from the simple first order models performing very badly to the complex third order models performing very well — one noticeable feature of the models is that with the first order model it is not at all easy to generate oscillations between less than three neurons, although it is theoretically possible with only two; with the second order model it is much easier, and with the third order it is hard to avoid, and indeed it is possible to set up an oscillation with only one neuron. This is reflected in the behaviour of the controllers, with very few repeated movements being seen in the first order robots, a lot coming in the second order though not much oscillatory behaviour, and nearly every single robot in the final set producing oscillatory movements; understandably this seems to help enormously in the generation of gaits, which are after all in their simplest form just a set of stable oscillations. It is difficult to say whether this is the main factor, as more complex neurons may be able to entrain better to the dynamics of the robot, but it is certainly true that the more complex neurons of Wallén *et al.* (1992) are the best for this task by far. Other experimenters have got good results with first order neurons (*e.g.* Kodjabachian and Meyer, 1998), so it is clearly possible. One possible explanation for this is that such simple fitness measures and network encodings were not good enough, and a more sophisticated approach might have done better, however it is noticeable that most of these experiments were done in two dimensions which is an easier problem which does not require dynamic stability. It is nonetheless significant that it was possible to get such good results with such a simple fitness function and network architecture with the higher order neural models.

Looking at the optimal number of neurons for different neural models, we see that it comes down with increasing complexity of neurons, from 10 for the sigmoidal and first order neurons to 8 for the second order, and only 4 for the third order. This approximately matches up with the complexity of the neurons themselves, suggesting that the increased complexity of the higher order neurons allows them to replace a few simpler neurons, and indeed do a better job at the same time. Note that we might expect more neurons to be optimal for the first order neural model, but in fact increasing the number of neurons being used seems to make it more difficult to evolve the best solution: the number of weights increases with the square of the number of neurons in a fully connected network and it seems likely that the evolutionary algorithm will begin to struggle to cope with the number of parameters it is evolving simultaneously, producing more and more suboptimal solutions even if they could be potentially better at that network size. Attempting to solve this by perhaps increasing the population size or the generations over which the GA is run would only result in even slower evolutionary runs which are already taking 17 hours at this size, and so would be impractical. However there are results which suggest that this might help if time were not an issue (*e.g.* Ackley and Littman, 1992).

It is also interesting to note that parameters suggested for the first and second order neurons, chosen after after careful mathematical examination of the dynamics of the neurons at least in the former case (Beer, 1995), proved to be less effective than just evolving the parameters along with the weights of the connections. It seems likely that the simple reason for this is that the criteria used to select these parameter values were inappropriate for this situation, which is unfortunate since both Beer and Taga in (Taga, 1995) were envisaging using their neurons for exactly this kind of work. Perhaps what was lacking was diversity in the neurons, needing a variety of different types for different purposes in the network. Either way quite the opposite was true for the third order neurons, and the set parameters for these were significantly better than evolved ones: it seems likely that it is significant that these parameter values were chosen because they matched real neurones controlling locomotion in a genuine vertebrate (albeit a fish), rather than satisfying perceived criteria seen from outside the problem. It is also possible that having a variety of different sets of values helped bring diversity to the neural network.

The other issue raised by these experiments is the results from the sigmoidal controllers. These were mostly worse than even the first order neurons but around 10% did much better, with a few even learning rudimentary gaits. This seems strange since the first order controllers showed no such tendency, but closer examination of the algorithms showed that the first order neurons had been evolving their adaptation rates at around the 1 second mark ($\tau_i = 1$ in Equation 5.2), which would allow them potentially to entrain to the dynamics of the mechanical system, whereas the sigmoidal neurons were updating every 0.001s (the stepsize of integration of the robot simulator), which was effectively their adaptation rate. It seems possible therefore that most were failing for understandable reasons but a few were actually learning to match specific input patterns and generate appropriate output patterns on the actuators as we ordinarily expect when training sigmoidal neurons. While it is interesting that they had some success in doing this, it seems unlikely to be a profitable approach to controlling walking in general because stability becomes so much more complicated when approached like this — the controller will have to learn every possible way of becoming unstable, and the apropriate outputs to rectify this, which is just too difficult in the general case. The approach also proved to be significantly worse than entraining the complex neurons to alter the dynamics of the system, and so it is not pursued any further.

5.4 Summary

The results in this chapter have been promising:

- Taking advantage of symmetries in the robot is essential in building a good controller — in the 50 experiments with an asymmetric controller only one succeeded in generating a repeating gait of any sort and that was extremely defective (using only 2 1/2 legs), whereas more than half of the degenerate controllers were acceptable and many produced recognizably stable gaits. Achieving a dynamically stable walking gait so quickly is very pleasing as it is a first using an evolutionary algorithm in three dimensions.

- Of the three continuous time neural models there was a clear progression with the more complex models being more suited to the control tasks given to them, both in producing more repeating gaits and in those gaits being more likely to be stable. This may be related to the increasing ease with which the more complex neurons produce oscillatory movements, and also the simplicity of evolving the fewer weights necessary in their increasing small networks.

- It seems significant that the actual parameter values taken from neurones in locomotory CPGs in a lamprey (Wallén *et al.*, 1992, from) were significantly better than evolved parameters for this task, unlike the simpler models, where the values chosen by researchers using perfectly plausible criteria proved to be less effective than allowing them to evolve with the connections.

- Little analysis is possible of the networks as they are large and fully connected to themselves and the 16 sensors and 8 actuators. This is frustrating as it hinders understanding of how the networks are operating, however in a subsequent chapter a controller is built for a robot which is sufficiently small that it is amenible to some analysis, and this is shown in Appendix A.

Now this system will be used as a basis for exploring further ways of improving the controllers generated.

Chapter 6

Encodings and fitness measures

The last chapter described experiments which allowed us to set up the basic platform for the rest of the thesis. Throughout all of the remaining experiments in this chapter and the next we will be using symmetric controllers and a third order neural model with the same robot as in the previous chapter. Now we will investigate the use of more sophisticated encodings than the current simplistic fully connected model to create the kind of connectivity which will make more useful controllers easier to evolve, and then we will look at the use of different and more complicated fitness functions and see how they can help evolve better walking behaviours.

These tasks are made difficult by the surprisingly high quality of the controllers generated in the last chapter by the third order models. Whilst the other models generated movements which at best merely satisfied the simple fitness requirement of moving forward, the third order models were extremely effective at generating genuine stable gaits, and it will be difficult to do better than this in these experiments.

6.1 Encodings

We discussed a variety of different possible encodings in Section 3.2, and it was decided that the encodings most likely to be useful for this task were:

- A simple direct encoding of weights, both fully connected and with the network architecture determined by an automatic analysis of the structure of the robot.

- A direct encoding of the weights and architecture similar to above, but allowing the connectivity to vary, perhaps favouring denser local connections and sparser distal connections.

- An encoding similar to Angeline *et al.* (1993), adapted to allow reuse of subnetworks.

The first (and simplest) encoding, which will now be referred to as the Full encoding, was implemented in the last chapter, and we will now look at how it compares with the other encodings.

6.1.1 The OneMotor encoding

From observations of connectivity in vertebrate CPGs used for control of locomotion (*e.g.* Wallén *et al.*, 1992), we know that neurones tend to be collected in small groups which are heavily interconnected, with fewer connections between the groups (these groups are the CPGs in fact); the CPGs are generally associated with a single muscle or group of muscles as they receive sensory signals from and output motor commands to only those muscles. It seems reasonable that this would be a good place to start in looking for ways to bias the networks being created by the genetic algorithm.

Consequently the first encoding examined here, called OneMotor, is a simple fully connected network, but one in which each neuron is associated with a specific joint, and can only send commands to the motor in that joint; it is also connected to the sensors associated with that joint, although there is a small possibility of connection to a distal sensor. This is one of the simplest possible encodings which has the correct kind of connectivity, and a typical example of connections generated by this encoding is shown in Figure 6.1. Details of what topological information is extracted from the robot model to allow the evolutionary algorithm to produce the symmetries in the encoding is found in Appendix D.2.

We used the same experimental setup as in the previous chapter (detailed in Table 5.1), except for using elitism[1] in creating new generations. This was seen to be useful

[1] putting the best of each generation directly into the next

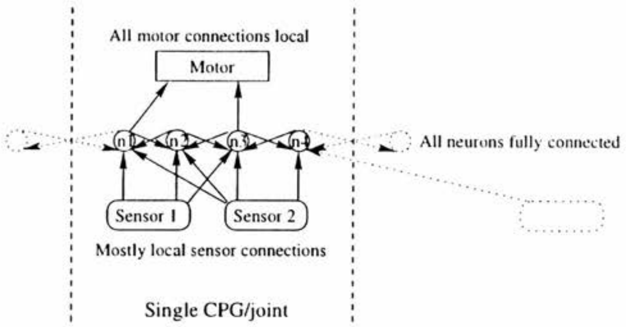
All motor connections local
Motor
n1
n2
n3
n4
All neurons fully connected
Sensor 1
Sensor 2
Mostly local sensor connections
Single CPG/joint

Figure 6.1: Typical connectivity of one CPG in OneMotor encoding with 4 neurons per actuator

as there was a tendency for good individuals to be lost during evolution, as can be seen from the dropping of fitness of the best individual in Figure 6.2 (the median run from best size — 4 neurons per actuator — of third order neural controller).

Running the experiments in Section 5.3.4 again with elitism produces significantly better results, and the median run from the best size network (now 6 or 8 neurons per actuator) can now be seen in Figure 6.3.

All the third order neuron experiments were rerun so that there could be a proper baseline against which to compare new results, and the expected results of that and the OneMotor experiments in the best of nine trials are shown in Figure 6.4. All experiments were run on the same robot with the same fitness function as in the previous chapter, as they are until we look at fitness functions in detail, and the same robot is used through the whole chapter.

Perhaps surprisingly the Full encoding (fully connected neurons) proves to be easily better than the OneMotor encoding, with the best OneMotor encoding being significantly worse than the worst Full encoding even at the 1% level. The same was also true when comparing expected results from the best of four trials. Although it is true that more neurons per effector could have been tested as no maximum had been found, it was felt that the results were so bad it wouldn't be effective use of the limited computing resources to study this simple encoding in any more detail. These results will

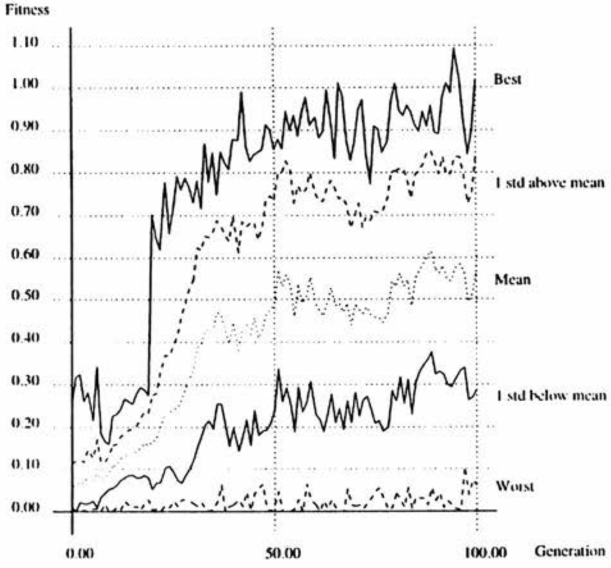

Figure 6.2: Typical evolutionary run for third order neurons without elitism

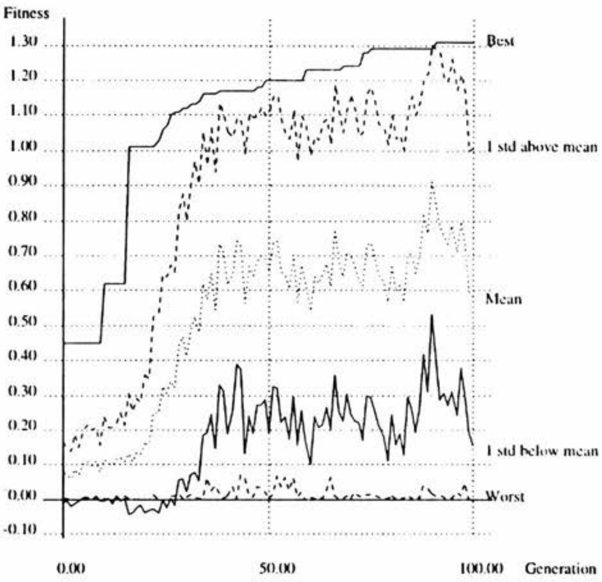

Figure 6.3: Typical evolutionary run for third order neurons with elitism

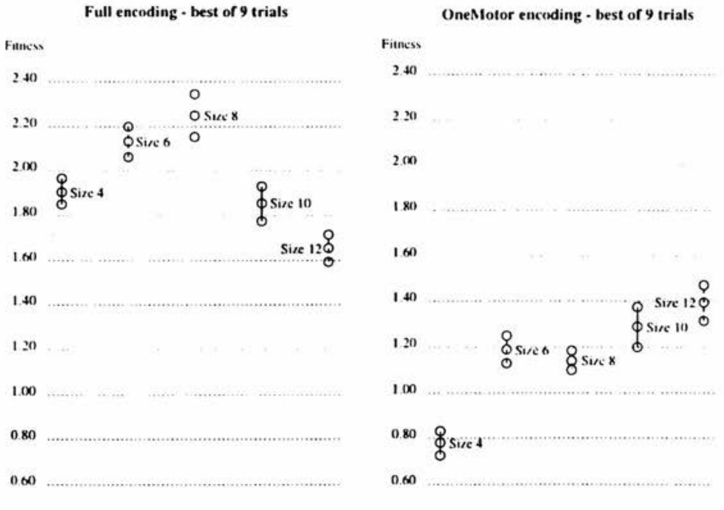

Figure 6.4: Comparison of Full and OneMotor encodings

be discussed further at the end of the section.

6.1.2 The LocalSparse encoding

Allowing the neurons to connect only to their local motors was not successful on its own, but in the LocalSparse encoding the connections between neurons is also changed so that distal neurons are less likely to be connected that nearby ones. This would make the connectivity more like that seen in real CPGs. Specifically, incomplete connectivity was allowed between the neurons, with local connections (to neurons in the same CPG, and the sensors and effectors associated with it) being made with a higher probability than distal ones (to other sensors and neurons). This encoding produces connectivity much like the OneMotor encoding but with much fewer interCPG and slightly fewer intraCPG connections (see Figure 6.5). The results can be seen in Figure 6.6.

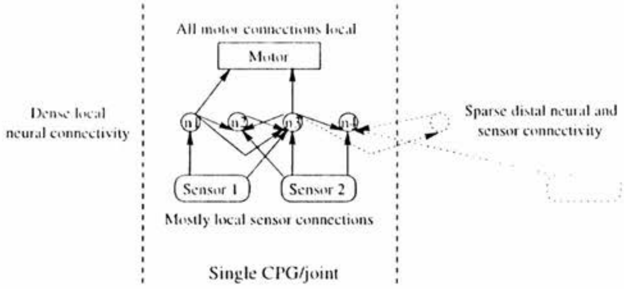
All motor connections local
Motor
Dense local neural connectivity
n1
n2
n3
n4
Sparse distal neural and sensor connectivity
Sensor 1
Sensor 2
Mostly local sensor connections
Single CPG/joint

Figure 6.5: Typical connectivity of one CPG in LocalSparse encoding with 4 neurons per actuator

These experiments show no significant improvement over the OneMotor encoding, even when looking at higher numbers of neurons per actuator; this was done here because it was felt that with the greatly reduced number of connections present in this encoding, it might be possible to evolve larger networks. However the results were still very significantly worse than the Full encoding.

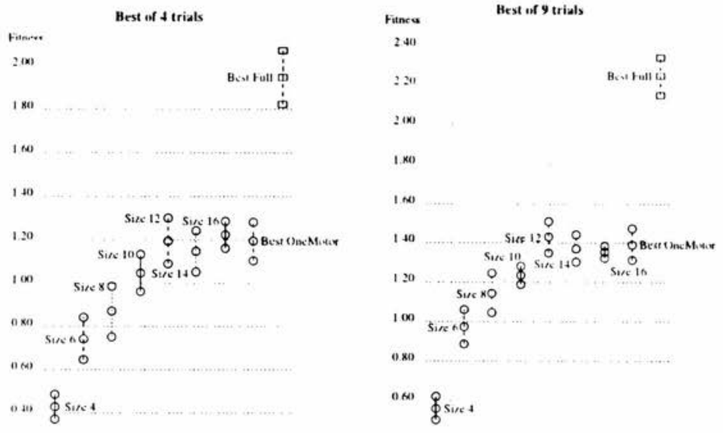

Figure 6.6: Comparison of results using LocalSparse encoding

6.1.3 The LocalGNARL encoding

The LocalSparse encoding has some similarities with the GNARL encoding of Angeline *et al.* (1993) described in Section 3.2.1, allowing mutations of connections and their weights, but has a fixed number of neurons instead of allowing mutation to change this as well, a bias towards local connections instead of a uniform probability of connectivity, and symmetry built into the system, which is absent from Angeline's work as it was unnecessary for the problems he was trying to solve. Also LocalSparse allows crossover unlike GNARL which had only mutation.

The LocalGNARL encoding is a compromise between the two, by allowing the number of neurons to vary. The crossover operator is also removed as it was not clear how to preserve connections during crossover in a network with varying numbers of neurons on each CPG, thus making the encoding even more like GNARL; this allowed addition and deletion of connections and changing their weights (which is already possible in the LocalSparse encoding), and also the addition and deletion of neurons. The number of neurons in each CPG is initially allowed to vary between $n-1$ and $n+1$ where n is the standard "number of neurons per effector" parameter used in other encodings, but after that is allowed to vary at will.

Two experiments were done here with the LocalGNARL encoding, the first as described above, and the second using a fitness proportionate mutation operator as described by Angeline (*ibid.*). This operator has a higher initial mutation rate, but this decreases for fitter individuals — fitter individuals take smaller mutation steps, and this makes them "slow" as they approach the summits of hills in fitness space. As usual an optimal network size was found for the initial population in each experiment, and Figure 6.7 shows how these optimal solutions compared with other encodings.

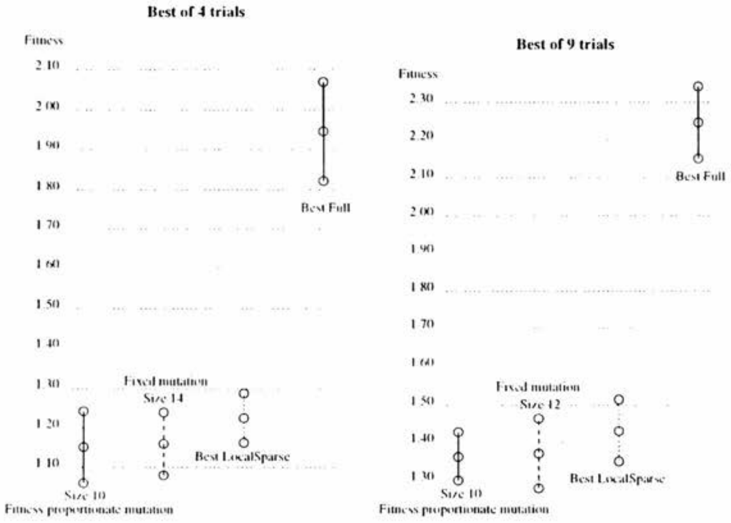

Figure 6.7: Comparison of results using LocalGNARL encoding

Fitness proportionate mutation seemed to make almost no difference with the results being nearly identical with or without it. Overall, however, the LocalGNARL encoding was slightly worse than the LocalSparse encoding, though the result was not significant.

6.1.4 The SymGNARL encoding

The local connections were stripped out of the LocalGNARL encoding to produce an encoding even more similar to the GNARL encoding of Angeline *et al.* (1993), allowing connections with equal probability to all sensors, actuators and neurons, but still using the symmetrical controllers of Section 5.2. This (SymGNARL) encoding was then compared with LocalGNARL and the original Full encoding, and the results are shown in Figure 6.8.

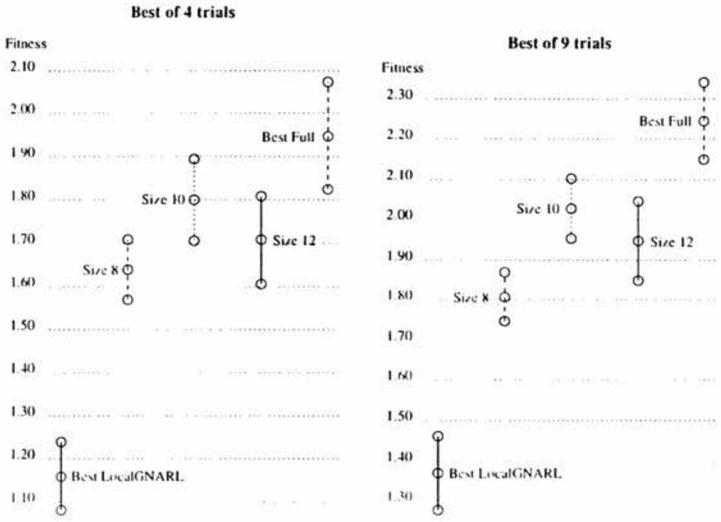

Figure 6.8: Comparison of results using SymGNARL encoding

Clearly these results are very significantly better than LocalGNARL, and indeed LocalSparse, even at the 1% level, but they are still slightly worse than the Full encoding, and this is significant at the 5% level for the best of 9 result.

6.1.5 The SymSparse encoding

The same was done to the LocalSparse encoding as to the LocalGNARL encoding, simplifying it so as to not differentiate between local and distal connections. The results are shown in Figure 6.9.

The results are almost identical to the original SymGNARL encoding.

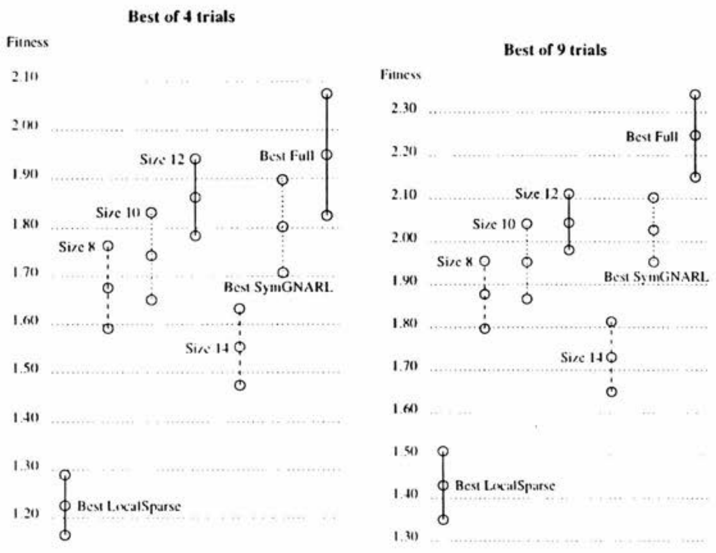

Figure 6.9: Comparison of results using SymSparse encoding

6.1.6 Discussion

It would have been nice to be able to discuss here what kind connectivity is best suited to this problem, and indeed it was originally intended to create a more sophisticated encoding based on information gleaned from these experiments, which would generate as close as possible to the ideal network configurations. However the results of this section have been very disappointing, showing, as they do, that it is very difficult to improve on the most basic encoding. Two possible reasons offer themselves:

1. The Full encoding is the best possible.

2. The wrong types of encodings were tried.

It is difficult to accept either of these, however, so we will look first at the individual results to see what the evidence shows.

Elitism

The results for the third order neurons were significantly better using elitism: this is usually a useful device for populations where most children are very unfit — even those of fit parents — as it allows the best member to survive until it can breed true. As we can see from Figures 6.2 and 6.3 the mean in the population is relatively low and stays there, and the worst usually has a fitness near 0, so it is unsurprising in retrospect that elitism is useful. It is also noticeable the best number of neurons per effector increases from 4 to 8 with elitism switched on, though in both cases 6 is not significantly worse; this is probably because size 4 networks previously bred truer and so benefited less from elitism (indeed there is very little improvement for this size of network), whereas larger networks benefited more.

Local connections

None of the three encodings with denser local connections were even close to being as good as the Full encoding, and the ability to vary the connectivity or even the number of neurons on each actuator made no significant difference. The LocalSparse encoding was slightly better, as might be expected as it allowed greater flexibility in the connections made, but LocalGNARL was worse, even though it allowed even more flexibility: it is likely this is because crossover was a slightly more effective operator than mutation alone, so its removal affected the GA more than the encoding did. It is also interesting that the fitness proportionate mutation had no effect on the results, which was also unexpected, but perhaps the fitness landscape is so epistatic[2] that it made very little difference: this seems plausible after our results with elitism, which also indicate that there is a rough fitness landscape which causes a lot of unfit individuals to be generated.

Since full connectivity of the neurons (in OneMotor) and sparse connectivity (in LocalSparse and LocalGNARL) were both tried with no significant difference in the results, and in the latter two it was possible to have many connections from other sensors, again without any improvement, it seems certain that the ability for individual neurons to

[2] small or few changes in parameters have large effects on fitness — causes a spiky fitness landscape

connect to more than one actuator is the significant factor here. It was assumed that the ability to connect freely to neurons in other CPGs which would then connect to the actuators could replace this, as happens in vertebrates for instance, but this is clearly not the case. The reasoning behind having more local connections was based on comparisons with vertebrate locomotor CPGs, but in retrospect it seems plausible that some other factor may drive this, such as physical separation of the muscles (and hence their associated CPGs) perhaps: indeed, physical separation of neurons is looked at in work by Kodjabachian and Meyer (1998), which shows that it can affect network development.

Whatever the reason, in this problem local connectivity is not useful for building walking controllers.

Sparse connections

In order to try to recover something from these poor results, the encodings were altered to remove their local connectivity, making them much more like standard encodings seen often before (although still with their symmetries). These were the SymSparse and SymGNARL encodings, and here the results were much better.

Neither encoding was quite as good as the best Full encoding, and this was significant for the best of nine result, but they were certainly of a comparable quality, with the best being better than many network sizes of the Full encoding. Their failure to improve on the simpler encoding may be because the optimal connection density was very high, at which point the encodings were less efficient than the Full encoding, as they had to encode both the connection's existence as well as its weight instead of just the latter. Consequently, the GA struggled to evolve appropriate networks (as it did with larger networks with Full encoding).

The SymSparse encoding was slightly better than the SymGNARL encoding, although the result was again not significant (as with the LocalSparse and LocalGNARL). This was slightly counterintuitive as the greater freedom allowed by the SymGNARL encoding seemed like it ought to help evolve better controllers. To test the hypothesis mentioned earlier that the GNARL encodings were worse simply because of the lack of crossover, the SymSparse encoding was rerun without the crossover operator, and SymGNARL now significantly outperformed SymSparse at the 5% level. However since no adequate crossover operator could be devised for SymGNARL this is of little comfort.

A variety of different initial connection densities was also tried for the SymSparse encoding to see if this could be optimised, but none of them outperformed the Full encoding, the one shown being a typical result (this had an initial connection density of 50%).

Conclusions

From the results in this section it seems that the best networks for controlling walking on this robot are densely connected with no real concept of neurons being local to any particular joint. This is unfortunate as it makes it very difficult to analyse the way the robot is being controlled, and thus generate confidence about the long term stability of the controller.

It would be possible to make an indirect encoding which could generate similarly complex networks, and indeed one was designed for this purpose, but it was never implemented as there was no evidence that biases inherent in this or any other indirect encoding would not act to make it worse than the simple Full encoding, just as the biases in all of the above encodings have already done. Although it might have been possible to get useful information for the indirect encoding by looking at the very densely connected SymSparse networks and seeing which connections were being removed, unfortunately the fitness of these was significantly lower the the Full encoding, so it was not clear that the correct connections were being removed!

In the end it is clear that the Full encoding with its simple built in symmetries is better at generating the kind of connectivity required for this task than any of the other encodings, and extensive research into a variety of possibilities failed to uncover any indication of what might be a better encoding.

6.2 Fitness Measures

Little attention has been paid so far to which fitness measure to use in evolving controllers. Certainly the extremely simple "average speed over 5 seconds" fitness function (Speed5) has been surprisingly effective, but we will now look at other more complicated measures which look at other concerns in controlling a physical robot and see whether they help. The simplest way to judge the controllers better is to see how they move over a much longer period of time, but because of simulation time limitations mentioned earlier it is not possible to examine the robots for longer in the standard fitness function.

Consequently we will examine different fitness functions to see how much they can help in evolving controllers which can walk for long periods of time, while still only testing them over short periods. We will do this by comparing the best evolved solutions directly over a much longer period of time (say 20 seconds), using the simple average speed fitness function.

First we will look at the original Speed5 fitness measure and see how controllers evolved for that do at the Speed20 fitness function. The results are shown in Figure 6.10.

The larger networks performed significantly worse over the longer timeframes, but the rest were roughly the same with a network size of 6 now slightly better than 8, although the result was not significant. This is roughly as we would expect since the 5 second evaluation time was chosen in Section 5.1 as just sufficient to judge whether the controller could keep a repeating pattern going. However, as we can see from the bootstrapped probability density functions in Figure 6.11, although the better controllers are still just as good, many of the worse ones clearly could not generate repeating gaits and their fitness consequently dropped a great deal.

What might help a robot keep moving over a long period of time then? Many possibilities exist, for instance:

- Keeping the centre of gravity high.
- Not allowing the body the touch the ground.

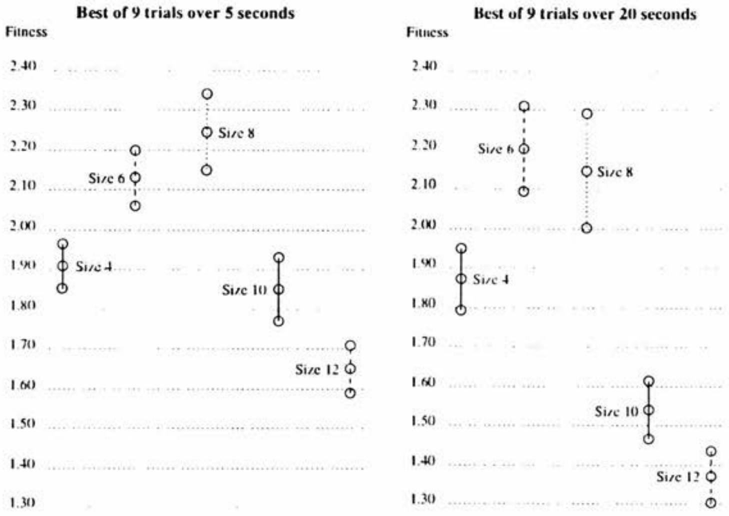

Figure 6.10: Comparison of fitness using evolved and extended fitness functions

- Making sure the legs oscillate.
- Making sure the neurons are active/oscillate.
- Minimising energy expenditure.
- Minimising ground forces.
- Putting more weight on the last couple of seconds than the first in the evaluation.

There are obviously many others. The first two are fairly straightforward, and are obviously related — if the robot's body is too low or touches the ground there is a significant danger of it tripping up. The second two are the kind of fitness functions often used in building legged robots in staged evolution experiments (*e.g.* Lewis *et al.*, 1992), and are felt to be useful as they are staging posts along the way to a good walking controller. The next two are fitness functions which appear to be driving forces in nature, the former to help endurance, and the latter perhaps to avoid damaging oneself. The last is a purely practical consideration — since the robot starts from

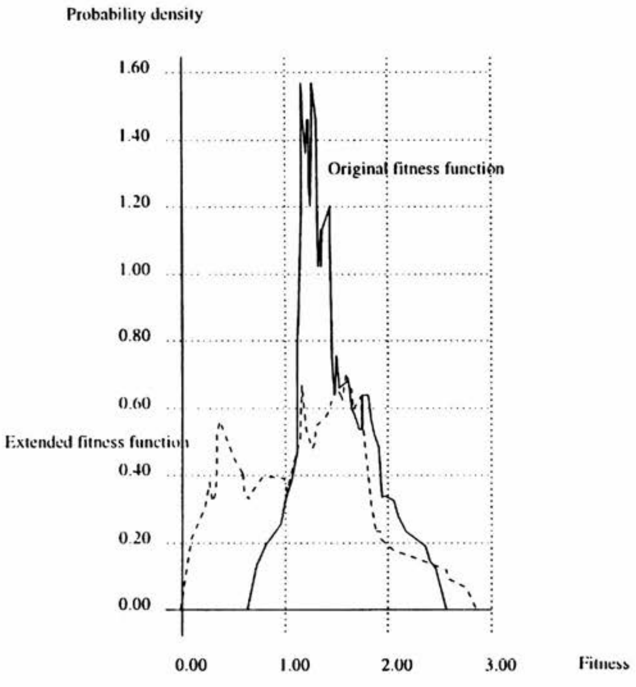

Figure 6.11: Prob. density of fitness of size 6 controllers using Speed5 and Speed20

standing still, it would be better to look at it after a couple of seconds when it had got into its rhythm, rather than straight away.

Measuring energy expenditure was eliminated as too computationally expensive (although keeping the body flat was tried unsuccessfully in an attempt to minimise unnecessary movement instead). Minimising ground forces was also eliminated as it was not clear how useful it would be, at least for the simulated robot we have at present. All of the other fitness measures were tried, individually and jointly. There were, however, far too many combinations to look at them all in depth, so I have selected a thread through them, adding one after another to find the most effective fitness measure. We will look at:

FND (forward not down) Average speed minus average distance of CoG below starting height.

DFND (decay FND) As above, but using an exponential decay of the fitness over the 5 seconds.

DFNDF (DFND or fall) As above, but with exceptional punishment if part of the body touches the ground.

DFNDFA (DFNDF active) As above, but making sure that the neurons and legs are active (*i.e.* the neurons are not full on or off and the legs are not locked against end stops.

DFNDFO (DFNDF oscillate) As DFNDF, but making sure that the legs and neurons oscillate.

Many other combinations were tried, but these were representative of them, and included all of the best results.

6.2.1 FND

This fitness measure was a simple extension of the Speed5 measure, punishing the robot for allowing its centre of gravity to drop too far as well. The results shown in

Figure 6.12 show how it compares with the Speed5 fitness measure when its controllers are re-evaluated using the standard Speed20 fitness measure.

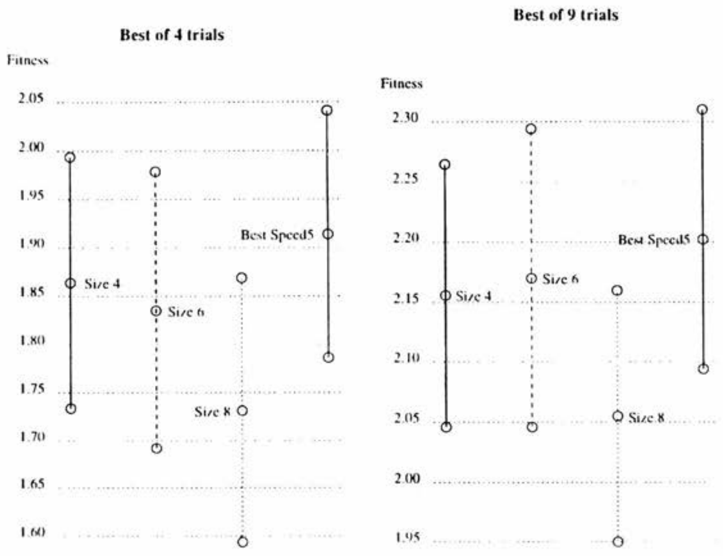

Figure 6.12: Comparison of results using the FND fitness measure

The FND fitness measure proves to be possibly slightly worse at generating good controllers, although the result is not at all significant. This was typical of many of the additional fitness functions on their own, including the "no falling" measure, but as we shall see later, they work well together. The size 4 and 6 networks are equally good here, with the size 8 slightly worse.

6.2.2 DFND

This next fitness function simply decays the previous one over time. The half-life of the fitness is about 2.4 seconds (the actual decay is 0.75^t), so the very first fitness measure has roughly 1/4 the weight of the last. The results of using this are shown in Figure 6.13.

This seems to be an improvement on the previous fitness measures, although the result

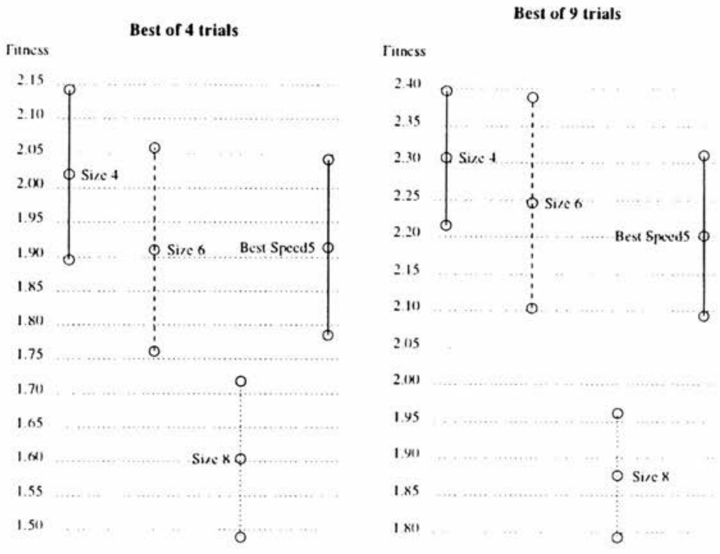

Figure 6.13: Comparison of results using the DFND fitness measure

is not quite significant at the 5% level. The size 4 network seems to be the best here, with the size 6 slightly worse, and the size 8 significantly behind.

6.2.3 DFNDF

This fitness measure is like the previous DFND, but with an additional penalty if part of the robot touches the ground, proportional to the length of time in contact (the parts of the robot defined as the body are detailed in the robot description). The results of this measure re-evaluated using the Speed20 fitness measure are shown in Figure 6.14.

This is a very significant improvement, both over the original Speed5 measure at the 1% level, and the DFND measure at the 5% level. Interestingly this fitness measure on its own and with the "not down" measure showed a similar performance to FND and Speed5, and it was only with the decaying fitness measure that this huge improvement was noticed. The best size of network for this measure seems to be 6 but 4 is not

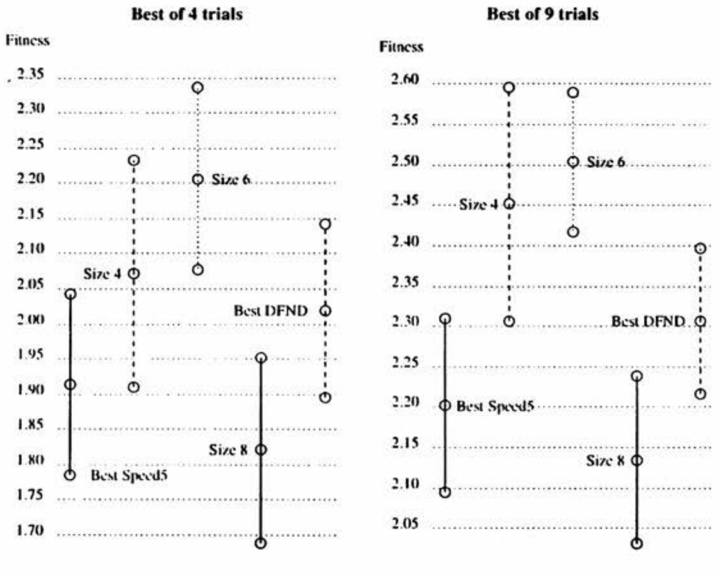

Figure 6.14: Comparison of results using the DFNDF fitness measure

significantly worse.

6.2.4 DFNDFA

This and the following fitness function are the only two that try to look at the detail of what is going on and specify directly what should be happening. In this case we take the best (DFNDF) fitness measure so far, and add to it a measure of how active the neurons and legs are. This was measured by penalising inactive (output 0) and totally overloaded (output ± 1) neurons and by penalising legs which are jammed up against their limits (sensors at ± 1), which we saw frequently in the last chapter.

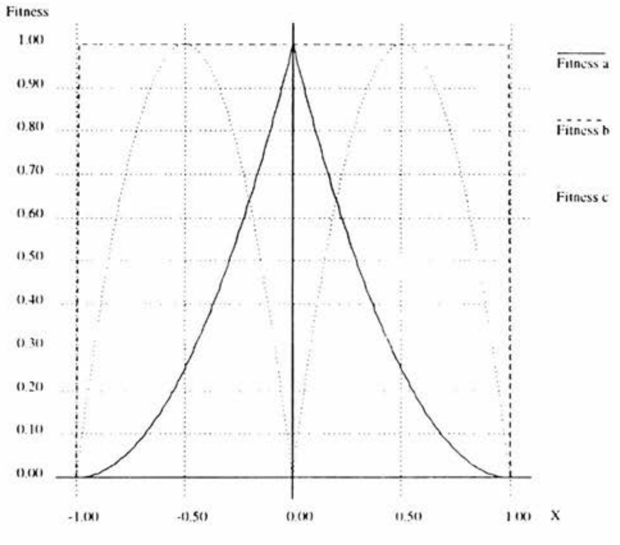

Figure 6.15: Three different functions used in DFNDFA

The fitness measures can be seen in Figure 6.15: the neural fitness measure is (c), and two different joint fitness were tried, (a) and (b), where (a) penalises being close to the joint limits, but (b) only penalises being pressed right up against them. They were both tried and are referred to as DFNDFAa and DFNDFAb respectively. The results are shown in Figure 6.16.

There is very little difference between these two fitness functions and the previous one

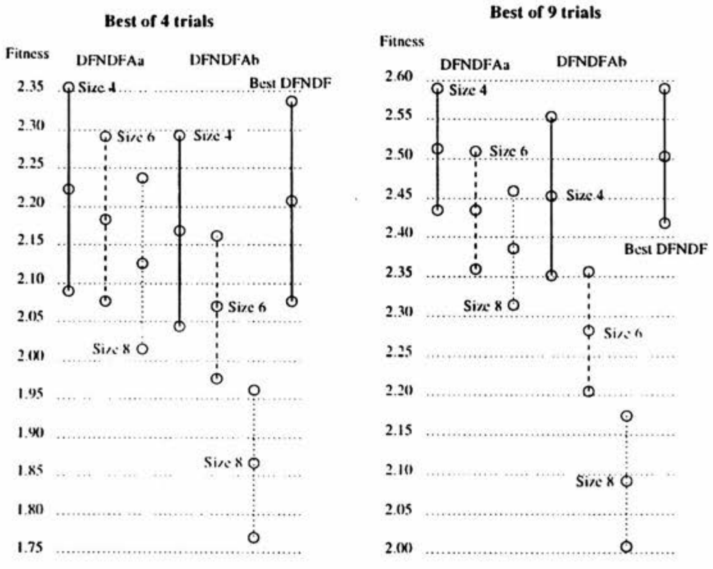

Figure 6.16: Comparison of results using the DFNDFAa and b fitness measures

which had no detailed neural and joint measure, one being an insignificant amount better, and the other worse. The optimal network size is 4 in both.

6.2.5 DFNDFO

Another feature often used in evolving controllers is that the neurons and joints should oscillate (*e.g.* Lewis *et al.*, 1992; Ijspeert *et al.*, 1997), so here we replace the previous measure of activity in the neurons and joints with a simple measure of variance. Again this is combined with the DFNDF fitness measure, and the results are shown in Figure 6.17.

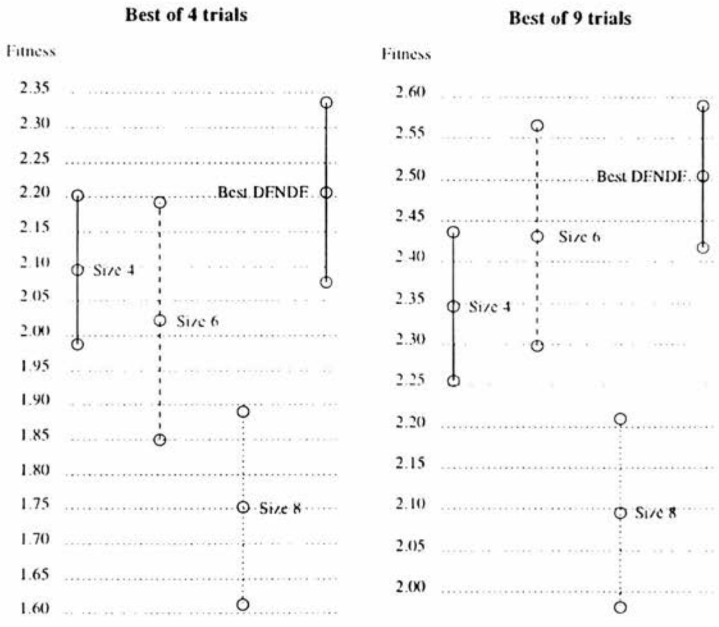

Figure 6.17: Comparison of results using the DFNDFO fitness measure

These results are worse than the DFNDF and DFNDFA results, although not significantly, with the best network size now being 4 or 6.

Many other combinations of fitness functions could have been shown here, but these results are typical of those found. Different fitness functions were not tried because other, simple functions like "not down" and "not fall" and "average speed" could not be found, the biologically plausible ones seemed either too complicated or not obviously useful, and the more detailed ones seemed not to help.

6.2.6 Discussion

Overall the simpler fitness measures (like "decay" and "not fall") worked well and the more complex (like "oscillate") which claimed to judge the internals of the robot and controller performed badly: however, the results were not quite that simple.

Neither "not down" nor "not fall" worked well on their own, but needed "decay" to produce good results. This seems reasonable, because many controllers early in evolutionary runs were unstable for the first second or two when they started moving, and so were penalised badly when no decaying of the fitness was included. That meant that it was difficult for evolution to start, so there was a tendency not to evolve as far or spend too much time optimising the first two seconds of motion and not enough time on the stable gait.

The more complicated fitness measures, "active" and "oscillate", used intuitive measures of what should be going on inside the robot and its controller to help guide the evolution. Unfortunately these did not turn out to be very useful — although DFNDFAa was very marginally better that DFNDF, the other two were marginally worse. Fundamentally these were based on the flawed premise that we should know what is the best behaviour inside the controllers should be, just as in the last chapter we found that choosing the parameters for the 1st and 2nd order neural models based on assumptions about the kind of behaviour neurons should exhibit was flawed.

In conclusion, when building fitness functions for this task at least, it is perfectly adequate to state the goals which we want to achieve and not try to prescribe the way to achieve them, either through staged evolution or more complex fitness functions as we tried here.

6.3 Conclusions

The results from this Chapter were not as clearcut as those from the last. Other encodings from the simple Full (symmetrical) encoding were tried, but with little success, but more complex fitness functions did improve the ability of the robots to move at speed. In conclusion:

- Sparse distal and dense local connectivity, although found universally in CPGs in vertebrates, was not useful in controlling these robots; indeed the ability for a single neuron to output to more than one actuator was crucial to building good neural controllers. Taking inspiration from nature is clearly risky when the "reasons" behind natural designs are not known. In this case, perhaps other factors drive the sparse distal connectivity such as actual physical separation of muscles, which causes separation of CPGs in the spine, and hence difficulties in actually making the distal connections as well as time delays in the signals being passed, neither of which are modelled here.

- Because the Full encoding proved to be better than all of the non-fully connected encodings which were tried, and the closer the encoding came to the Full encoding, the better it did, it was decided that there was no point in implementing the indirect encoding which had been designed, there being no reason to believe that the biases in it would be more desirable than those in a fully connected network.

- Simple additions to the fitness functions such as a decay term to eliminate problems at the beginning of the simulation, and penalties for the body of the robot hitting the ground, or the robot lowering its centre of gravity, worked well to increase its ability to move at speed over longer periods of time than the 5 seconds allowed for fitness evaluations.

- More complex fitness functions to determine whether the neurons or joints were doing precisely the "right" thing were not very successful, as anticipated in Section 3.2.1, when we decided against staged evolution because it was impossible to tell what was the right behaviour for an arbitrary neuron in an arbitrary robot was.

The common thread between the results in this chapter and the last is that we have discovered again that having a seemingly reasonable feeling (say that we want the legs to oscillate) does not necessarily mean that it will be useful in practice, and we should stick instead to the goals we wish to achieve — if we want a robot to move forwards without falling over, then it is perfectly possible to specify just that and the evolutionary algorithm will take care of the rest. It is reassuring and somewhat surprising however that this simple approach does work.

Chapter 7

Testing the system

In the last two chapters a system has been developed which can repeatably generate low level controllers for our target quadruped, allowing it to walk or run in a straight line and at a constant speed. In this chapter we will examine whether the system is as general purpose as is claimed, and whether it is fit for purpose as part of an active walking mechanism which can be controlled by a simple higher level controller.

7.1 A selection of robots

To test whether the system is sufficiently general purpose to use on arbitrary legged robots, a series of different designs were created, and the best evolutionary setup from the previous chapters was taken and used directly on the new design with no modifications. The details of this setup are shown in Table 7.1 (the GA parameters are the same as used in the previous Chapters).

Neuron type:	Third order
Neurons per effector:	6
Encoding:	Full
Fitness measure:	DFNDF

Table 7.1: Details of setup of experiments for Chapter 7

The robot designs broke down into three categories:

- Other quadrupedal robots

- The original quadruped in different environments (*e.g.* low friction, low gravity)
- Robots with different numbers of legs

It was one of the design criteria of the system that it should be possible to encode other robot designs, but it is an significant extra that the environment can also be modified like this as it is a feature which is not investigated in other similar research (*e.g.* Kodjabachian and Meyer, 1998). We will look at the variations in the robot designs in the order given above.

7.1.1 Quadrupeds

The first new robot was similar to the original (Quad_in), but with all of the knees bending in the same direction (see Figure 7.1), and was called Quad_same. This meant that all of the legs could be given the same controller, and I experimented with this as well as with separate front and back leg controllers. With different controllers for front and back legs, it was very similar to the Quad_in robot, producing some fast if slightly unconventional gaits: but with only one controller duplicated across all of them it easily outperformed them, producing some very good controllers using fast trotting gaits, one of which is shown in Figure 7.1.

All of the speeds for the quadrupeds (measured using the standard Speed20 fitness measure) are shown in Figure 7.2, though it should be noted that as different robots have different maximum speeds, no strong conclusions can be drawn from them.

A robot was then built with free movements of all of its joints, so that each could move through 360° (although not beyond ±180° in any direction), called Quad_full. A huge variety of different gaits evolved, as might be imagined, some of which are shown in Figure 7.3, though in general it was slightly slower than Quad_in.

The next robot tested had prismatic (sliding) joints instead of hinge joints at the knees, and again all of its legs were given the same controllers. It was called Quad_prism, and again easily outperformed Quad_in. Most of the robots evolved to walk on their knees, using only a small amount of their lower limbs to give them a slightly longer reach at the beginning of their stride (see Figure 7.4).

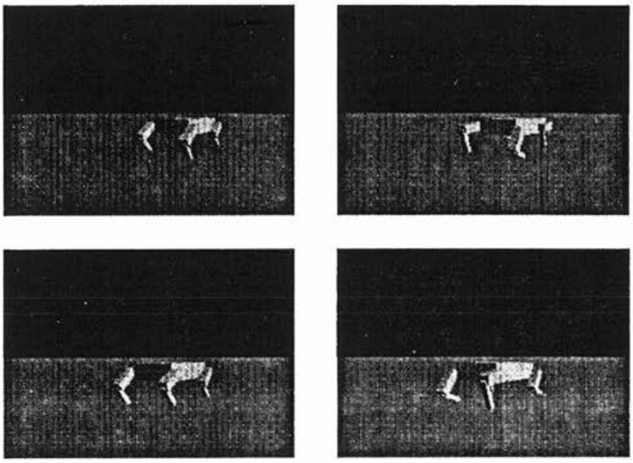

Figure 7.1: A fast controller for the Quad_same robot

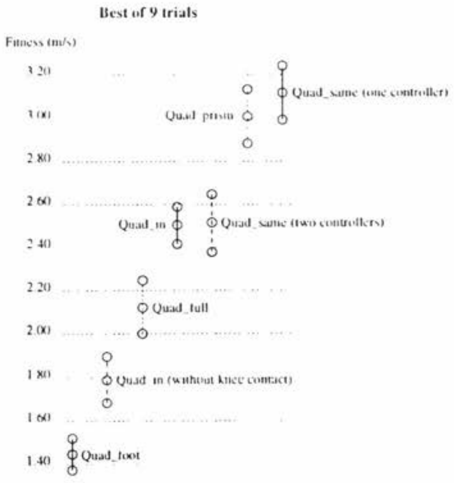

Figure 7.2: Average speed of best of 9 runs for each quadrupedal robot

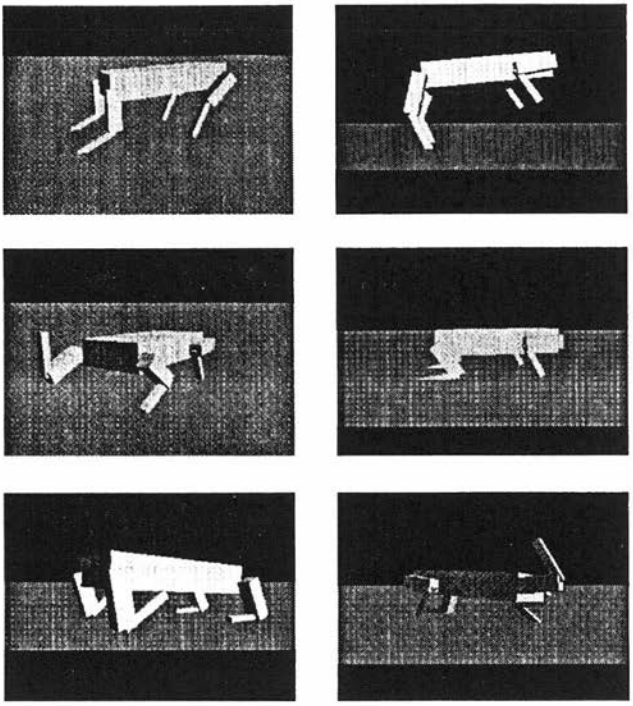

Figure 7.3: A variety of gaits for the Quad_full robot

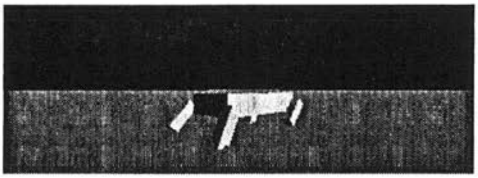

Figure 7.4: Quad_prism: a quadruped with prismatic knee joints

Finally an extra joint was added to the original robot to give it feet, and the new robot was called Quad_foot. Mostly the robot evolved to use one set of feet and not the other, as is seen in Figure 7.5, though running backwards as this one does was quite unusual. Generally, the robot was much slower than any of the previous ones.

One other experiment was done, penalising the robot for walking on its knee joints, by extending the definition of what was part of the body of the robot (touching the body on the ground was already penalised). The robot did learn to walk on its feet, but it was considerably slower than it was when walking on its knees. All of these results will be discussed further at the end of this section.

7.1.2 Different environments

The Quad_in robot was then put into a low friction and a low gravity environment and evolved to see whether it was able to deal with these changes. The results are shown in Figure 7.6.

The low friction environment made it easier for the robot to move, apparently because it did not have to worry about tripping up any longer, and skidding did not worry it at all — indeed many of the faster controllers start like a car starting a race with a great deal of skidding until the robot gets up to a fast enough speed. The low gravity on the other hand was much more problematic, with most of the robots falling over after a few steps, and having trouble adapting to the necessarily slow pace of walking in such conditions. This may well be simply because of a lack of time spent changing time constants or the strengths of the muscles however, which would allow the new robots to move more slowly.

7.1.3 Triped and Biped robots

Two final robots were built to test the ability of the system further, a triped and a biped (a hexapod was also acquired, but a design error and time pressure made it impossible to run experiments with it). Neither moved very fast (see Figure 7.7 for details), but the biped was very poor. Only a very few managed to develop any kind of gait, and none of these were at all satisfactory, nearly all falling onto their arms for

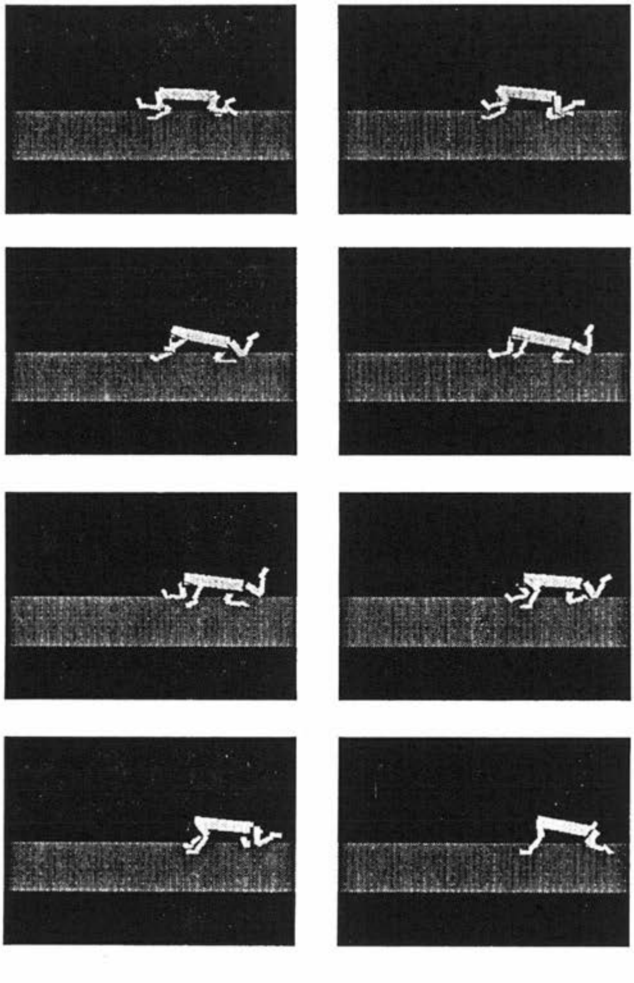

Figure 7.5: One of many controllers for the Quad_foot robot

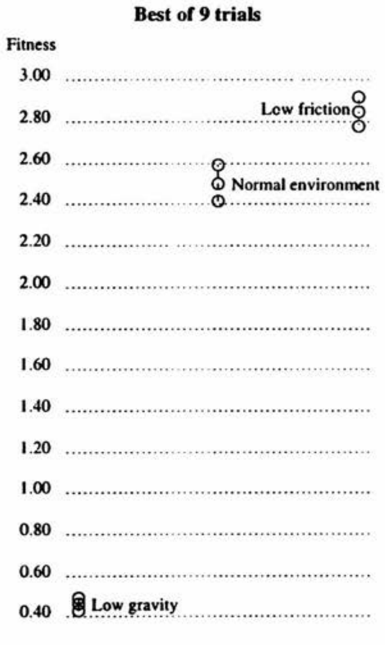

Figure 7.6: Average speed of best of 9 runs for different environments

support, and none alternating their legs in a sensible fashion. One of the few bipeds to stay on its feet is shown in Figure 7.8, and it is clear that this is not a useful gait to have learnt (it uses the momentum from swinging its arms to help it hop very slowly) — even here it fell over after a few hops. Some recent experiments done by Reil (1999) on a biped have shown that bipeds can learn to walk with a similar experimental setup, so it is likely that some parameters such as actuator strength just needed to be adjusted.

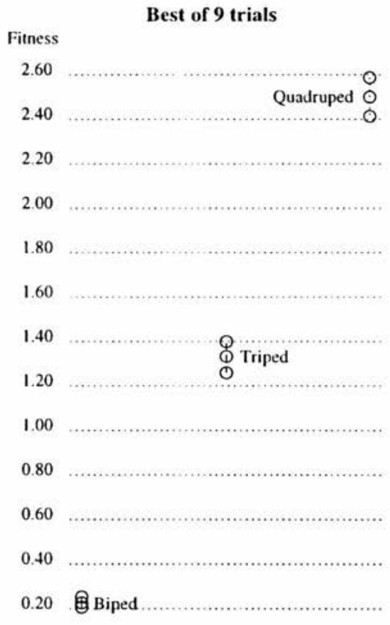

Figure 7.7: Average speed of best of 9 runs for triped and biped

The triped on the other hand was successful, but just not physically capable of moving at a great speed — a typical example of the triped is the bound shown in Figure 7.9, which the robot could keep up indefinitely. The robot was in general slightly more prone to falling over than others, but that is understandable given its design, and apart from that the system worked well at generating gaits for it.

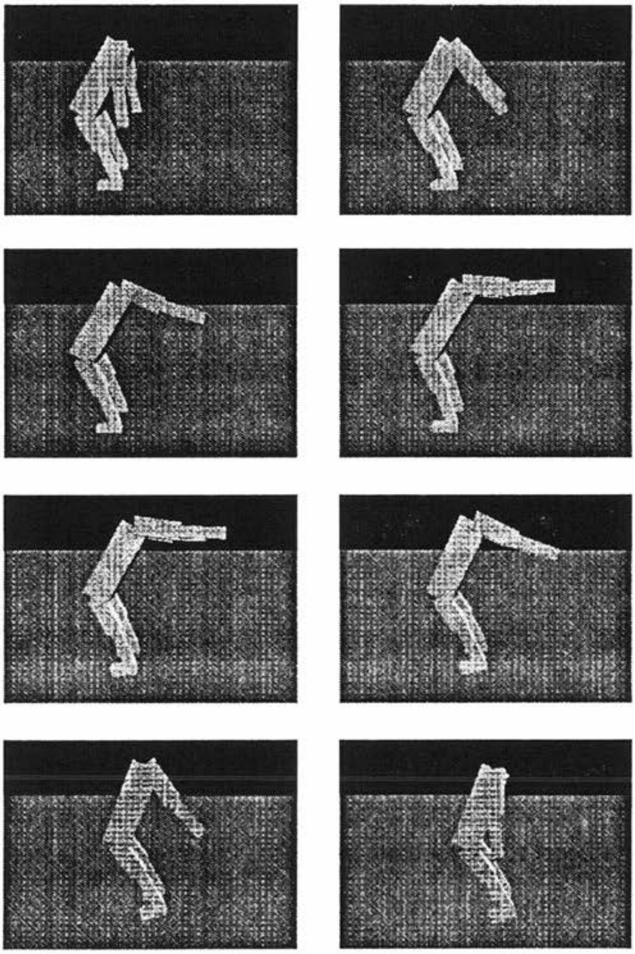

Figure 7.8: One of the better biped controllers!

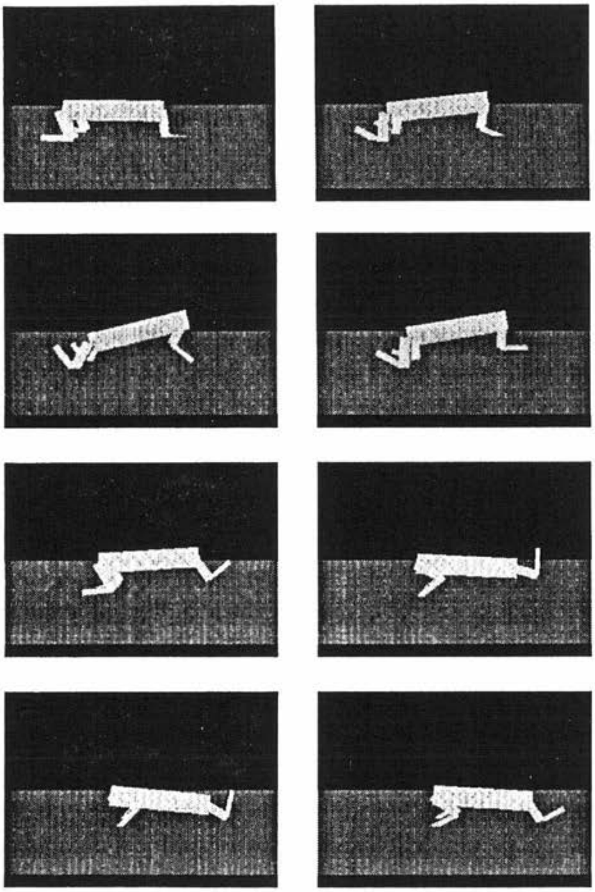

Figure 7.9: A typical triped

7.1.4 Discussion

Most of the robots moved well, with some of them significantly outperforming the original design. Of course such comparisons are largely meaningless as some robots can inherently move faster than others because of their limb configurations and strengths, but it is nonetheless an indication that the system is not specifically tailored to the original robot and can adapt to different ones.

Penalising the robots for going down onto their knees also succeeded in getting the models to walk on their feet more, though generally at some expense in speed. However this speed reduction may indicate that design problems with the original robot made it better able to move on its knees than its feet, particularly when it is considered that the fastest robot (which had all of its knee joints bending the same way) learnt to run on its feet without any penalties at all.

The only robot which really disappointed was the biped which, although it did generate repeating gaits which enabled the robot to move along, certainly couldn't be considered to have succeeded in generating useful walking gaits. There are two possibilities here - either the system broke down on the biped, or the biped was not itself well designed for walking (again). It is fair to say that the system did break down, and perhaps more work needs to be done to make control of bipeds easier, as they constitute a much more difficult balancing problem than quadrupeds, and extra features may have to be incorporated into the fitness evaluation. However, since occasional evolutionary runs in this thesis have accidentally generated bipedal or mostly bipedal gaits like knuckle walking in the Quad_in quadruped, I believe that it is likely that there is also a problem with the design of the robot, but that this is certainly not an insuperable problem. Indeed, it is entirely possible that a better designed biped would be capable of walking using just this system, and this is backed up by the very similar results which have just been obtained by Reil (1999). Unfortunately an alternative biped which was tried caused problems for DynaMechs due to the stiffness of the integration, so it was not possible to investigate this further.

One further experiment was tried which involved changing the conditions that the robot operated in. Changing the friction coefficient of the ground made very little difference, with the robot able to cope quite easily although it did slide a lot. However, changing the gravity to $3ms^{-2}$ did make it very difficult for the robot. Some gaits were evolved, but generally they were quite unsuccessful. Watching the robot, it was clear that it had trouble adapting to conditions in which it could push itself clear of the ground completely, and although moderating the strength of the motors did help, the legs still seemed to be moving too fast — perhaps the adaptation rate of the neurons needed to be lower in these conditions, but since we were using the fixed lamprey neuron parameters, it was decided not to investigate this any further at this point. This would seem in fact to be a case where the addition of a fitness function like minimising ground forces mentioning in Section 6.2 might be of practical value.

In general though, the system succeeded in generating a variety of gaits for the robots with which it was presented, with the fastest exceeding 10mph, a very reasonable speed. More research ought to be done using more realistic robot models however, and we will discuss this further in Section 8.2.

7.2 Active mechanisms and higher level controllers

The final test of the system is to see how easy it is to build high level controllers on top of it. In vertebrates, as we said in the introduction, simple signals can be passed down the spinal cord to the CPGs to change the speed or direction of movement, and we will try to add control signals which induce a turning movement or acceleration in the robot. More complicated tasks such as recovering from tripping up, or increasing stability when changing gaits require more sophisticated controllers, and so will be left for now. However in principle there is no problem in doing this — instead of simply adding more connections to the existing neurons, more neurons would be added to the network as well.

The best Quad_in controller was taken as the basis for all of these experiments, and connections were evolved from the control inputs to all of the neurons in the controller.

7.2.1 Steering

Two control inputs were used for this, with the connections being symmetrical so that the connections from the first input to the left of the body were the same as the connections from the second to the right and vice-versa. The connections for one CPG are shown in Figure 7.10. Since there were very few parameters being evolved (only 1 per neuron after symmetries have been taken into account, which makes 48 in total) the population and number of generations of the genetic algorithm were halved to 25 and 50 respectively. The fitness function was the integral of the product of the radial and angular velocities, to ensure that the robot continued to move forward as well as turning. Ten runs of the GA were carried out to see whether it was possible to evolve a steering controller, and nine of them succeeded. The best is shown in Figure 7.11.

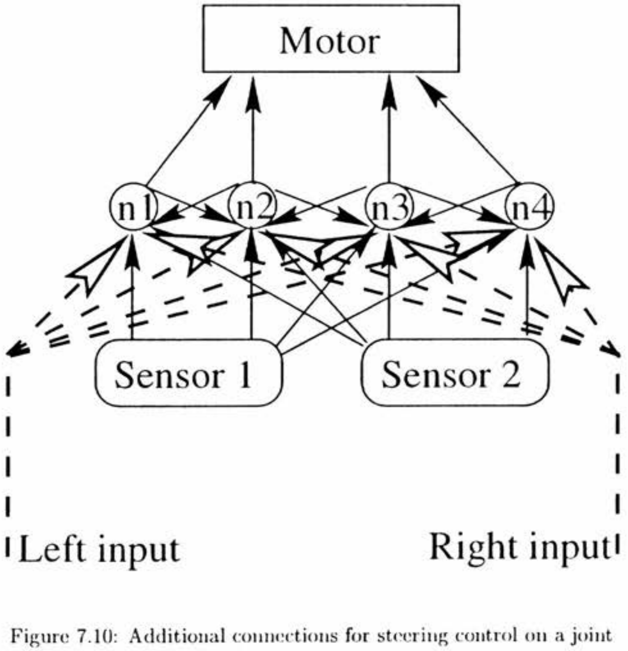

Figure 7.10: Additional connections for steering control on a joint

The robot can now be steered left (with inputs $+x$ and 0) and right (0 and $+x$) at will, with higher x causing faster turns. In fact giving a signal to both inputs slows down

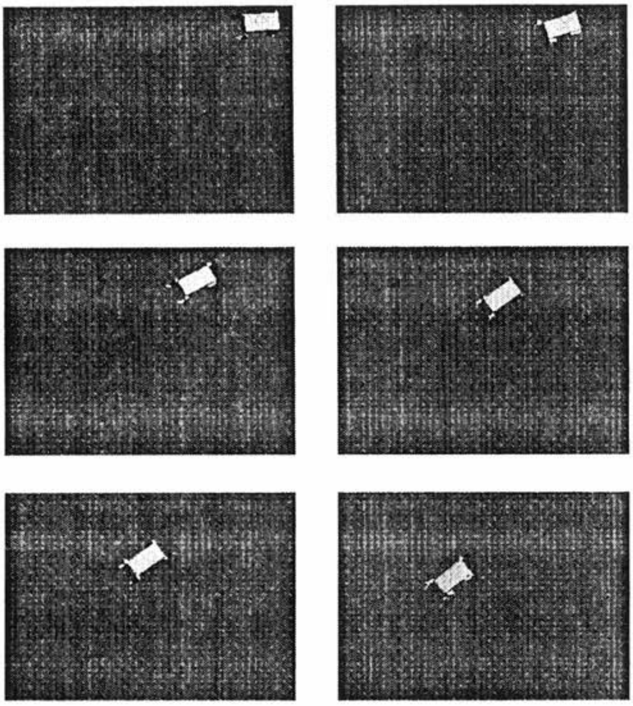

Figure 7.11: Turning left

the robot slightly by shortening the step length, so the inputs appear to be slowing one side of the robot or the other.

7.2.2 Slowing down

Since the fitness evaluations so far have all included a "maximise speed" element, it seemed unlikely that a control signal could be included which would speed up the robot any further, so it was decided to try to add a signal which would slow it down instead. The important feature here was that the robot should not fall over (a very effective way of slowing down!), so the fitness function used penalised moving forward above half the speed it managed without the control input, but penalised falling down much more heavily. The hope was the this would allow a control signal to specifically learn to reduce the speed of the robot but not stop it, and using a lower control input would allow variation between the two speeds achieved.

Stopping the robot was not attempted because it was felt that this would require a different low-level controller than that which had been evolved for walking, as standing still could use the muscles to keep the joint angles constant for instance, which would require a very different controller from walking slowly. This is in contrast to previous experiments such as that done by Kodjabachian and Meyer (1998) where walking slowly and stopping were very similar because the experiment was in 2-D which meant no balancing was required.

Again ten runs of the GA were performed with half the population and number of generations. With only one control input, and the connections symmetrical so that a weight to a neuron on the left was the same as the weight to the same neuron on the right, there were only 24 parameters to be evolved. This time all of the evolutionary runs succeeded in producing speed controllers, with most of them able to slow the robot down by about 1/5 and some up to 1/3 whilst maintaining a stable gait. Although this is not as much as might be expected, it seemed likely that more was possible, with perhaps more thought put into the fitness function, since many solutions involved slowing down the robot by the maximum 1/2 for the first 5 seconds that the evaluation ran over, and then slowly speeding up thereafter.

7.2.3 Combining controllers

A final experiment was carried out combining the best steering controller with the best speed controller on the same underlying neural network. This was possible because in both cases the connections were evolved as additions to the same network. The controllers combined easily on a single system, and the robot's direction and speed were then controllable together.

7.2.4 Discussion

The addition of higher level controllers to drive the active mechanism were very successful, with almost every single evolutionary run creating a successful controller. This was an excellent result, and quite unexpected; even more so was the ability to combine simple higher level controllers to make more sophisticated ones.

The controllers evolved were only simple ones to control speed and direction of locomotion, but the results strongly backed up the original contention that higher level control is easier to implement on a walking robot than a basic legged one, so much so indeed that they do not need any additional neurons to carry out its function. This may be because the basic walking behaviour is stable and so can be pushed easily by control inputs in a variety of directions (such as to allow turning or speed changes) without it becoming unstable. Some more sophisticated controllers, such as ones which could carry out transitions from walking to standing or running for instance, might require more complicated controllers, as indeed they do in vertebrates, but investigation into this more sophisticated kind of control is a subject for further research.

7.3 Conclusions

The system created in this thesis for evolving low level neural controllers for arbitrary walking robots works well, with some minor reservations concerning bipedal robots. Furthermore, attempts to add higher level controllers to the system in the form of a speed and a steering controller were successful on the first attempt, with only the simplest possible controllers being necessary to achieve the aims set out.

This opens up the possibility of further work to investigate how to improve the low level controller for bipeds, and to make more sophisticated high level controllers. This will be discussed further in Section 8.2.

Chapter 8

Conclusions

8.1 Summary of contributions

This thesis proposes a new control methodology for walking robots. Currently robots are controlled either by having springs and dampers built into the physical robot to produce an excellent walking robot, which is easily controlled but with little of the general purpose ability to climb over obstacles or manoeuvre which makes walking robots desirable, or having monolithic controllers designed for specific robots, which is a very difficult problem, not easily generalisable and thus very time consuming for each new robot.

Instead this thesis proposes simulating the springs and dampers of the passive robot through the motors to produce an active mechanism which is a walking robot in its own right. This eliminates the problem of the lack of multifunctionality of the passive robots, by allowing the active mechanism to be switched off or altered to cope with different conditions, but avoids the higher level controller having to "compensate for the limitations of a poorly designed mechanism" (*Raibert, ISToMM'93*), thus greatly simplifying the control task.

This control model is implemented using neural networks to provide the low level dynamic control of the motors, and a simple genetic algorithm to evolve the connections of the networks. It is shown that these tools allow simple active walking (and running) mechanisms to be built for a variety of robots very easily, and that adding higher level controllers for speed and direction using the same tools is very easy, as predicted in the introduction.

The thesis also investigates a number of issues in evolutionary neural networks. It finds that more sophisticated (and biologically plausible) neural models are significantly better at this kind of control task than the simpler models usually used. It is suggested that this could be a result of the greater richness of behaviour of individual neurons; this means that fewer neurons are needed for a given problem, which in turn simplifies the search for a solution.

A variety of neural encodings are also investigated to discover which connectivity patterns are best for this problem, with the expectation that connectivity somewhat like CPGs in vertebrates may turn out to be useful. However, results indicate that the best controllers have very dense intra- and inter-joint connectivity, and indeed that the ability of neurons to connect to more than one effector is very useful in building effective controllers; both of these results are unexpected, and perhaps hint that other factors such as simple physical separation and consequent time delays in signals may perhaps drive the connectivity patterns found in CPGs.

Finally an analysis is made of different types of fitness functions to determine which kind of fitness measures help in the development of active mechanisms. It is found that, despite their extensive use in the field, there is no evidence that fitness measures relating to the internal workings of neurons and joints are helpful in the design of good walkers. It is suggested that this is because we simply don't know what these neurons and joints ought to be doing in a good walking gait, but instead it is sufficient to specify broad outlines of the task, such as maximising speed whilst keeping the body off the ground, and that such simple fitness measures can produce the desired results on their own.

8.2 Future Directions

The experiments done in this thesis were of necessity exploratory, and some elements were only touched on. In particular little energy was expended on optimising the evolutionary algorithm, and it would undoubtedly be useful to examine this in more depth. A more serious consideration is that the robot models used were made up on the spur of the moment to represent real or imagined generic classes of robots (the triped will sadly probably never make it onto a production line!). It would be extremely instructive to try out the system on a real robot, as it was obvious in retrospect that many of the robots were poorly designed. This was not practical during the thesis for a number of reasons — the lack of a legged robot to use as a model, the complicated nature of real robots, for which inertia matrices and the like have to be computed for the simulator, and the modelling of the motors which would be difficult and entirely unrelated to the research being carried out. However, it is the purpose of the control model to be used on a real robot, so it would be very interesting to see how it coped, and whether the controller could pass across from simulation onto the actual robot.

It would also be a useful extension to the work to see how easily other, more complicated, higher level controllers could be added to the system, to increase the stability of the system for instance, or to allow transitions from one gait to another.

Finally it would be a very interesting project to try modelling a real vertebrate, perhaps a large cat or even a human, using anatomical information and sophisticated muscle models which are easily available (far more so than for robots), and to see if gaits similar to those found in nature could be duplicated on the simulant. There is every reason to believe they would, given the similarity of many of the gaits found in the quadrupeds studied here to natural ones. If this was successful further work could investigate which optimisation criteria produced which gaits in which animals, providing insights which could then be used in zoological research such as the inverse optimality problem discussed in Section 2.2.5. Information could also be gathered directly from the simulant for instance as to the energy efficiency of various gaits which can only be roughly estimated on real animals.

Bibliography

Ackley, D. and Littman, M. (1992). Interactions between learning and evolution. In *Proceedings of the 2nd conference on Artificial Life*, pages 487 – 509. Addison-Wesley.

Alexander, R. (1984). The gaits of bipedal and quadrupedal animals. *Int. J. Rob. Res.*, **3**(2), 49–59.

Alexander, R. (1990). Three uses for springs in legged locomotion. *Int. J. Rob. Res.*, **9**(2), 53–61.

Alexander, R. (1991). Energy saving mechanisms in walking and running. *J. Exp. Biol.*, **160**, 55–69.

Angeline, P. J. (1993). *Evolutionary Algorithms and Emergent Intelligence*. Ph.D. thesis, Ohio State University.

Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1993). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*.

Assaiante, C. and Amblard, B. (1995). An ontogenetic model for the sensorimotor organization of balance in humans. *Human Movement Science*, **14**, 13–43.

Beer, R. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, **3**(4), 469–509.

Beer, R. and Gallagher, J. (1992). Evolving dynamic neural networks for adaptive behavior. *Adaptive Behavior*, **1**, 91–122.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Boers, E., Kuiper, H., Happel, B., and Sprink huizen Kuyper, I. (1993). Designing modular artificial neural networks. In H. Wijshoff, editor, *Proceedings of Computing Science in The Netherlands*, pages 87–96, SION, Stichting Mathematisch Centrum.

Boers, E. J. and Kuiper, H. (1992). *Biological Metaphors and the design of modular artificial neural networks*. Master's thesis, Leiden University, The Netherlands.

Boone, G. and Hodgins, J. (1995). Reflexive response to slipping in biped running robots. In *IROS*, volume 3, pages 158–194.

Boone, G. and Hodgins, J. (1997). Slipping and tripping reflexes for bipedal robots. *Autonomous Robots*, **4**, 259–271.

Brooks, R. A. (1989). A robot that walks; emergent behaviors from a carefully evolved network. AI memo 1091, MIT.

Cangelosi, A., Parisi, D., and Nolfi, S. (1994). Cell division and migration in a 'genotype' for neural networks. *Network: Computation in Neural Systems*, (5), 497 – 515.

Cheng, M.-Y. and Lin, C.-S. (1996). Measurement of robustness for biped locomotion using a linearized Poincaré map. *Robotica*, **14**, 253–259.

Cohen, P. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press.

Collins, J. and Richmond, S. (1994). Hard-wired central pattern generators for quadrupedal locomotion. *Biol. Cybernetics*, **71**, 375–385.

Collins, J. and Stewart, I. (1993a). Coupled non-linear oscillators and the symmetries of animal gaits. *J. Nonlinear Sci.*, **3**, 349–392.

Collins, J. and Stewart, I. (1993b). Hexapodal gaits and coupled non-linear oscillator models. *Biol. Cybernetics*, **68**, 287–293.

Collins, R. J. and Jefferson, D. R. (1990). An artificial neural network representation for artificial organisms. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 259–263.

Cruse, H., Brunn, D., Bartling, C., Dean, J., Dreifert, M., Kindermann, T., and Schmitz, J. (1995). Walking: a complex behaviour controlled by simple networks. *Adaptive Behavior*, **3**, 385–418.

de Garis, H. (1990a). Genetic programming: Artificial nervous systems artificial embryos and embryological electronics. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 117–123.

de Garis, H. (1990b). Genetic programming: Building artificial nervous systems using genetically programmed neural network modules. In *Proceedings of the 7th Inernational Conference on Machine Learning*.

Delcomyn, F. (1980). Neural basis of rhythmic behaviour in animals. *Science*, **210**, 492–498.

Eilam, D. (1995). Comparative morphology of locomotion in vertebrates. *J. Motor Behavior*, **27**, 100–111.

Ekeberg, O. (1993). Neural control of vertebrate locomotion: A computer simulation study. In B. Svensson, editor, *Int. Work. on Mechatronical Computer Systems for Perception and Action*, Halmstad University, Sweden.

Elliott, P. (1997). Personal communication.

Featherstone, R. (1984). *Robot dynamics algorithms*. Ph.D. thesis, Department of Artificial Intelligence, University of Edinburgh.

Fullmer, B. and Miikulainen, R. (1991). Using marker-based genetic encoding of neural networks to evolve finite-state behaviour. In *Proceedings of the 1st European Conference on Artificial Life, Paris, France.*

Gallagher, J. and Beer, R. (1992). A qualitative dynamic analysis of evolved locomotion controllers. In *From Animals to Animats II*, pages 71–80.

Gallagher, J., Beer, R., Espenscheid, K., and Quinn, R. (1996). Applications of evolved locomotion controllers to a hexapod robot. *Robotics and Autonomous Systems*, **19**, 95–103.

Goldberg, D. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.

Goswami, A., Espiau, B., and Keramane, A. (1997). Limit cycles in a passive compass gait biped and passivity-mimicking control laws. *Autonomous Robots*, **4**, 273–286.

Gray, J. and Lissmann, H. W. (1940). *Journal of Experimental Biology*, **17**, 237.

Gray, J., Lissmann, H. W., and Pumphrey, R. J. (1938). *Journal of Experimental Biology*, **15**, 408.

Grillner, S. (1985). Neurobiological bases of rhythmic motor acts in vertebrates. *Science*, **228**, 143–149.

Grillner, S. and Wallén, P. (1977). *Brain Research*, **127**, 291.

Grillner, S., Wallén, P., and Brodin, L. (1991). Neuronal network generating locomotor behaviour in lamprey: Circuitry, transmitters, membrane properties, and simulation. *Annual Review of Neuroscience*, **14**, 169–199.

Gruau, F. (1994a). *Automatic Definition of Sub neural networks*. Ph.D. thesis, Ecole Normale Supérieure de Lyon.

Gruau, F. (1994b). Automatic definition of sub neural networks. Technical Report 94-28, Ecole normale Supérieure de Lyon.

Gunter, B. (1991). Bootstrapping: How to make something from almost nothing and get statistically valid answers. 1. Brave New World. *Quality Progress*, pages 97–103.

Gurfinkel, V., Gurfinkel, E., Schneider, A., Devjanin, E., Lensky, A., and Shitilman, L. (1981). Walking robot with supervisory control. *Mechanism and Machine Theory*, **16**, 31–36.

Harp, S. A., Samad, T., and Guha, A. (1989). Towards the genetic synthesis of neural networks. In *Proceedings of the 3rd International Conference on Genetic Algorithms*.

Hirose, S. and Umetani, Y. (1980). The basic motion regulation system for a quadruped walking vehicle. In *ASME Conf. on Mechanisms*.

Hodgins, J. (1996). Three-dimensional human running. In *IEEE Int. Conf. on Robotics and Automation*, pages 3271–3276.

Honda Motor Company Ltd., Tokyo (1997). The Honda humanoid robot. http://www.honda.co.jp/english/technology/robot/.

Ijspeert, A., Hallam, J., and Willshaw, D. (1997). Artificial lampreys: Comparing naturally and artificially evolved swimming controllers. In *European Conference on Artificial Life*, pages 256–265.

Jalics, L., Hemami, H., Clymer, B., and Groff, A. (1997). Rocking, tapping and stepping: A prelude to dance. *Autonomous Robots*, pages 227–242.

Kelso, J. (1995). *Dynamic Patterns: the Self-Organization of Brain and Behavior*. MIT Press.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, **4**, 461–476.

Kitano, H. (1995). Cell differentiation and neurogenesis in evolutionary large scale chaos. In *Proceedings of the 3rd European Conference on Artificial Life*, pages 341–352.

Ko, H. and Badler, N. (1996). Animating human locomotion with inverse dynamics. *IEEE Computer Graphics and Applications*, **16**, 50–59.

Kodjabachian, J. and Meyer, J.-A. (1995). Evolution and development of control architectures in animats. *Robotics and Autonomous Systems*, **16**, 161–182.

Kodjabachian, J. and Meyer, J.-A. (1998). Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. *IEEE Transactions on Neural Networks*, **9**, 796–812.

Koopman, L. (1987). *Introduction to contemporary statistical methods*. Buxbury Press, Boston, 2nd edition.

Koza, J. R. (1992). *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press.

Koza, J. R. (1994). *Genetic Programming II: automatic discovery of reuseable programsnolfi*. MIT Press.

Lewis, M. A. (1996). *Self-organization of locomotory controllers in robots and animals*. Ph.D. thesis, University of Southern California.

Lewis, M. A., Fagg, A. H., and Solidum, A. (1992). Genetic programming approach to the construction of a neural network for control of a walking robot. In *Proceedings of the 1992 IEEE International conference on Robotics and Automation, Nice, France*, pages 2618–2623.

Lindenmeyer, A. (1968). Mathematical models for cellular interaction in development, part i and part ii. *Journal of Theoretical Biology*, **18**, 280–315.

Luk, B., Collie, A., and White, T. (1993). Nero: a teleoperated wall climbing vehicle for assisting inspection of a nuclear reactor pressure vessel. In *ASME Int. Comp. Eng. Conf.*

Luk, B., Cooke, D., Collie, A., and White, T. (1994). Robug iii: a semi-intelligent teleoperated walking and climbing robot for disordered and hazardous environments. In *European Robotics and Intelligent Systems Conf.*

Maniezzo, V. (1994). Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions in Neural Networks*, **5**(1).

Marey, E. (1874). *Animal Mechanisms: a treatise on terrestrial and aerial locomotion.* Appleton, New York.

Marsolais, E. and Kobetic, R. (1983). Functional walking in paralysed patients by means of electrical stimulation. *Clin. Orthoped. Rel. Res.*, **175**, 30–36.

Matsuoka, K. (1979). A model of repetitive hopping movements in man. In *Proceeding of Fifth World Congress on Theory of Machines and Mechanisms*. International Federation for Information Processing.

McGeer, T. (1989). Powered flight, child's play, silly wheels and walking machines. In *Proceeding of the 1989 IEEE International Conference on Robotics and Automation*, pages 1592–1597.

McGeer, T. (1990). Passive dynamic walking. *Int. J. Rob. Res.*, **9**(2), 62–82.

McGhee, R. B. (1976). Robot locomotion. In R. Herman, S. Grillner, P. Stein, and D. Stuart, editors, *Neural Control of Locomotion*, pages 237–264. Plenum Press.

McMillan, S. (1994). *Computational Dynamics for Robotic Systems on Land and under Water*. Ph.D. thesis, Ohio State University.

Miall, R. C., Weir, D. J., Wolpert, D. M., and Stein, J. F. (1993). Is the cerebellum a smith predictor? *Journal of Motor Behavior*, **25**(3), 203–216.

Michel, O. and Biondi, J. (1995a). From the chromosome to the neural network. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*.

Michel, O. and Biondi, J. (1995b). Morphogenesis of neural networks. *Neural Processing Letters*, **2**(1).

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms 3*, pages 379–384.

Miura, H., Shimoyama, I., Mitsuishi, M., and Kimura, H. (1984). Dynamic walk of a biped. *International Journal of Robotics Research*, **3**(2), 60–74.

Mjolsness, E., Sharp, D. H., and Alpert, B. K. (1987). Recursively generated neural networks. Research Report YALEU/DCS/RR-549, Department of Computer Science, Yale University.

Moriarty, D. E. and Miikulainen, R. (1994). Efficient reinforcement learning through symbiotic evolution. Technical Report AI94-224, University of Texas at Austin.

Mosher, R. S. (1968). Testing and evaluation of a versatile walking truck. In *Proceeding of the Off-Road Mobility Research Symposium*. Int. Soc. for Terrain Vehicle Systems.

Muybridge, E. (1887). *Animals in Motion*.

Narendra, K. and Parthasarathy, K. (1989). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, **1**(1).

Nolfi, S. and Parisi, D. (1992). Growing neural networks. Technical report, Institute of Psychology, National Research Council.

Nolfi, S. and Parisi, D. (1993). Phylogenetic recapitulation in the ontogeny of artificial neural networks. Technical report, Institute of Psychology, National Research Council.

Nolfi, S. and Parisi, D. (1995). "genotypes" for neural networks. In M. A. Arbib, editor, *The handbook of brain theory and neural networks*. MIT Press.

Nolfi, S., Floreano, D., Miglino, O., and Mondada, F. (1994a). How to evolve autonomous robots: different approaches in evolutionary robotics. In R. Brooks and P. Maes, editors, *Artificial Life IV proceedings*, Cambridge, MA. MIT Press.

Nolfi, S., Miglino, O., and Parisi, D. (1994b). Phenotypic plasticity in evolving neural networks. In P. Gaussier and J.-D. Nicoud, editors, *Proceeding of the Intl. Conf. From Perception to Action*.

Ogo, K., Kanse, A., and Kato, I. (1980). Dynamic walking of biped walking machine aiming at completion of steady walking. In A. Morecki, G. Bianchi, and K. Kedzior, editors, *Third Symposium on Theory and Practice of Robots and Manipulators*. Elsevier Scientific Publishing Co.

Perkins, S. J. (1999). *Incremental Acquisition of Complex Visual Behaviour using Genetic Programming and Robot Shaping*. Ph.D. thesis, University of Edinburgh.

Playter, R. (1994). *Passive Dynamics in the Control of Gymnastic Maneuvers*. Ph.D. thesis, MIT.

Plustech Oy., Finland (1997). Plustech web. http://www.plustech.fi/.

Raibert, M., Brown, H., and Chepponis, M. (1984). Experiments in balance with a 3d one-legged hopping machine. *Int. J. Rob. Res*, **3**(2), 75–92.

Raibert, M. H. (1986). *Legged Robots That Balance*. MIT Press.

Raibert, M. H. (1988). Dynamically stable legged locomotion. Technical Report 1179, MIT.

Reeve, R. (1999a). *Generating walking behaviours in legged robots*. Ph.D. thesis, University of Edinburgh.

Reeve, R. (1999b). *Walking Robots, a step forward*. Kluwer Academic Publishers.

Reeve, R. E. (1994). *Control of walking by Central Pattern Generators*. Master's thesis, Department of Artificial Intelligence, University of Edinburgh.

Reeve, R. E. and Hallam, J. (1995). Control of walking by central pattern generators. In *Proceeding of the 3rd Conference on Intelligent Autonomous Systems*.

Reil, T. (1999). *Artificial Evolution of Neural Controllers in a Real-Time Physics Environment*. Master's thesis, COGS, University of Sussex.

Rosner, B. (1982). *Fundamentals of biostatistics*. Duxbury Press, Boston.

Rotaru-Varga, A. (1999). Modularity in evolved artificial neural networks. In *ECAL'99 — Advances in Artificial Life*, pages 256 – 260. Springer.

Rumelhart, D. and McClelland, J. (1986). *Parallel Distributed Programming*, volume 1 and 2. MIT Press.

Sharp, D. H., Reinitz, J., and Mjolsness, E. (1991). Genetic algorithms for genetic neural nets. Technical Report YALEU/DCS/TR-845, Department of Computer Science, Yale University.

Sims, K. (1994a). Evolving 3d morphology and behaviour by competition. In R. Brooks and P. Maes, editors, *Artificial Life IV Proceedings*, pages 28 – 39, Cambridge, MA. MIT Press.

Sims, K. (1994b). Evolving virtual creatures. In *Proceedings of SIGGRAPH '94*, Annual Conference Series, pages 15–22. Computer Graphics.

Sony Corporation (1999). http://www.world.sony.com/robot/top.html.

Spencer, G. (1994). Automatic generation of programs for crawling and walking. In K. J. Kinnear, editor, *Advances in Genetic Programming*.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge MA.

Taga, G. (1995). A model of the neuro-musculo-skeletal system for human locomotion. *Biological Cybernetics*, **73**, 97–111.

Taga, G., Yamaguchi, Y., and Shimizu, H. (1991). Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, **65**(3), 147–159.

Torreele, J. (1991). Temporal processing with recurrent networks: An evolutionary approach. In *Proceedings of the 4th International Conference on Genetic Algorithm*, pages 555–561.

Utecht, U. and Trint, K. (1994). Mutation operators for structural evolution of neural networks. In *Proceedings of the 3rd Workshop on Parallel Problem Solving from Nature*.

Vaal, J., van Soest, A., and Hopkins, B. (1995). Modelling the early development of bipedal locomotion: A multidisciplinary approach. *Human Movement Science*, **14**, 609–636.

van de Panne, M. (1996). Parameterized gait synthesis. *IEEE Computer Graphics and Applications*, **16**(2), 40–49.

van Soest, A. and van Galen, G. (1995). Coordination of multi-joint movements: An introduction to emerging views. *Human Movement Science*, **14**, 391–400.

Wadden, T., Örjan Ekeberg, and Lansner, A. (1993). Towards ann based control of simulated legged locomotion. In B. Svensson, editor, *Int. Work. on Mechatronical Computer Systems for Perception and Action*, pages 379–382, Halmstad University, Sweden.

Wallén, P., Ekeberg, O., Lansner, A., Brodin, L., Travén, H., and Grillner, S. (1992). A computer based model for realistic simulations of neural networks. II. the segmental network generating locomotor rhythmicity in the lamprey. *Journal of Neurophysiology*, pages 1939–1950.

Wettergreen, D., Thorpe, C., and Whittaker, R. (1993). Exploring mount erebus by walking robot. *Robotics and Autonomous Systems*, **11**, 171–185.

Whitley, D. and Hanson, T. (1989). Optimising neural networks using faster, more accurate genetic search. In J. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications*, pages 391–396.

Wieland, A. P. (1991). Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks, Seattle*, volume 2, pages 667–673.

Yamaguchi, G. and Zajac, F. (1990). Restoring unassisted natural gait to paraplegics via functional neuromuscular stimulation: A computer simulation study. *IEEE Trans. Biomed. Eng.*, **37**, 886–902.

Yoneda, K., Iiyama, H., and Hirose, S. (1996). Intermittent trot gait of a quadruped walking machine dynamic stability control of an omnidirectional walk. In *IEEE Int. Conf. on Robotics and Automation*, pages 3002–3007.

Appendix A

Controller breakdown

Unfortunately the neural networks used in most of the experiments were too big to analyse, being from 50 to 100 neurons all fully connected to each other. However, the controller used in Section 7.2 was small network, as all of the legs had the same controller and there were only 6 neurons per actuator. This meant that there were 48 neurons, but only 12 unique ones. The experiment was using the Quad_same robot described in Section 7.1.1. This Appendix will briefly look at the kind of behaviour this network exhibited, and will show how it responded to various degradations (such as sensors or neurons being removed).

A.1 Unmodified network behaviour

When the robot is allowed to walk normally without any alteration to network or the sensors, walking starts smoothly and continues very stably at roughly $4.3ms^{-1}$. The four sets of twelve neurons all behave identically with pairs almost perfectly synchonized, and each pair in antiphase to the other. This results in a diagonal trot, with the near forelimb and the off hindlimb in phase, and the near hind and off fore in antiphase. For example Figure A.1 shows two seconds recording of the third neuron in each set of twelve (as we have said in describing the third order neurons in Section 5.3.4, all the neurons have an output which varies either between 0 and 1 or 0 and -1).

Only two signals can be readily distinguished, but all four can be seen in the enlargement of Figure A.2. The differences are insignificant however, and vary from step to step, though they seem to decrease over time.

Looking at a whole set of twelve neurons in Figure A.3 we immediately see that only five of them are active, and this leads us to believe that perhaps the other seven neurons are not useful. However if we look at the first second when the robot sets off from its initial stationary pose, three other neurons are also active (Figure A.4). We shall see later that these are also important. However four of the neurons are completely inactive, and it seems likely these could be removed with no detrimental effect.

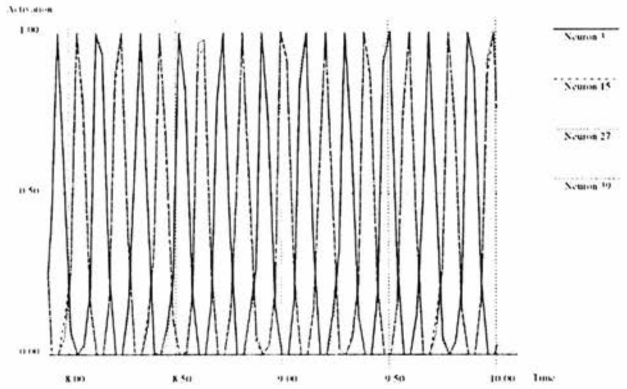

Figure A.1: Recording of all four neuron 3 signals

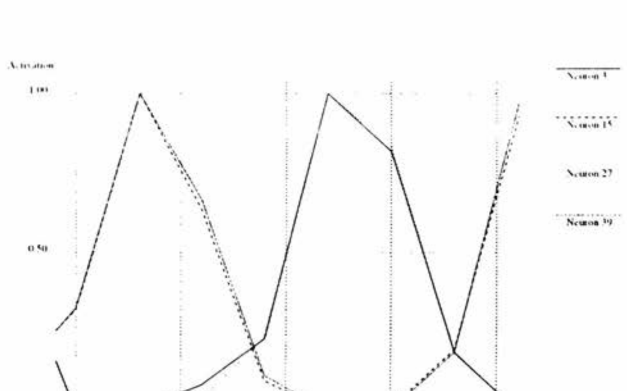

Figure A.2: Detail of 0.2s of neuron 3 signal

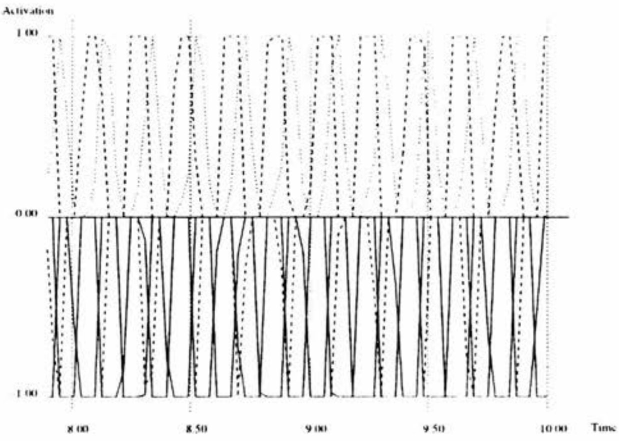

Figure A.3: Behaviour of a maximal unique set of neurons

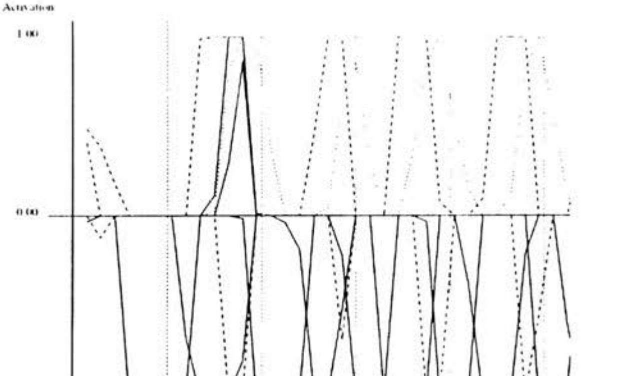

Figure A.4: Starting behaviour of set of neurons

A.2 Killing neurons

The first experiments to degrade the network were to remove individual neurons from it while the robot was already trotting (it was allowed to run for 2 seconds before the neurons were killed) and analyse the behaviour. The results are shown in Table A.1.

Neuron	Effect on locomotion for each neuron group
1	$++++$
2	$++++$
3	$----$
4	$++++$
5	$\pm++\pm$
6	$++++$
7	$----$
8	$----$
9	$++++$
10	$++++$
11	$----$
12	$++++$

where $+$ indicates continued movement, $-$ stopped, and $\pm$ indicates a marked effect, such as slewing to one side.

Table A.1: Result of removal of neurons on continued movement

It is clear from the results that four of the neurons in each group are crucial for the generating a normal locomotion pattern, with a typical movement pattern for a robot deprived of one of these neurons being to fall unsteadily onto one leg, and then stop moving completely. There seemed to be no correlation between the group of the neuron removed and the leg that was fallen onto, however. Unsurprisingly these are four of the five neurons which oscillated ordinarily during movement; the other was neuron 5, and removing this seriously affected the walk in two cases where the robot swung heavily to one side and limped slightly. There was very little effect on neural behaviour outside the group the neurons were in, but inside the group the output from neuron 3 reduced dramatically, as can be seen in Figure A.5 — similar neural behaviour was seen in all four cases, although in only two of them was there a visible effect on the locomotion. In the undamaged network the two neurons were in phase, and there was a strong excitatory connection from 5 to 3, which would explain the diminution in the output from that neuron. It is possible that the effects on the two neurons cancelled each other out to some extent.

The next experiment was to remove groups of neurons to see whether there was some cumulative effect, but there was very little — unsurpisingly all of the neurons inactive during walking (numbers 1, 2, 4, 6, 9, 10, and 12 in each set) could be removed simultaneously without affecting the movement at all, leaving only 20 neurons stably controlling the locomotion. The four "Neuron 5"s could not be removed simultaneously, however sometimes 2 could be removed without the walking pattern braking down.

After this the experiment was changed slightly to investigate the effect of removing

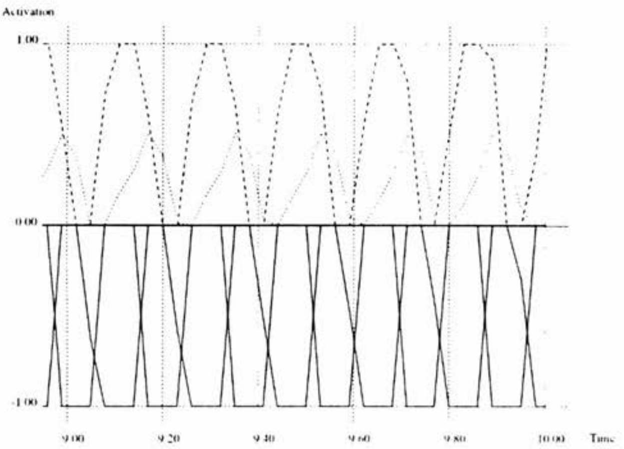

Figure A.5: Effect on a group of neurons of removing Neuron 5

neurons on locomotion from a standing start, as this is what is done in the evolutionary runs. Exactly the same experiments were done, except that the neurons were killed immediately. The results are shown in Table A.2.

Nine extra neurons became critical now, with three others making initialising walking very unstable. Four of the neurons in each group (6, 9, 10 and 12) still have no effect under any circumstances, meaning that 16 neurons are completely unused by the controller. These neurons are never seen to be active, and can all be removed simultaneously with no adverse effects. Of the remaining neurons which were not fatal to locomotion before, more than half now are. In all of the experiments so far the robot has started with the near fore and off hind limb back slightly and the others forward; reversing this starting position now reverses the effects of neuron 4 — the neurons from the first and third groups are now crucial whereas those from the second and fourth are not (note that the order of the legs is off hind, near hind, near fore, off fore, so 1 and 2 are the hind limbs, 1 and 3 is one diagonal pair and 2 and 4 is the other). On the other hand that had no effect on neurons 1 and 2, which seemed to only be important for the forelimbs. Trying further initial orientations of limbs (near side forward, off back, *etc.*), showed that all of the Neuron 4 neurons could be important depending on the initial position of the limbs, as could the remaining Neuron 5, but Neurons 1 and 2 in the first two groups were very rarely useful. Indeed generally the unimportant Neurons 1 and 2 and all of 6, 9, 10 and 12 could be removed with only some instability, but it was often difficult to remove any of Neurons 4 or 5 with any others without the robot failing to set off.

Neuron	Effect on locomotion for each neuron group
1	$\pm + --$
2	$\pm + --$
3	$----$
4	$+-+-$
5	$\pm ---$
6	$++++$
7	$----$
8	$----$
9	$++++$
10	$++++$
11	$----$
12	$++++$

where $+$ indicates continued movement, $-$ stopped, and $\pm$ indicates a marked effect, such as an unstable start.

Table A.2: Result of removal of neurons on initiating movement

A.2.1 Summary

Four of the twelve unique neurons (6, 9, 10 and 12) served absolutely no purpose in any group, two (1 and 2) were essential for starting (though not for continuing) in only two groups, but helped initial stability in the other groups: one (4) other was essential for starting in all groups depending on what the initial position of the robot was, but unimportant for continuing; another (5) was essential for starting, and helped stability when moving normally, and the remaining four (3, 7, 8 and 11) were essential to create a gait at all.

A.3 Killing Sensors

As a further experiment the sensors were killed one by one to see whether they affected the ability of the robot to set off or continue walking (for setting off, we looked at two different starting positions). The results are shown in Table A.3.

The sensors detecting the orientation of the robot had no noticeable effect on the walking, presumably because they varied very little compared to the other sensors. The sensor for the height of the robot off the ground (number 6) was crucial, perhaps because it was acting as a constant excitatory input to the network. Of the other inputs, which were all sensors on the legs, in general the gated inputs (which were gated by whether the foot of that limb was touching the ground) helped to stabilise the locomotion, but very few of them were essential (one of the two forelimb shin angles was needed for starting depending on the initial position of the robot, but nothing else). On the other hand nearly all of the ungated inputs were essential for starting, with the particular ones needed depending on the initial configuration of the legs; when already walking generally the shin angles were very important, as were the thigh angles of the

Sensor	Effect on starting	Effect on continuing
Orientation of robot		
1	+	+
2	+	+
3	+	+
Position of robot		
Sensors 4 and 5 would have been the x and y coordinates of the robot but were not passed to the neurons		
6	−	−
Off hind thigh angle, ungated and gated		
7	∓	−
8	+	+
Off hind shin angle, ungated and gated		
9	−	−
10	+	+
Near hind		
11	±	−
12	+	+
13	−	−
14	+	+
Near fore		
15	∓	+
16	+	+
17	−	+
18	±	+
Off fore		
19	±	+
20	+	+
21	−	−
22	∓	+

where + indicates success, − failure, and ± or ∓ indicates that it is sometimes required

Table A.3: Result of removal of sensors on initiating and continuing movement

hind limbs, but the forelimb thigh angles were not important. Interestingly, if starting from stationary the hind thigh angles were less important, but this may perhaps be a feature of the fact that the robot is going faster when it starts with no injuries than when it starts already damaged, and it cannot cope with the speed when injured and so trips up. Another point that was noticed is that the robot often slightly drags the leg which it cannot sense properly.

Interestingly, looking at the neural activity when some of the sensors have been removed, neurons which were not previously active become active (see Figure A.6, for example). The extra active neuron here turns out to be one of those which is ordinarily only active when setting off, and as the sensors are progressively damaged, it appears that they can help to alleviate the damage. For instance if two sensors are removed, say the two near forelimb ungated thigh and shin sensors, and we then try continuing to move, the robot manages to keep going with a lot of extra neural activity (Figure A.7), but if these neurons normally only active when setting off are removed as well, then the robot fails to maintain a rhythm.

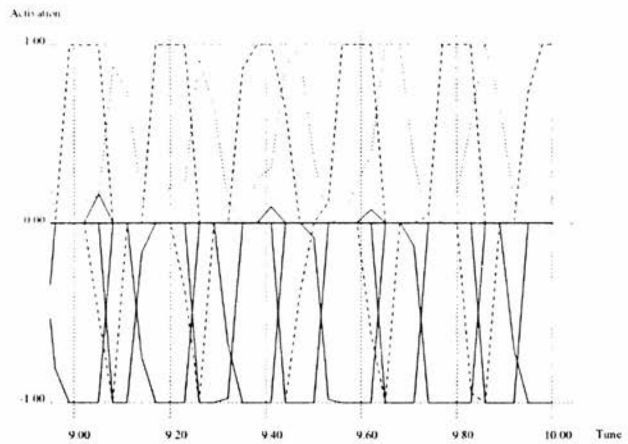

Figure A.6: Extra neural activity when sensor 15 is removed during locomotion

Further investigation of the damage that can be done to the sensors showed that all of the gated inputs and the orientation sensors could be removed simultaneously and the robot would still be able to walk, but removing two of the ungated inputs could only be done if (they were not ones which were crucial anyway and) the extra neurons were present.

A.3.1 Summary

The controller is robust to some damage being done to the sensors, particularly the gated inputs, but the ungated inputs are very important to maintaining a stable gait.

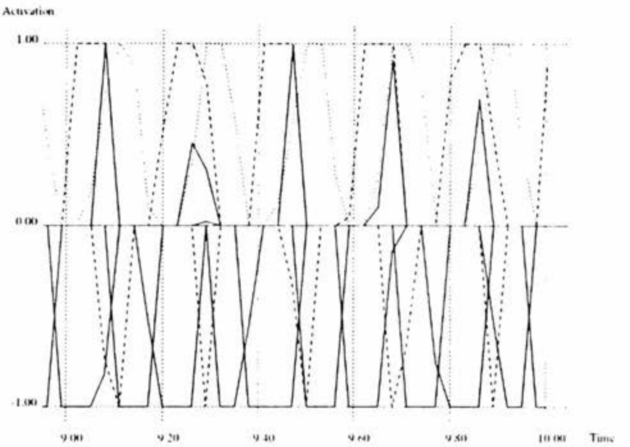

Figure A.7: More activity when sensors 15 and 17 are removed during locomotion

However the neurons which are normally only active when setting off from a stationary pose were found to be active and to some extent to replace lost sensors, albeit not perfectly.

A.4 Damaging the actuators

The final experiment that was carried out was to damage or remove one of the actuators on the robot to see the effect it had. There were very few to damage and the results were quite simple — the hind limbs could not be altered by more than about 30% without the robot failing to set off and only one joint could be damaged that much at a time, but when affected by less than that the robot would just limp slightly; with the forelimbs on the other hand two joints could be damaged by that much, or one by as much as 50% before the robot would trip over or fail to set off, otherwise it would just limp as before. Damaging the sensors associated with the damaged leg or the neurons themselves seemed to have little, if any, additional effect.

A.5 Conclusions

This Appendix necessarily only scratches the surface of the behaviour of the controllers. This is largely because most of the networks were too complicated to analyse at all, which is a failing of the type of controllers chosen. Generally there were over 20 and sometimes as many as 50 unique neurons in the controllers, and twice that many actual ones. This made any analysis of them a serious research task in and of itself, if not entirely impossible. Consequently this one fairly small controller was looked at in as much detail as was possible. A few other controllers were looked at briefly, but no significant differences were seen with this one, though the numbers of neurons which seemed to be active for locomotion and setting off varied slightly.

The analysis showed that the controller can be damaged in many ways without failing, and indeed it can lose over half of its sensors and neurons and still continue, but unfortunately these have to be carefully chosen! It is not robust to damage done to the other half of the neurons and sensors, which are crucial to the proper functioning of the neural network. Interestingly some neurons seem to be dedicated to starting the robot off, but are not used in normal locomotion; these were also found to be active when the sensors were damaged, perhaps indicating that their purpose at the start is to simulate the correct sensory inputs to the other neurons so that they can then set the robot moving. Damaging the actuators caused more serious problems, with no joint being dispensible: however they could all be damaged slightly — the hind limbs could survive less injury, perhaps because they are behind the centre of gravity and thus generate more of the forward force, however all joints could be damaged slightly and created a limp in that leg as one might intuitively expect.

From a technical point of view the controller did not act as a Central Pattern Generator as no walking behaviour could be seen when all of the afferent inputs had been removed, and so it behaved more like a reflex chain (see Section 2.1.3), and this is a shame from the point of view of where this research started, but it is inevitable when you consider that there was no evolutionary pressure to be able to generate rhythmical motion without the sensory feedback.

However it was robust to some damage to the neurons, sensory inputs and actuators, and perhaps the ways in which the network failed could suggest possible changes to the evolutionary regime. For instance the controller could be evolved further after something which can walk has been produced, with random damage inflicted on it before each evaluation to try to breed a more robust controller, as clearly this one had some way to go to achieve that end. At any rate some pruning of the network could certainly take place after the evolutionary run has finished to get rid of the neurons and connections which are permanently inactive. It is certainly unfortunate that controllers which were analysable were only found at the very end of the thesis, as otherwise some of these insights could have been profitably been used to improve the results further.

Appendix B

Statistical techniques

B.1 Why Use Computer Intensive Tests?

Chapters 5, 6 and 7 make extensive use of non-parametric and computationally intensive statistical tests to analyse data instead of using more traditional statistical techniques. These tests are described in detail in (Rosner, 1982; Koopman, 1987; Gunter, 1991; Cohen, 1995, for instance), but since they are a little less commonly used than the more usual parametric tests, this appendix gives a brief overview[1].

Computer intensive statistical tests can be used to derive similar quantities to standard parametric tests (such as Student's t-test), including confidence intervals and significance levels. They have several advantages over such tests however:

- Parametric tests typically assume that the distributions they are dealing with are normally distributed, or that sample sizes are sufficiently large that the central limit theorem applies. These tests can generate errors if used on skewed distributions. Computer intensive tests work with any kind of distribution — they are as powerful as parametric tests on normal distributions, and more powerful on non-normal distributions.

- Certain quantities, such as confidence intervals on the median, cannot be derived analytically, and computer intensive tests provide the only way of obtaining these quantities.

- Despite the name, computer intensive tests can be run to a sufficient degree of accuracy for moderate sample sizes in a matter of seconds on modern computers.

B.2 Resampling to calculate confidence intervals

Suppose we want to estimate some statistical parameter, such as the mean, for a particular (unknown) population. Call the true mean μ. The usual procedure is to

[1] Some of this overview comes from (Perkins, 1999)

take some sample $\{x_1, x_2, \ldots, x_N\}$ of size N and measure its mean $\overline{x}$. Our best guess is that $\mu = \overline{x}$. Of course in practice we will be slightly wrong. How can we estimate how wrong we are?

Analytical sampling theory tells us that if the population we are sampling from is normally distributed then the standard deviation of the sample mean (or 'standard error') is:

$$\sigma_{\overline{x}} = \frac{\sigma}{\sqrt{N}}$$

where σ is the true population standard deviation. In practice of course we don't know that, so we substitute it with the estimated population standard deviation obtained from the sample s:

$$\hat{\sigma}_{\overline{x}} = \frac{s}{\sqrt{N}}$$

The 95% confidence interval on the estimated value of the mean is then simply $\overline{x} \pm 1.96\hat{\sigma}_{\overline{x}}$.

To obtain the same answer in a computer intensive way we perform a resampling (or a bootstrap). The procedure is to assume that the sample we see is representative of the population it was drawn from. Therefore we can simulate drawing samples from the true population simply by drawing fresh samples of size N with replacement from our original sample. If we measure the mean of each of the new samples there will be some variability in this value. Under the assumptions of the bootstrap, this variability will be exactly the true variability of means of samples drawn from the true population, which is just what we're after.

However suppose now that we want to know the mean and standard error for the "best of n samples" taken from the original population. The standard approach would be to take a sample $\{x_{11}, x_{12}, \ldots, x_{1n}, x_{21}, \ldots, x_{Nn}\}$ of size $N * n$, measure the "best of n" for each subsample to produce $\{b_1, b_2, \ldots, b_N\}$, and then continue as before.

This would require a much larger set of samples to get a good estimate of the mean and particularly the standard error. Instead, if we are constrained as to the number of samples we can draw from the original population, we can take a sample of size $N >> n$ which we assume is representative of the population it was drawn from, and then we can draw samples of size n from the new population with replacement to produce new "best of n" values at almost no expense.

Consequently we can draw arbitrarily many of these values and calculate exactly the mean and standard error for the best of n values drawn from the original sample of size N. Assuming that the original sample is representative of the population from which it was drawn, we have calculated the mean and standard error we were looking for, but with far fewer samples from the original population than would have been necessary traditionally.

These are the calculations carried out in this thesis to estimate the mean and the 95% confidence intervals on that estimate ($\overline{x} \pm 1.96\hat{\sigma}_{\overline{x}}$).

To directly compare two different populations to see whether the "best of n" samples taken from one are better than those taken from the other, we can simply resample both together, taking the difference of the two best of ns and calculating the mean and standard error for this difference, and then we can see whether the difference differs from 0 with 95% confidence.

Appendix C

Interface

The interface for the neural and mechanical simulators was implemented as a C++ superclass called Simulant, and specified a set of methods which allowed the details of the specific simulator to be ignored. All aspects of communicating with the simulators is dealt with at this level except the loading of new structures (of robot or network), which had to be dealt with at a lower level due to the inherent differences between the programs.

```
class Simulant
{
public:
  Simulant();
  virtual ~Simulant();

// Inter-Simulant communication functions:

  // Write current state of Simulant to array, and return size.
  virtual int getState(Float *store)=0;
  // Read current state of Simulant from array.
  virtual void setState(Float *store)=0;
  // Put current outputs from Simulant into array, and return number.
  virtual int getOutputs(Float *store) { return getState(store); }
  // Get current inputs to Simulant from array.
  virtual int setInputs(Float *store)=0;

// simulant-world communication functions:

  // Define save level (eg don't save, save outputs, save state)
  inline void saveLevel(RepLevel rep_level) { _save_level = rep_level; }
  // Define save filename
  int saveTo(char *filename);
  // Define load filename
  int loadFrom(char *filename);
  // Save current state to file
  virtual void save(Float save_time)=0;
  // Load current state from file
  virtual int load(Float *load_time)=0;
  // Stop save.
  void closeSave();
  // Stop save.
  void closeLoad();

// simulation functions:

  // Initialize simulation variables (ie set up arrays to store state, etc.)
  virtual void initSimVars()=0;
  // Reload simulation variables (eg after new network installed)
  virtual void reloadSimVars()=0;
  // Test whether SimVars are initialised
  virtual int isReadyToSim()=0;
  // Set integration type (Newton Euler, Runge Kutta, etc.)
  inline void simulationType(SimType sim_type) { _simulation_type = sim_type; }
  // Run an integration step.
  virtual int simulate(Float idt)=0;

// rendering functions:

   // Initialise drawing (for gui).
   virtual void drawInit()=0;
   // Draw current state.
   virtual void draw()=0;

// fitness evaluations:

  // Set current fitness measure
  virtual void setFitnessMeasure(FitnessMeasure fm);
  // Reset fitness measure
  virtual void resetFitness();
  // Return current fitness
  virtual Float getFitness();

protected:

  // Update current fitness (called by simulate).
  virtual int updateFitness(Float idt);

  // File terminator for saving and loading (eg .NNsave for neural sim)
  char*     _file_terminator;
  // Save file stream
  ofstream* _save_stream;
  // Load file stream
  ifstream* _load_stream;
  // Is save active?
  int       _save_active;

  // Current Save and Load levels
  RepLevel _save_level, _load_level;

  // If structure varies and can be loaded on the fly these should both be
  // instantiated, otherwise both should be left blank
  virtual void saveStructure() {}
  virtual void loadStructure() {}

  // Simulation type
  SimType  _simulation_type;

  // Current fitness measure
  FitnessMeasure _fitness_measure;
  // Current fitness
  Float _fitness;
};
```

Appendix D

Example Robot

D.1 Robot Configuration File

This is the configuration file for the robot shown in Figure D.1.

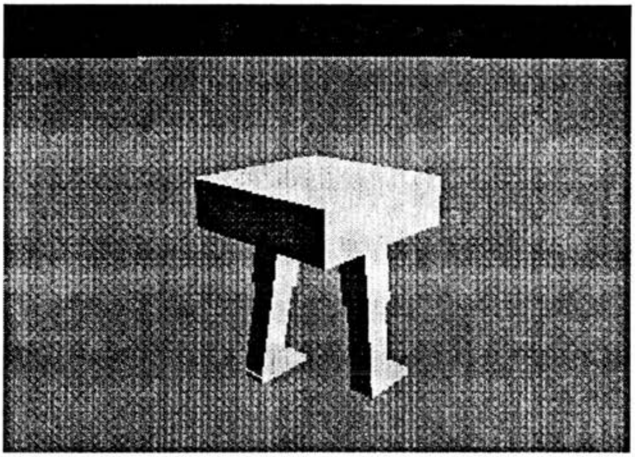

Figure D.1: An example bipedal robot

```
# A bipedal robot with feet

Graphics_Models {
  Number_Graphics_Models  4 # Number of different graphical objects used
  # filename for each graphical object, with size and offset of CoG
  "/hame/richardr/software/DynaMechs/2.0.3/models/shapes/cube_centre.zan"
    0.5 0.5 0.15 0.0 0.0 0.0
  "/hame/richardr/software/DynaMechs/2.0.3/models/shapes/cube_centre.zan"
```

```
    0.3 0.1 0.1 0.15 0.0 0.0
  "/hame/richardr/software/DynaMechs/2.0.3/models/shapes/cube_centre.zan"
    0.3 0.08 0.08 0.15 0.0 0.0
  "/hame/richardr/software/DynaMechs/2.0.3/models/shapes/cube_centre.zan"
    0.2 0.02 0.08 -0.06 0.01 0.0
}
System { # Robot
  DynamicRefMember { # Reference member - first piece of robot, part of body
                     # which articulations (legs) are attached to
    Graphics_Model_Index 0 # First graphical object

    Mass 25.0  # Mass of body (kg)
    Shape 0 # Cuboid (other options are cylinder, sphere, or arbitrary
          #          where other parameters (eg inertia matrix) are required)
    Size 0.5 0.5 0.15 # Size in metres (x,y,z)
    Center_of_Gravity 0.0 0.0 0.0 # Position of centre of gravity
                                  # relative to origin

    Number_of_Contact_Points 8 # Ground contact is only detected
                               # at specified points on the body
    Contact_Locations 0.25 0.25 -0.1  # Contact points are 8 corners of cube
                      0.25 -0.25 -0.1
                     -0.25 0.25 -0.1
                     -0.25 -0.25 -0.1
                      0.25 0.25 0.1
                      0.25 -0.25 0.1
                     -0.25 0.25 0.1
                     -0.25 -0.25 0.1

    Pcsition 10.0 10.0 0.62 # Initial position of body
    Pose 0.0 0.0 0.0 # Euler angles (phi,theta,psi)
    Velocity 0.0 0.0 0.0 0.0 0.0 0.0 # Initial velocity of body
  }

  Articulation { Leg Left 1 # First articulation off body
                            # This is a leg, on the left of the robot
                            # First set of articulations
    RevoluteLink { Body # A hinge joint connects this to the reference member
                        # This object counts as part of the body for the
                        # purposes of penalising the robot if it falls onto it
                        # Other options are Prismatic (sliding) and
                        # Ball and Socket joints.

      Graphics_Model_Index 1 # Second graphical object

      Mass 4 # 4 kg
      Shape 0
      Size 0.3 0.1 0.1
      Center_of_Gravity 0.15 0.0 0.0 # Displacement of Centre of Gravity
                                     # from joint origin

      Number_of_Contact_Points 1 # One contact point (on knee)
      Contact_Locations 0.3 0.0 0.0
    # Modified Denavit-Hartenburg coordinates are used to get from the origin
    # of the previous object (the reference member in this instance) to the
    # location of the joint (also the origin) connecting to this object.
    MDH_Parameters 0.0 -1.5707963 0.15 1.7207963
    Initial_Joint_Velocity 0 # Initial velocity of this joint (radians/s)
    Joint_Limits 0.1 3.041592653 # Joint end stops
    Joint_Limit_Spring_Constant 10000.0 # restoring spring on endstop
    Joint_Limit_Damper_Constant 1000.0 #damper on joint endstop

    Actuator_Type 2 # New direct drive motor
    Joint_Friction 10.0 # Friction in joint
    Max_Torque 80.0 # Maximum torque motor can apply
  }

  RevoluteLink { Leg # Another hinge joint connects this
                     # to the previous object. This is part of the leg
    Graphics_Model_Index 2

    Mass 2
    Shape 0
    Size 0.3 0.08 0.08
    Center_of_Gravity 0.15 0.0 0.0

    Number_of_Contact_Points 0

    MDH_Parameters 0.3 0.0 0.0 -0.05
    Initial_Joint_Velocity 0.0
    Joint_Limits -2.0 0.1
    Joint_Limit_Spring_Constant 10000.0
    Joint_Limit_Damper_Constant 1000.0

    Actuator_Type 2
    Joint_Friction 5.0
    Max_Torque 40.0
  }

  RevoluteLink { Foot # Another hinge joint, this object is a foot
                      # This means that contact with the ground
                      # is used to gate some of the sensor readings
    Graphics_Model_Index 3

    Mass 1
    Shape 0
    Size 0.2 0.02 0.08
    Center_of_Gravity -0.06 0.01 0.0

    Number_of_Contact_Points   4
    Contact_Locations 0.04 0.02 -0.04 # Four corner of base of foot
                      0.04 0.02 0.04
                      -0.16 0.02 -0.04
                      -0.16 0.02 0.04

    MDH_Parameters 0.3 0.0 0.0 -1.6707963
    Initial_Joint_Velocity 0.0
    Joint_Limits -3.1415927 -0.8
    Joint_Limit_Spring_Constant 10000.0
    Joint_Limit_Damper_Constant 1000.0

    Actuator_Type 2
    Joint_Friction 2.5
    Max_Torque 20.0
  }
} # End of Articulation

Articulation { Leg Right 1 # New articulation is a leg on the
                           # right of the body, also in the first set
                           # of articulations (ie identical to previous one)
  RevoluteLink { Body
    Graphics_Model_Index 1

    Mass 4
    Shape 0
    Size 0.3 0.1 0.1
    Center_of_Gravity 0.15 0.0 0.0

    Number_of_Contact_Points 1
    Contact_Locations 0.3 0.0 0.0

    MDH_Parameters 0.0 -1.5707963 -0.15 1.4207963
    Initial_Joint_Velocity  0
    Joint_Limits 0.1 3.041592653
    Joint_Limit_Spring_Constant 10000.0
    Joint_Limit_Damper_Constant 1000.0

    Actuator_Type 2
    Joint_Friction 10.0
    Max_Torque 80.0
  }

  RevoluteLink { Leg
    Graphics_Model_Index 2

    Mass 2
    Shape 0
    Size 0.3 0.08 0.08
    Center_of_Gravity 0.15 0.0 0.0

    Number_of_Contact_Points 0

    MDH_Parameters 0.3 0.0 0.0 -0.05
    Initial_Joint_Velocity 0.0
    Joint_Limits -2.0 0.1
    Joint_Limit_Spring_Constant 10000.0
    Joint_Limit_Damper_Constant 1000.0

    Actuator_Typo 2
    Joint_Friction 5.0
    Max_Torque 40.0
    }

    RevoluteLink { Foot
      Graphics_Model_Index 3

      Mass 1
      Shape 0
      Size 0.2 0.02 0.08
      Center_of_Gravity -0.06 0.01 0.0

      Number_of_Contact_Points 4
      Contact_Locations 0.04 0.02 -0.04
                        0.04 0.02 0.04
                       -0.16 0.02 -0.04
                       -0.16 0.02 0.04

      MDH_Parameters 0.3 0.0 0.0 -1.3707963
      Initial_Joint_Velocity 0.0
      Joint_Limits -3.1415927 -0.8
      Joint_Limit_Spring_Constant 10000.0
      Joint_Limit_Damper_Constant 1000.0

      Actuator_Type 2
      Joint_Friction 2.5
      Max_Torque 20.0
    }
  }
}
```

D.2 Communication

Most of the information in this file is used exclusively by the robot simulator, but some is passed on to the neural simulator, and the evolutionary algorithm.

The neural simulator needs to know:

- The number of sensors.
- The number of actuators.

These are extracted from the number and types of joints, and whether those joints are on legs or part of the body (on legs there is a sensor which is gated by contact of the foot on the ground). It uses this information to check that neural networks it is given are of an appropriate structure to work on the robot currently being used.

The evolutionary algorithm needs far more information:

- The number of articulations (2 in the above example).
- The number of distinct articulations (1 above).

- The order in which the articulations appear in the sensor/actuator lists.
- What side of the robot the actuator is on (for building steering controllers).
- The number of sensors and actuators.
- How the sensors and actuators match up inside the articulations.

This information comes from the first line of each new object and is purely topological in nature:

```
Articulation Leg Left 1
  RevoluteLink Body
  RevoluteLink Leg
  RevoluteLink Foot

Articulation Leg Right 1
  RevoluteLink Body
  RevoluteLink Leg
  RevoluteLink Foot
```

After the first loading of the robot, there is no further communication of structural information between the mechanical simulator and the other parts of the software, and all remaining communications are new neural structures being sent to the neural simulator, and requests for fitness evaluations for them, as well as constant exchange of inputs and outputs between the neural and mechanical simulators (see Figure D.2).

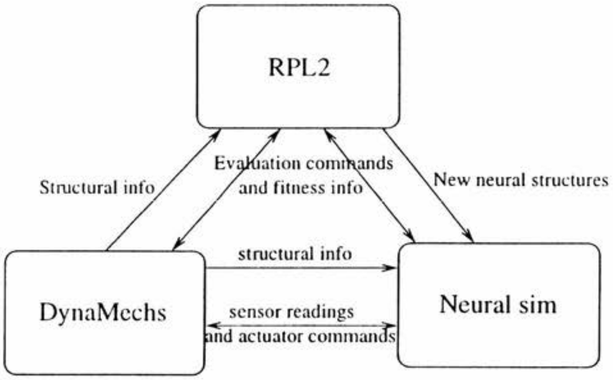

Figure D.2: Communications between different parts of the system