THE UNIVERSITY OF
WESTERN AUSTRALIA

Simulation of Biped Walking using Genetic Algorithms

2002 Honours Dissertation

Adrian Boeing

Supervisor: Assoc. Prof. Thomas Bräunl

Adrian Boeing
15 Kinkuna Way
City Beach WA 6015

November 4, 2002

The Dean
Faculty of Engineering, Computing and Mathematics
The University of Western Australia
Nedlands, WA 6009

Dear Sir,

Re: Submission of Honours Thesis

It is my great pleasure to present to you this thesis, entitled "Simulation of Biped Walking using Genetic Algorithms", in partial fulfillment for the Bachelor of Electrical and Electronic Engineering degree with Honours at The University of Western Australia.

Yours Sincerely,

Adrian Boeing

Abstract

Designing a control system for legged locomotion is a complex process. Human engineers can only produce and evaluate several configurations, although there may be numerous competing designs that should be investigated. Automating design of the control system allows the evaluation of thousands of competing designs, without requiring prior knowledge of the robot's walking mechanisms.

Development of an automated approach requires the implementation of a control system, a test platform, and an adaptive method for automated construction of the controller. Evolutionary algorithms provide a powerful method for automated problem solving. As with previous approaches, a genetic algorithm was successfully applied to the construction of locomotion controllers.

Two control systems were presented and evolved. Both control systems successfully generated locomotion controllers for bipedal robots. A detailed investigation of the spline control system was performed, and the extensibility of the controller investigated. The spline controller was applied to multiple robots with widely varying morphology and successfully demonstrated dynamic control for a number of legged robots.

Acknowledgements

First and foremost I would like to thank my supervisor, Associate Professor Thomas Bräunl for not only providing me with such an interesting project, but also for all the opportunities he has presented. A special thanks to Daniel, who has provided a great deal of advice and assistance on both the code and the thesis. Thanks to Estelle for all her kindness, help, and tolerance over the years and especially during the writing of the thesis. Steve, for always believing in me, and all the encouragement and ideas he comes up with. Jason for his programming help and friendship. Kesh for her support and affection, and my family for all the things they have done for me. Finally, I would like to thank all my other friends for their assistance and friendship during the year.

Contents

Abstract iii
Acknowledgements iv
Contents v
List of Figures viii
List of Tables ix
1. Introduction 1
1.1 Motivation 1
1.2 Objectives 2
1.3 Applications 3
1.4 Thesis Outline 4
2. Theory 5
2.1 Walking 5
2.2 Evolution 5
2.2.1 Genetic Algorithm 5
2.2.2 Fitness Functions 6
2.2.3 Selection Schemes 7
2.2.4 Operators 8
2.2.5 Encoding 9
2.2.6 Staged Evolution 9
2.3 Control Systems 10
2.3.1 Neural Networks 10
2.3.2 Splines 11
2.3.3 PID Controllers 13
2.4 Sensors 14
2.4.1 Inclinometer 14
2.4.2 Positioning 14
3. Literature Review 15
3.1 Walking and Legged Robots 15
3.2 Genetic Algorithms 15
3.2.1 Evolving Controllers 16

3.2.2 Fitness Functions 16
3.2.3 Staged Evolution 18
3.3 Control Systems 18
3.3.1 Spline Controllers 19
3.3.2 Neural Control 19
4. Design 22
4.1 Mechanical Simulator 22
4.1.1 Available Simulators 22
4.1.2 Robot Structure 24
4.1.3 Modeling 26
4.1.4 Simulation Parameters 27
4.1.5 Simulating Sensors 28
4.2 Genetic Algorithm 29
4.2.1 Operators 29
4.2.2 Selection Schemes 29
4.2.3 Fitness Functions 30
4.2.4 Randomness 31
4.2.5 Tracking 32
4.3 Control Systems 33
4.3.1 Hermite Spline Control 33
4.3.2 Spline Controller Encoding 36
4.3.3 Neural Controller 38
4.3.4 PID Control 40
4.4 Parallel Computation 40
4.4.1 Server 41
4.4.2 Clients 41
4.5 System Summary 42
5. Investigations 44
5.1 Genetic Algorithm 44
5.1.1 Method 44
5.1.2 Results and Discussion 46
5.1.3 Conclusion 48
5.2 Evolving Gaits with Splines 49
5.2.1 Spline Controller 50

5.2.2 Spline and PID Control 50
5.2.3 Spline Control with Integrated Feedback 51
5.2.4 Non Bipedal Robots 54
5.2.5 Conclusion 55
5.3 Neural Controller 55
5.3.1 Method 56
5.3.2 Results and Discussion 56
5.3.3 Conclusion 58
6. Conclusions 59
6.1 Summary of Contributions 59
6.2 Future Outlook 60
References 61
Appendix 1. Evolved Gait 67
Appendix 2. Sample Robot Model 70
Appendix 3. Software Overview 74

List of Figures

Figure 1 - The crossover operator 8
Figure 2 – The mutate operator 8
Figure 3 – Sigmoid Function 11
Figure 4 – Dynamechs Link Structure 24
Figure 5 – Modified Denavit-Hartenberg Parameters and Link coordinate Assignment 25
Figure 6 – Dynamechs object format construction utility 27
Figure 7 – Simple spline controller 33
Figure 8 – Generic extended spline controller 35
Figure 9 – Output Voltage and Spike for PTNRN 38
Figure 10 – Oscillator Output 39
Figure 11 – System Overview 42
Figure 12 – Change in required simulation time 46
Figure 13 – Average increase in fitness per generation 47
Figure 14 – Biped Robot Models 49
Figure 15 – Early Biped Gait 50
Figure 16 – Sustained Robot Walk 51
Figure 17 – Jumping Robot 51
Figure 18 – Robot walking over terrain 52
Figure 19 – Extended Spline Controller 52
Figure 20 – Fitness vs Generation for Extended spline controller 54
Figure 21 – Snake Gait 54
Figure 22 – Tripod Gait 55
Figure 23 – Gait produced from Neural Controller 57
Figure 24 – Fitness vs Generation for Neural Controller 57

List of Tables

Table 1 – Table of Dynamechs Physical Constants....28
Table 2 – Increasing chromosome complexity....37
Table 3 – Evolved Neuron Parameters and Encodings....39
Table 4 – Genetic Algorithm Parameters....45
Table 5 – Relative chromosome generation success rate....48

1. Introduction

1.1 Motivation

Legged robots exhibit a number of advantages for locomotion [1]. The mobility offered by legged vehicles is far greater than that of wheeled or tracked vehicles, as they are not limited to paved terrain. The increased mobility offered allows for a far larger range of applications to legged vehicles. Another incentive for exploring the use of legs for locomotion is that it provides an insight to the systems responsible for human and animal locomotion. Humans are capable of complex movements whilst maintaining orientation, balance and speed. Robots that could mimic human movements could seamlessly integrate with the human world, enlarging the number of possible applications for legged vehicles. This makes the study of bipedal locomotion particularly attractive.

Although there are a multitude of existing locomotion control techniques and well described design processes, the automated generation of these controllers for robots provides significant advantages. Often the design process is quite complex, time consuming to perform, and requires the control system to be completely redesigned for small alterations to the robot [2]. Furthermore, humans often have difficulty in understanding which sensors to incorporate to provide the best possible feedback to the robot. Designers are often biased towards feedback sensors that are present in humans, such as vision and touch. These senses are not necessarily the best sensors for the desired application.

Automated evolution of the locomotion controller frees the robot designer from the controller design process, removing possible human bias. The resulting control systems are more flexible, and the evolution enables the controller to utilize sensory inputs that would normally be disregarded by human designers. The resulting controllers are more adaptive to the robot's environment. They are also more robust, flexible, and usually provide superior performance to human designed controllers [2].

1.2 Objectives

There were three major objectives to this project. Firstly, to create a software system which could simulate robot locomotion. Secondly, to evolve the locomotion control system and investigate various aspects of the evolutionary and control strategies. Finally, to optimize the performance of the system with respect to the evolution time required to generate the desired gait.

The construction of the software simulation system involved:

- The construction of a control system to generate the control signals required to drive the actuators of a robot.
- A mechanical simulator for simulating the walking patterns generated from the control system.
- Implementing a set of sensors in the mechanical simulator for providing feedback to the control system.
- Creation of a set of tools to simplify robot modeling, and to provide a front end to the software to improve ease of use.

The construction of the evolutionary software involved:

- Implementing a genetic algorithm for evolving the control parameters
- Creating a system to extract information from the genetic algorithm, allowing the investigation of the performance and effects of different genetic operators and selection schemes.
- Encoding the control parameters into a format evolvable by the genetic algorithm

Additional objectives were:

- To incorporate feedback into the controllers to increase the robustness of the gait, allowing the robot to maintain its desired path despite perturbations.
- Extending the control system to allow the robot to maneuver over irregular terrain.
- Investigating various aspects of the genetic algorithm that impacted on its performance.
- Abstracting the system from robot morphology, enabling control of non-legged robots.

1.3 Applications

There are many practical application areas for legged robots. Progress to date has been limited to entertainment robots [2], exploration robots [3], and research robots [4,5,6,7]. It has been generally accepted that successful robot designs will need to be suitable to human environments [8]. Bipedal robots would have the distinctive ability to integrate with society and maneuver within the domestic environment. This would allow for bipedal robots to have uses in almost all areas, from household tasks to office and factory work.

Evolutionary locomotion systems for legged vehicles have already been employed in the development of existing entertainment robots [2] and in the special effects industry. Realistic modeling of human and animal locomotion puts a large strain on animators using traditional animation techniques. Evolving dynamically simulated gaits with genetic algorithms generates realistic and visually pleasing results [9]. Sony [2] utilized evolutionary strategies to optimize the performance of their gaits for the AIBO robot, and utilized similar systems to overcome manufacturing imperfections.

Further extensions to such systems would allow the entire control system to be evolved based on the robot's designs. This would significantly reduce the development time to construct robots, and simplify the design process.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter Two introduces the theory behind genetic algorithms, various control systems, and provides a basic introduction to walking and sensors. Chapter Three reviews background literature and presents some of the results of relevant previous research in control, robotics, computer graphics, neuroscience and evolutionary algorithms. Chapter Four provides an overview of the architecture and discusses the implementation details of the system. Investigations of the system and its results are presented in Chapter Five. An overall summary, as well as a future outlook is given in Chapter Six. The Appendix describes the software system developed for conducting the investigations, as well as example robot configuration files, and a fully illustrated gait sequence.

2. Theory

2.1 Walking

Human and animal gaits have been extensively studied for a lengthy period of time, starting in the late 18th century with researchers such as Eadweard Muybridge [10]. As early walking robots were constructed, a system for balancing the robots was required. Statically balanced robots maintain balance by ensuring the center of mass is within the supporting leg base area. As a result, statically balanced robots feature a small footstep, and slow speed. With further research into walking robots, dynamic walking was realized [5]. During dynamic walking, the center of gravity may lie outside the supporting leg base area during some periods of the walk cycle. The removal of the balance constraint allows faster walking speeds to be achieved [5].

2.2 Evolution

Evolutionary algorithms are a set of search and optimization algorithms, which make use of some of the principles of biological evolution [9]. These types of algorithms typically specify what is to be done, rather than how the task should be accomplished [11]. Although evolutionary algorithms are only a basic approximation to the biological evolutionary process in reality, they have been proven to provide a powerful means of problem solving [12,13,14,15,4,16].

2.2.1 Genetic Algorithm

A common implementation of an evolutionary algorithm is the Genetic Algorithm (GA) [11]. As with most evolutionary algorithms, the Genetic Algorithm is based on Darwin's theory of natural selection, ensuring the survival of the fittest. The genetic algorithm operates on a set of encoded variables representing the parameters for the potential solution to a problem. The parameters (or genes) are combined together to form a string of values, referred to as a chromosome [17]. Each of these possible solutions are then assigned a fitness value according to how well it solves the problem. The better solutions are then selected to "reproduce" with other solutions,

generating a new set of chromosomes which have inherited features from the chromosomes they were created from. The least fit (worst solutions) are less likely to be selected for reproduction, and thus eventually are removed from the set of chromosomes on which the algorithm operates.

The basic methodology for the genetic algorithm is presented below:

1. Randomly initialize a population of chromosomes
2. While the terminating criteria has not been satisfied
    a) Evaluate the fitness of each chromosome
    b) Remove the lower fitness individuals
    c) Generate new individuals, determined by a certain selection scheme, utilizing selected operations.

Each iteration of these steps creates a new population of chromosomes. The total set of chromosomes at one iteration of the algorithm is known as a generation. As the algorithm progresses, it searches through the solution space, refining the solutions to find one which will fulfill (or come as close as possible to fulfilling) the desired criteria, as described by the fitness function.

2.2.2 Fitness Functions

Each problem to be solved requires a unique fitness function describing the problem. Given a particular chromosome a fitness function must return a numerical value indicating the appropriateness of a solution with respect to the overall goal [18]. For some applications, such as function optimization problems, the fitness function will simply return the value of the function itself. However, for many applications there is no straightforward performance measurement of the goal. Thus, the fitness function must be expressed as a combination of the desired factors.

2.2.3 Selection Schemes

In the natural world the organisms that reproduce the most before dying will have the greatest influence on the next generation. In order to simulate this effect in the genetic algorithm a selection scheme is employed. The selection scheme determines which individuals of a given population will contribute to form the new individuals for the next generation. Tournament selection, Random selection and Roulette Wheel selection are three commonly used selection schemes.

Tournament selection operates by selecting two chromosomes from the available pool, and comparing their fitness values when they are evaluated against each other. The better of the two is then permitted to reproduce. Thus, the fitness function chosen for this scheme only needs to discriminate between the two entities. In roulette wheel selection (sometimes referred to as fitness proportionate selection [19]) the chance for a chromosome to reproduce is proportional to the fitness of the entity. Thus, if the fitness value returned for one chromosome is twice as high as the fitness value for another, then it is twice as likely to reproduce. However its reproduction is not guaranteed as in tournament selection.

Random selection randomly selects the parents of a new chromosome from the existing pool. Any returned fitness value below a set operating point is instantly removed from the population. Although it would appear that this selection mechanism would not produce beneficial results, this selection mechanism can be employed to introduce randomness into a population that has begun to converge early than desired.

Although genetic algorithms will converge to a solution if all chromosomes reproduce, it has been shown that by duplicating unchanged copies of the chromosomes future generations will generally produce a significant increase in the convergence rate towards the optimal solution.

2.2.4 Operators

The operators comprise the method by which one or more chromosomes are combined to produce a new chromosome. Traditional schemes utilize only two operators: Mutate, and Crossover [17]. Crossover takes two individuals and divides the string into two portions at a randomly selected point inside the encoded bit string. This produces two "head" segments and two "tail" segments. The two tail segments for the chromosomes are then interchanged, resulting in two new chromosomes, where the bit string preceding the selected bit position belongs to one parent, and the remaining portion belongs to the other parent. This process is illustrated in Figure 1. The mutate operator (Figure 2) randomly selects one bit in the chromosome string, and inverts the value of the bit. Historically, the crossover operator has been viewed as the more important of the two techniques for exploring the solution space, however without the mutate operator portions of the solution space may not be searched, as the initial chromosomes may not contain all possible bit values [17].

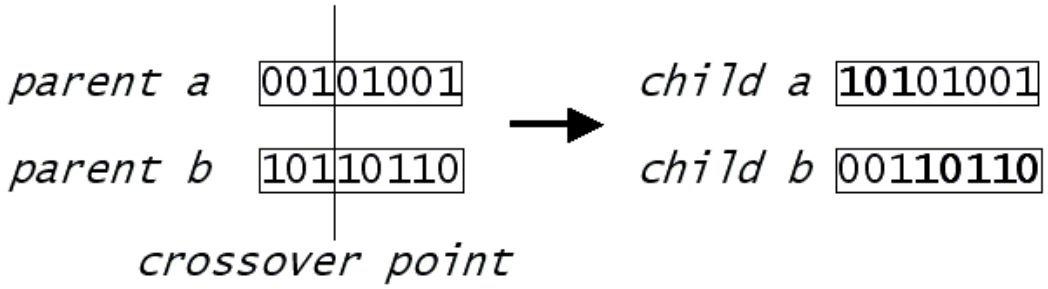

Figure 1 - The crossover operator

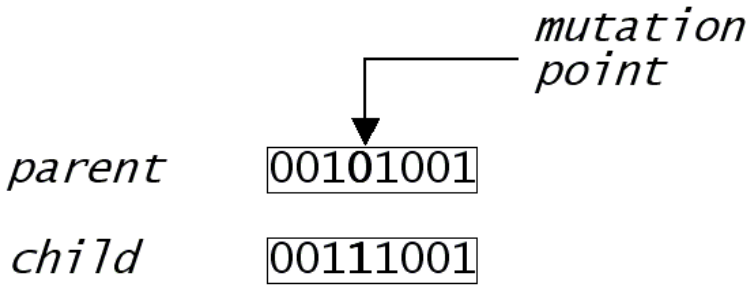

Figure 2 – The mutate operator

There are a number of available extensions to the set of traditional operators. The two point crossover operates similarly to the single point crossover described previously, except that the chromosomes are now split in two positions rather than just one [12]. The mutate operator can also be enhanced to operate on portions of the chromosome larger than just one bit, increasing the randomness that can be added to a search in one operation. Further extensions rely on modifying the bit string under the assumption that portions of the bit string represent non-binary values (such as 8 bit integer values, or 32 bit floating point values). Two commonly used operators that rely on this interpretation of the chromosome are the Non-Binary Average, and the Non-Binary Creep operators [12]. Non-Binary Average interprets the chromosome as a string of higher cardinality symbols and calculates the arithmetic average of the two chromosomes to produce the new individual. Similarly Non-Binary Creep treats the chromosomes as strings of higher cardinality symbols and increments or decrements the values of these strings by a small randomly generated amount [12].

2.2.5 Encoding

The encoding method chosen to transform the controller parameters to a chromosome can have a large effect on the performance of the genetic algorithm. A compact encoding allows the genetic algorithm to perform efficiently. There are two common encoding techniques applied to the generation of a chromosome [20]. Direct encoding explicitly specifies every parameter within the chromosome, whereas indirect encoding uses a set of rules to reconstruct the complete parameter space. Direct encoding has the advantage that it is a simple and powerful representation, however the resulting chromosome can be quite large. Indirect encoding is far more compact, but it often represents a highly restrictive set of the original structures.

2.2.6 Staged Evolution

A number of possibilities exist to enhance the performance of a genetic algorithm. Staged evolution is based on the concept of behavioral memory, and increases the convergence rate by introducing a staged set of manageable challenges [21]. Initially limiting the search to a subset of the full solution space enables an approximate solution to be determined. Incrementally increasing the complexity of

the problem will increase the solution space, providing the possibility of increased performance as further refinements of the solution are possible. Applying this strategy to a particular problem task requires that the problem is capable of being divided into smaller sub-tasks that can be sequentially solved.

2.3 Control Systems

There are a number of control systems that are applicable to robot locomotion. Possible control systems range from simple oscillators [15] to neural networks [2] to simple assembly programs [22]. The simplest oscillators consist of a set of sinusoidal function generators whose outputs are combined to generate the control signal for an actuator. These systems can represent a range of outputs by altering the phase and amplitude of the sinusoids [15]. Such systems are generally incapable of expressing the complexity required for sustained locomotion [9]. Thus, more sophisticated forms of control are desirable.

2.3.1 Neural Networks

Neural Networks are a popular form of controller for robot locomotion [13,14,23,24,4,21]. These controllers complement the biological systems believed to be responsible for walking movement in humans and other animals. This form of control allows the application of the knowledge gained by neuroscientists studying rhythmical controllers in animals [23].

Neural Networks consist of a set of interconnected processing element nodes, whose functionality is based on the animal neuron [25]. Neurons process information by summing the signals that appear at the node's inputs. Each of the input signals is multiplied by a weight to simulate differing input strengths. The weighted sum is passed through an activation function, which will produce an output if the transformed sum passes a calculated threshold level [25]. Traditionally, artificial neurons have been idealized for the sake of mathematical tractability [14]. One of the simplest, and most commonly implemented neuron models, is the sigmoidal neuron [13]. This neuron is a simple extension of the binary neuron model to express a

continuous softened step-function. The equations governing the output of the sigmoidal neuron are given below.

$$y_i = \Sigma_{j=1}^{n} w_{ji} S_j \tag{2.1}$$

$$S_i = \frac{1}{1 + e^{-y_i}} \tag{2.2}$$

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

y_i is the internal state of neuron i

Equation 1- Sigmoid Neural Model

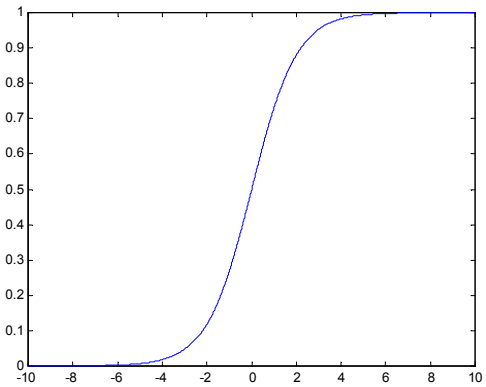

Figure 3 – Sigmoid Function

There are many extensions possible to this simple model, and many have been applied to the area of robotic locomotion.

2.3.2 Splines

Splines are piecewise polynomial functions expressed by a set of control points [26]. There are many different forms of splines, each with their own attributes. There are two desirable properties for the splines to posses.

- Continuity, so that the generated control signal translates to smooth velocity and acceleration changes.
- Locality of the control points, to reduce the influence of alterations of one control point to the overall shape of the spline.

Two of the most commonly used splines in computer graphics [26] have also been applied to robotic control [27]. The B-spline is defined by Equation 2. The B-spline features two important properties, locality and continuity. Each segment of the B-spline curve is dependent on only a limited number of the neighbouring control points. Thus a change in the position of a distant control point will not alter the shape of the entire spline [26,27]. The continuity of the spline is determined by the order of the polynomial functions utilized. A B-spline of order K is also generally C^{K-2} continuous.

A B-spline function with four control points $s_0,\ldots,s_3$ parameterized by t=0,…,1, is expressed in Equation 2.

$$f(t) = s_0 b_0(t) + s_1 b_1(t) + s_2 b_2(t) + s_3 b_3(t) \tag{2.3}$$

where

$$\begin{aligned} b_0(t) &= -\frac{t^3}{6} + \frac{t^2}{2} - \frac{t}{2} + \frac{1}{6} \\ b_1(t) &= \frac{t^3}{2} - t^2 + \frac{2}{3} \\ b_2(t) &= -\frac{t^3}{6} + \frac{t^2}{2} + \frac{t}{2} + \frac{1}{6} \\ b_3(t) &= -\frac{t^3}{6} \end{aligned} \tag{2.4}$$

Equation 2 – The B-Spline

The Hermite spline is expressed by Equation 3. Unlike the B-spline, the curve generated from the spline passes through the control points that define the spline. Thus, a set of predetermined points can be smoothly interpolated by simply setting the predetermined points as the control points for the Hermite spline. Like the B-spline, the curve generated from the Hermite spline is dependent only on the neighboring control points. The Hermite spline can also be constrained such as to achieve C^{K-2} continuity. However, the disadvantage of the Hermite spline is that the control point tangent values must be explicitly specified.

The function used to interpolate the control points, given starting point p_1, ending point p_2, tangent values t_1 and t_2, and interpolation parameter s, is shown below:

$$f(s) = h_1 p_1 + h_2 p_2 + h_3 t_1 + h_4 t_2 \tag{2.5}$$

where

$$\begin{aligned} h_1 &= 2s^3 - 3s^2 + 1 \\ h_2 &= -2s^3 + 3s^2 \\ h_3 &= s^3 - 2s^2 + s \\ h_4 &= s^3 - s^2 \\ 0 &\le s \le 1 \end{aligned} \tag{2.6}$$

Equation 3 – Hermite Spline

2.3.3 PID Controllers

Proportional-Integral-Differential (PID) controllers provide robust performance under a wide range of operating conditions [28]. Only three parameters need to be determined to construct a PID controller, the proportional gain, integral gain and derivative gain [29]. A proportional controller reduces the time required for a response, however the response value will never completely settle. Integral control will eliminate these oscillations at the expense of increased response time. Derivative control increases the systems stability, thus reducing the overshoot, and may improve the transient response. The transfer function for a PID controller is given in Equation 4.

$$g(s) = K1 + \frac{K2}{s} + K3 \cdot s \tag{2.7}$$

Equation 4 – PID Controller Transfer Function

2.4 Sensors

To enable feedback within a control system, some form of sensory input is required to provide the desired information. There are a number of sensors that provide various types of information about the current state of a system.

2.4.1 Inclinometer

Inclinometers return an object's angle of inclination about a specified axis. These sensors are used to monitor a robot's movement to obtain information about the slope of the terrain, or to provide corrective feedback. Inclinometers contain a conductive liquid that changes the resistance between two electrodes when tilted. The resistance value returned from the inclinometer is proportional to the orientation angle.

2.4.2 Positioning

Distance sensors determine the distance between an object and the sensor. There are numerous forms of distance sensors, such as vision based methods, sonar, radar, and infrared [30]. Most distance sensors operate by emitting a signal (such as an ultrasonic or infrared pulse) and detect the time or angle of the signal reflected from the object. Combining a set of distance sensors with compass sensors allows the robot's position and orientation to be accurately determined, provided the robot is within a controlled environment.

3. Literature Review

3.1 Walking and Legged Robots

Interest in the study of walking dates back millennia, but scientific research first began in the late 18th century. The two most significant early studies of gaits were by Marey and Muybridge [14]. Marey [31] recorded stepping patterns from the feet of subjects through the use of pneumatic recording devices. Muybridge [5] investigated animal gaits through the use of high-speed photography. Previous to this, attempts had been made to mimic human life with complicated systems of levers and bellows [14].

The earliest walking machines used gears to produce fixed patterns of leg motion for walking. In 1968 General Electric developed a more agile walking truck capable of climbing over large obstacles [32]. This walking truck however still relied on a human as the controller. The earliest self-contained computer controlled walking robot was a hexapod developed by Sutherland and Raibert in 1986 [33]. By this time a number of control algorithms for legged locomotion had been proposed.

3.2 Genetic Algorithms

Genetic algorithms were first implemented on a computer by John Holland [11]. Investigations into further extensions of the basic algorithm were examined by several researchers [12]. DeJong [34] investigated the multi-point crossover operator, and concluded that the two-point crossover provided benefits, however any higher order crossover provided little improvement. Bramlette [35] proposed interpreting chromosome symbols as integer values. Davis [36] indicated that many problem parameters are often numeric, and thus interpreting the chromosome in its original numeric format can be advantages. This allows the definition of more meaningful crossover and mutate like operators. As a result, the non-binary average operator emulates the crossover operator, and the non-binary creep imitates the mutate

operator. Koza [37] applied evolutionary techniques to construct lisp programs, and outlined the fundamentals of the genetic programming approach.

3.2.1 Evolving Controllers

Genetic algorithms were applied to the evolution of neural controllers for robot locomotion by numerous researchers [13,14,15,4,16]. Lewis et al. [4] successfully generated locomotion patterns for a hexapod robot using a simple traditional genetic algorithm with one point crossover and mutate. Isjpreet [16] evolved a controller for a simulated salamander using an enhanced genetic algorithm. The large number of experiments in this area clearly indicates that genetic algorithms are capable of evolving neural controllers that describe legged robot locomotion [13].

The genetic programming approach was shown to successfully generate locomotion patterns for various control strategies. Banzhaf et al. [38] demonstrated the pure genetic programming approach to develop assembly programs for robot control. The system was then expanded to control a hexapod robot [27] using a B-Spline based approach. Lewis also applied genetic programming to his neural controller [21]. This demonstrated that both the genetic programming approach and the genetic algorithm approach should both be capable of evolving adequate control systems for legged locomotion [21,16].

Parker et al. [6] explored the use of cyclic genetic algorithms for locomotion control of a small hexapod robot. The system demonstrated that the cyclic nature needed to generate the oscillatory motions necessary for legged robot locomotion could be abstracted from the control system and transferred and encoded into the genetic algorithms chromosomes.

3.2.2 Fitness Functions

Ijspeert and Kodajabachian developed a set of fitness functions for evolving swimming gaits for a simulated lamprey [39]. Their function was aimed at developing swimming gaits that resulted in high swimming speeds, and was able to smoothly

alter its swimming speed and change directions at any time. The fitness function Ijspeert et al utilized is given in Equation 5.

$$fitness = fit\_contorsion \cdot fit\_max\_speed \cdot fit\_speed\_ramge \cdot fit\_turning \in [(0.05)^4, 1.0] \quad (3.1)$$

Equation 5 – Ijspeert's Fitness Function

This fitness function rewards four factors: contorsion (around the center of the lamprey from head to tail), the maximum speed achievable, the range of speeds capable, and the ability to alter the lampreys direction.

Reeve [14] experimented with various legged robots and investigated a number of fitness functions. Although Reeve reports that the Speed5 fitness function (average speed of the walker over five seconds) performs adequately, improvement on the performance of the algorithm can be achieved through the use of more complex functions. Reeve proposed five different extended fitness functions:

- FND – (forward not down) The average speed the walker achieves minus the average distance of the center of gravity below the starting height.
- DFND – (decay FND) Similar to the FND function, except it uses an exponential decay of the fitness over the simulation period.
- DFNDF – (DFND or fall) As above, except a penalty is added for any walker whose body touches the ground.
- DFNDFA – (DFNDF active) This function incorporates features of the actual control system into its evaluation of the gait. The function evaluates the individual neurons and ensures they are active, and are not stuck at an on or off value.
- DFNDFO – (DFNDFA oscillator) As above, with the added constraints that both the neurons and legs oscillate.

Ziegler et al [27] utilized fitness functions which compared the generated trajectory of a gait to the desired path. The trajectory was specified to include an initial acceleration, then a straight walk along the desired path, and a deceleration. The square difference of the actual walk from the desired was then summed over the duration of the gait and returned as the fitness value. In order to optimize the performance of the evolution algorithm, Ziegler et al introduced premature

termination conditions to the fitness function. The premature termination condition ensured that the initial trajectory was within a valid range of the desired trajectory. Thus, if the desired trajectory were forwards movements, then any gait that produced backwards movement would be terminated.

3.2.3 Staged Evolution

De Garis first introduced the concept of behavioral memory to robotics [40]. De Garis proposed that the evolution of a controller is dependent on the starting conditions of the search, thus preliminary controllers could be evolved using less constrained fitness functions. Lewis et al. extended this concept and introduced staged evolution theory. Staged evolution implies that sub-controllers can be evolved which can then be combined to create the overall controller. Lewis et al demonstrated the successful application of this technique to the evolution of walking patterns for a six-legged insect like robot using a neural controller. The evolution was performed over two phases. Initially, individual oscillators were evolved. These then combined to create the overall neural network that controlled the robot's walking patterns. During the second phase of evolution, the weightings between individual oscillators within the network were evolved to produce an optimized controller.

3.3 Control Systems

A large number of control systems have been applied to legged robot locomotion. Simple sinusoidal oscillatory systems have been commonly used in computer graphics applications, due to their ease of evolution [15]. However Arnold indicates that these systems are insufficient for describing satisfactory legged motion in realistic dynamic simulations, or on real robot hardware [9]. Arnold investigated spectral synthesis techniques [9], and generated gaits for a range of virtual creatures. The system provided satisfactory performance in the computer graphics domain and generated lifelike movements. Incorporating sensory information into the spectral system proved to be difficult, and thus other methods for robot control need to be considered for practical application.

3.3.1 Spline Controllers

Machine code control architectures for robot control were explored by Banzhaf et al [38]. However, when this architecture was applied to robot locomotion the evolution periods required to generate the controller were too lengthy for the system to be of practical use. Furthermore, the control signals generated began to approximate spline behavior. A B-Spline approach was then explored and successfully produced a controller that described bipedal locomotion for both simulated and real robots. Zhang et al. [41] have been investigating fuzzy control systems for numerous control applications, and have determined that B-Spline hyper-surfaces can adequately describe many control applications. This indicates that spline based control systems can be seamlessly integrated with other control methods, and may also provide significant performance gains from doing so. Banzhaf et al. [38] indicates that this form of control is relatively unexplored for locomotion, and future work needs to be completed in order to determine the efficiency of the spline based control architecture.

3.3.2 Neural Control

Neural networks have traditionally been the controller of choice for automated gait generation. The investigation of the biological mechanisms behind animal locomotion has been the driving force behind advances in neural control in legged robot locomotion. Although Sigmoid neurons are capable of producing oscillatory output, the number of neurons needed to achieve this negates any of the benefits obtained by using such a simple model [13]. MacGregor proposed a neuron model based on a simplified version of Hodgkin and Huxley's equations [42]. The model was developed to provide repetitive neuron firing based from three first order differential equations.

$$\frac{dV_m}{dt} = \frac{-V_m + (Current + GK \cdot (EK - V_m))}{T_{mem}} \tag{3.2}$$

$$\frac{dTh}{dt} = \frac{-(Th - Th_0) + ampThreshold \cdot V_m}{T_{thresh}} \tag{3.3}$$

$$Spike = \begin{cases} 0 \ \ if\, E < Th \\ 1 \ \ if\, E \ \geq \ Th \end{cases} \tag{3.4}$$

$$\frac{dGK}{dt} = \frac{-GK + ampGK \cdot Spike}{T_{GK}} \tag{3.5}$$

where: *E,Th,GK* and *Spike* are the output state variables,

Current is the input function, and

EK, Th_0, Tmem, ampGK, and *ampThreshold* are constants

Equation 6 – MacGregor's Repetitive Firing Model

Beer [24] devised continues time recurrent neural networks to overcome the difficulties in generating the oscillatory output necessary for legged locomotion. The equations governing Beer's neuron model is described below (refer to Equation 1 for previous parameter definitions).

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{n} w_{ji} S_j \tag{3.6}$$

$$S_i = \frac{1}{1 + e^{(\theta_i - y_i)}} \tag{3.7}$$

where: θ_i is a bias term, and

τ_i is the adaptation rate of the neuron

Equation 7 – Beer's Neuron Model

Although this neuron model is only a first order differential equation, it was still shown to produce complex oscillatory controΦl signals when interconnected in a network [14]. Beer demonstrated that the model was capable of producing control signals required for a statically stable walking robot. Taga [43] extended this model to two coupled first order differential equations. The extended model proved to be capable of generating a larger variety of control signals and oscillators. Taga's model was used to control a dynamically stable bipedal robot. Wallen et al [23] extended this concept further to produce a third order model given in Equation 8.

$$\tau_i^{\,D}\frac{d\xi_i^{+}}{dt} = -\xi_i^{+} + \sum_{j\in\psi_+} w_{ji}S_j \tag{3.8}$$

$$\tau_i^{\,D}\frac{d\xi_i^{-}}{dt} = -\xi_i^{-} + \sum_{j\in\psi_-} w_{ji}S_j \tag{3.9}$$

$$\tau_i^{\,\wedge}\frac{d\theta_i}{dt} = S_i - \theta_i \tag{3.10}$$

$$S_i = \max(0, 1 - e^{(\theta_i - \xi_i^{+})\Gamma_i} - \xi_i^{-} - \mu_i\theta_i \tag{3.11}$$

where: ξ_i^{+} and θ_i are the internal state of neuron *i*, and

$\psi_{\pm}$ is the set of all excitory (or inhibitory) inputs, and

Γ_i and μ_i are bias terms

Equation 8 – Wallen's Neuron Model

This model was deduced from the analysis of the neurons found in the lamprey spinal cord, and was successfully used to simulate the neural control of swimming lampreys. Krueger [13] successfully utilized his neural model to generate walking gaits for simulated quadrupeds.

Some of the problems encountered during the evolution of neural models was the difficulty in evolving acceptable oscillators, and the methods in which to combine the oscillators to produce the desired control signals. In some cases the neural oscillators needed to be designed by hand [4], and only the combined network of oscillators could be evolved. Currently, the optimal method for organizing the controlling neural network is unknown, and thus a fully interconnected network has been used [13]. This significantly increases the complexity of the system that needs to be evolved, and leads to undesirably long evolution periods.

4. Design

As with previous approaches [27,14,13,9], the proposed system consists of three major sub systems:

- A genetic algorithm to evolve the parameters for the control system.
- The control system itself, to generate the signals required to produce a walking pattern
- A mechanical simulator for executing the generated control signals and evaluating the resulting gait.

4.1 Mechanical Simulator

The physical simulator is one of the most crucial components of the overall system, as the practical usefulness of the generated gait is limited by the capabilities of the simulator. Although a number of control systems can only describe a limited subset of the required control signals for legged locomotion [15], the control system can always be expanded [9], or altered to a form which is capable of more complex control [13]. Thus, the limiting factor on testing the gait is the accuracy and capabilities of the simulator itself.

4.1.1 Available Simulators

There were four simulators under consideration for use during the course of the project.

MathsEngine[44] produces a commercial dynamic simulator design for use in computer games. It boasts the best performance of all publicly available simulators [45]. The system has been previously used to generate gaits for simulated robots [46], but has never been used in a capacity that indicates its practicality when transferred to a real robot. The simulator only has first order accuracy making it inappropriate for practical robotics use.

Darwin2k [47] is a simulator produced by Chris Ledgers PhD research [19]. The simulator was designed with realistic simulations in mind. The system has since been used for some robot design, however there have been no examples of control programs developed on the simulator which have been immediately transferable to an identical hardware system. To date, the software is still unstable and largely incomplete.

AREO [48] is a free simulator designed for simulations for computer graphics applications. It is a relatively fast simulator with reasonable accuracy [45]. Banzhaf et al [27] used the simulator as a base for the evolution of a controller for both simulated and real legged robots. The simulator was considerably modified to enable its use in this form [27].

Dynamechs [49] is the simulator that is the result of McMillans PhD work [50]. The system was developed for simulating an underwater robot constructed by the US Naval Postgraduate School [49]. The simulator provides a number of configurable integration methods enabling highly accurate and fast dynamic simulations [45]. The simulator however does not provide full collision simulation.

The desired characteristics for the simulator were:

- Proven, tested, realistic simulations
- Accurate and fast simulation
- Portable (between both Linux and Windows)
- Free, and Open Source
- Ability to be easily extended to simulate any sensors, actuators or joints required.

The only proven system that was portable across both the Linux and Microsoft Windows platforms, and was open source was Dynamechs. Hence the Dynamechs simulation library was used in this project, despite its limited collision simulation capabilities.

4.1.2 Robot Structure

The robot structure is described within the Dynamechs v4.0 format [51]. A Dynamechs robot is expressed in terms of its links, with separate model files to describe the morphology of the robot. The link format is depicted in Figure 4.

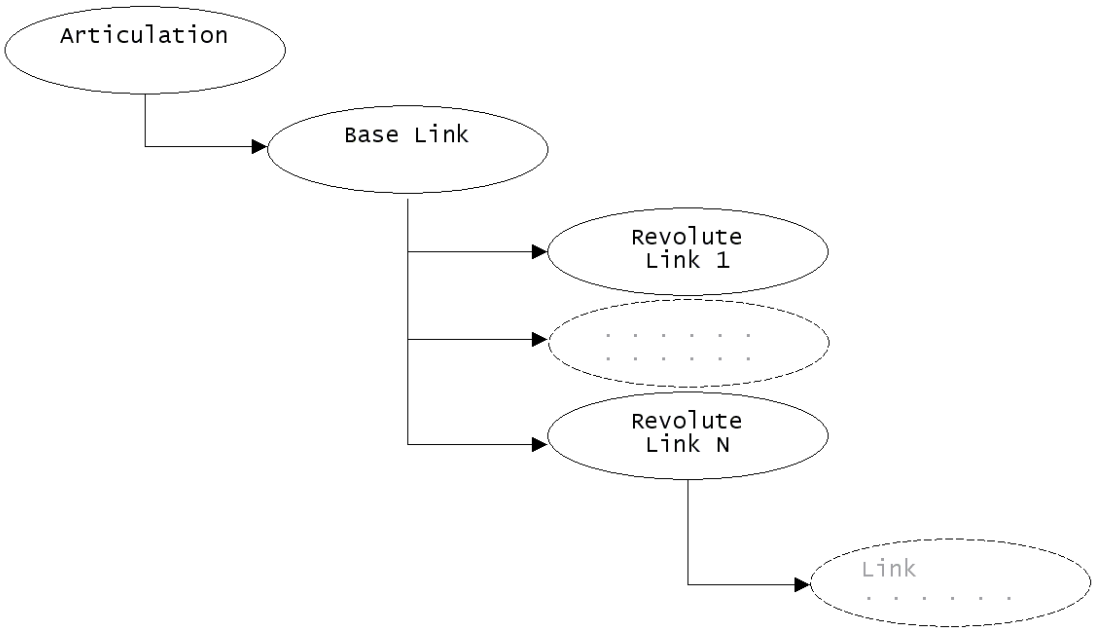

Figure 4 – Dynamechs Link Structure

The base object for every robot is the "articulation", which contains the starting position of the robot specified in Cartesian coordinates, and the orientation specified as a quaternion. A reference member, indicating the "body" of a robot, must then directly follow the base articulation. The base articulation and all other link objects contain the following information structures:

- Name, A string which describes the object
- Graphics Model, Points to the file containing morphological information
- Inertia, This is a 3x3 inertia matrix. The inertia matrix needs to be calculated from the mass and dimensions of the object.
- Center_of_Gravity, A 3d Cartesian point indicating the center of gravity

- Contact_Locations, This is a list of 3d points where the object can make contact with the terrain. This is usually simply a list of vertex positions from the graphics model.

In addition to these data structures, the base object holds information on the position of the base (a Cartesian point), its orientation (as a quaternion) and its velocity (as a Cartesian vector).

Other links that follow the base link contain additional information describing the location of the joint in modified Denavit-Hartenberg format [50]. The Denavit-Hartenberg notation consists of four parameters:

- a_i, the distance from the preceding link to the current link with respect to the previous links 'x' axis.
- α_i, the angle which the link has been rotated in the 'x' plane
- d_i, the perpendicular distance from the preceding link to the current link. i.e. the 'z' (up) axis
- θ_i, the angle which the link has been rotated in the 'z' plane.

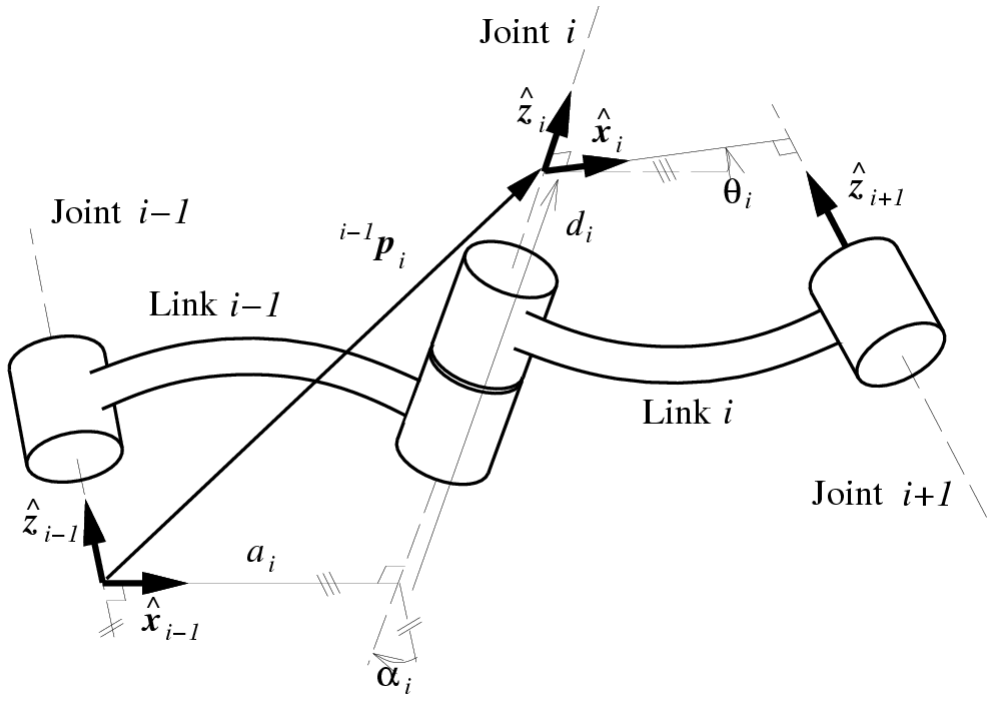

Figure 5 – Modified Denavit-Hartenberg Parameters and Link coordinate Assignment [50]

4.1.3 Modeling

Dynamechs did not have a built in modeling utility, thus an external 3d design package needed to be either created or modified to support the Dynamechs file format. The G-MAX [52] modeling environment was initially under consideration for extension as it was a widely used professional modeling package. However, the development time required to create the software to integrate with the package was expected to exceed the release date for a purpose built software package [53]. RobotBuilder was a robot modeling package which could directly export to the Dynamechs file format. Unfortunately the package did not automate the required calculations for modeling individual model parts.

A separate utility was created to simplify the generation of various basic shapes. The utility allowed users to view a rotated version of the object, and would calculate the orientation quaternion. The other major function the utility performed was the calculation of the inertia matrix for the objects. The equations used to calculate the moments of inertia for a rectangular prism are given in Equation 9.

$$\begin{aligned} Ix &= \frac{m}{12}(a^2 + b^2) \\ Iy &= \frac{m}{12}(a^2 + c^2) \\ Iz &= \frac{m}{12}(b^2 + c^2) \\ m &= \frac{a \cdot b \cdot c \cdot rho}{g} \end{aligned} \tag{4.1}$$

where: *a,b,c* are the objects width, height, and depth

m is the mass,

g is the acceleration of gravity,

rho is the density, and

I is the mass moment of inertia.

Equation 9 – Inertia for Rectangular Prism

Figure 6 illustrates the interface presented to the user for constructing various objects. The user can specify the dimensions and color of the object, and the center of reference for the object. The utility calculates the inertia matrix values for the user and can export the data to a format directly readable by RobotBuilder and Dynamechs.

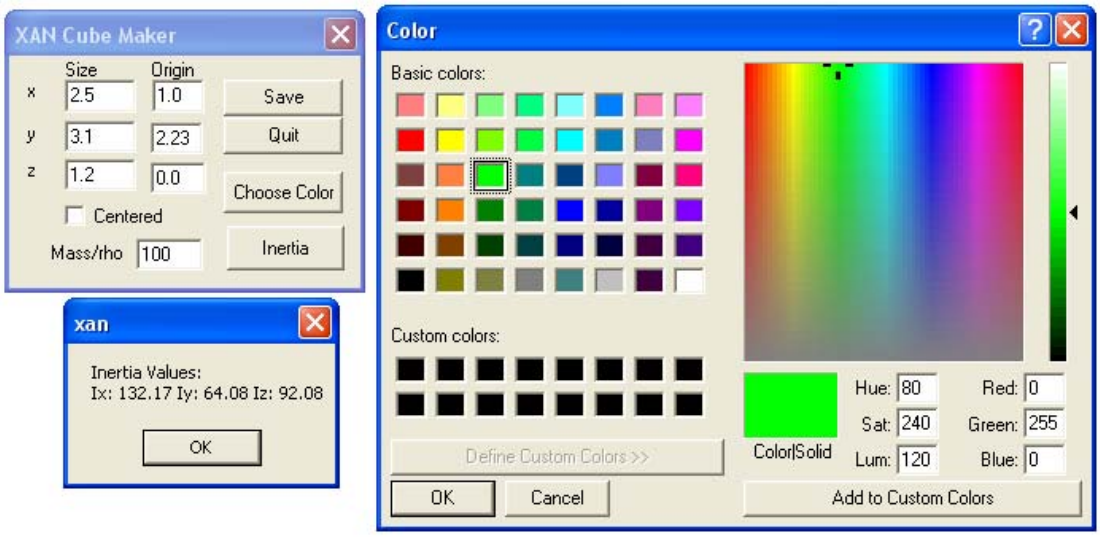

Figure 6 – Dynamechs object format construction utility

4.1.4 Simulation Parameters

The simulation requires a number of parameters to be defined. These include the integration step size, physical constants, and terrain information. An appropriate integration step size was previously determined by Kruger [13]. For a percentage error of less than 5% a step size of 30ms was required. Subsequently, this value was also used in the simulations for this project.

The physical constants were taken from the underwater quadruped project undertaken by the US Military Postgraduate School [49].These values were chosen as this was the only available information from a system which successfully realistically modeled a robot. The values are listed in Table 1.

Constant	Value
Gravity	-9.81
Ground Planar Spring Constant	55000.0
Ground Normal Spring Constant	75000.0
Ground Planar Damper Constant	500.0
Ground Normal Damper Constant	500.0
Ground Static Friction Coefficient	2.840
Ground Kinetic Friction Coefficient	2.010

Table 1 – Table of Dynamechs Physical Constants

4.1.5 Simulating Sensors

There were a number of sensors that needed to be simulated in the mechanical simulator. The sensors that were required during the course of this project were:

- Inclinometers
- Position Sensors
- Orientation Sensors

The inclinometers were simulated by extracting the orientation quaternion from the Dynamechs object, and transforming it to a rotation matrix. The rotation matrix was then converted to a Euler angle format to provide angular information on the orientation of the object. Distance sensors were simulated by recording the starting locations of the objects in the scene, and then extracting the Cartesian coordinates for the location of the objects at any time in the simulator. The orientation of a robot could be extracted using a method similar to the conversion process used to simulate the Euler angles. None of the simulated sensors included any form of white noise or drift, and thus were completely idealized sensors.

4.2 Genetic Algorithm

The implemented genetic algorithm was required to evolve any control system for robot locomotion, and provide the means to do so in the most efficient manner possible. In order for the system to operate on any control system, an encoding method for the controller needed to be found such that the genetic algorithm would always be able to evolve the controller parameters. Consequently, the genetic algorithm could not operate using specific knowledge of the controller's parameter types. To achieve efficient evolution, a number of operators and selection schemes were implemented to allow the configuration of the genetic algorithms parameters to achieve a reduction in the evolution period.

4.2.1 Operators

To allow for a comparison against the traditional genetic algorithm form [17], the crossover operator was implemented as a single point crossover, and the mutate operation was implemented as a single random bit inversion. Davis [36] indicated that performance enhancements can be obtained by treating the chromosome as a string of numeric values. Hence the extended operators implemented were the non-binary average, creep, and mutate. Each of the non-binary operators treated the symbol alphabet as 8-bit unsigned integer values. Thus, the non-binary average operator simply arithmetically averaged corresponding 8-bit integers from two chromosomes, and generated a new chromosome based on the resulting values. The creep operator selected one 8-bit value randomly within the chromosome and randomly added or subtracted a randomly generated value ranging between one and two. The non-binary mutate operator, simply replaced an entire 8-bit value within the chromosome with a randomly generated 8-bit value.

4.2.2 Selection Schemes

In order to provide a flexible genetic algorithm implementation, a number of selection schemes were implemented. Compatibility with the traditional genetic

algorithm [17] was provided through Roulette Wheel selection. For research purposes two more selection schemes, Random and Elitist were implemented. Random selection selected the parent chromosomes at random, with no respect to the fitness value of the chromosome. However, negatively fit chromosomes (that is, robots which fell over or otherwise terminated) were still removed from the pool. Elitist selection performed weighted Roulette Wheel selection on the top ten percent of chromosomes. Thus highly fit chromosomes were far more likely to contribute to future generations than lower fitness valued chromosomes, which could not contribute at all.

4.2.3 Fitness Functions

The fitness functions implemented was based on the combined findings of Ijspeert [39], Reeve [13], and Ziegler et al [27]. Both Ijspeert and Reeve found increased performance when controller and morphology specific information was added to the fitness functions. Reeve found that by evaluating individual neurons performance the fitness function resulted in increasing the performance of the GA. Ijspeert found that by incorporating the desired movement behaviors of the lampreys structure the fitness of each gait could be better evaluated. Ziegler et al. determined that premature termination largely increased the GA performance. As a result, a number of fitness functions were implemented for differing purposes.

The basic fitness function implemented followed both the principles implemented by Reeve and Ziegler et al. During the initial phases of evolution the fitness was evaluated purely from the distance the robot traveled forward, minus the distance the robots center of gravity lowered. This is somewhat of a combination of Reeve's FND and Zieglers premature termination conditions [13,27]. During later phases of evolution the average speed at which the robot moved, and the distance the robot wavered from the desired path were incorporated. Finally, the distance the robot was at termination from its desired terminating point was taken into consideration, to emulate the effect of Reeve's DFND [13].

A number of early termination conditions were added to the fitness evaluation. These included:

- An automatic termination for a robot's central body coming too close to the ground. This termination condition was implemented rather than the DFNDF method proposed by Reeve, as it allowed the simulation of robots whose central body may touch the ground during locomotion (eg: snake-like robots).
- Automatic termination for a robot who moved too far from the ground. This removed the possibility of robots achieving high fitness values early in the simulation by propelling themselves forwards through the air (jumping).
- For bipedal android robots, an additional termination condition was provided, which would degrade the fitness of a gait that resulted in the robot's head tilting too far forwards. This ensured the robots were somewhat stable and robust.

The choice of the fitness functions and terminating conditions was designed to allow any robot morphology to be evaluated. The actual fitness function parameters are passed to the simulator at run time, allowing the user to specify their own fitness function.

4.2.4 Randomness

The pseudo-randomness available using the standard C-library rand() function (ANSIC LCG [54]) was insufficient for generating the long sequences of random values that were required for larger chromosomes. Entacher [54] performed a large number of spectral tests on the randomness available from various Linear Congruential Generators (LCG). The results suggest that the LCG developed by Karin and Goyal [55] provides far better pseudo-random sequences than the ANSIC LCG.

The LCG implemented is given below:

$$y_{n+1} = a \cdot y_n + b \bmod m \text{ with seed } y_0 \tag{4.2}$$

where $m = 10^{12}-11$

$a = 427419669087$

$b = 0$

$y_0 = 1$

Equation 10 – The Maple LCG

4.2.5 Tracking

To be able to determine the performance impact of differing operators, selection schemes and fitness functions on the genetic algorithm, chromosome tracking information was added to the implementation. The collected information included: the fitness of the chromosomes that were combined, the fitness of the resulting chromosome, the operator used to produce the new chromosome, and the parent chromosomes. The resulting output allowed the construction of a family tree for each individual chromosome. This assisted in determining which operators contributed to the optimal solution, and the impact of each operator.

Further records of the complete genetic algorithm were also maintained. The recorded information included population size, average and best fitness values, the current generation, the number of new individuals created, and the time required to evaluate the generation.

4.3 Control Systems

The only fully implemented control system for robot locomotion was the spline based controller. Some basic neural oscillators and simple neural network controllers were implemented, however the functionality provided by the neural controller was severely limited. A proportional controller was also implemented for simple actuator control, although it was not applied to generate the robot locomotion control signals directly.

4.3.1 Hermite Spline Control

The spline controller implemented consists of a set of joined Hermite splines. One set contains robot initialization information, to move the joints into the correct positions and enable a smooth transition from the robot's standing state to a travelling state. The second set of splines contains the cyclic information for the robot's gait. Each spline can be defined by a variable number of control points, with variable degrees of freedom. A pair of start and cyclic splines corresponds to the set of control signals required to drive one actuator within the robot.

An example of a simple spline controller is illustrated in Figure 7. Each spline indicates the controller's output value for one actuator.

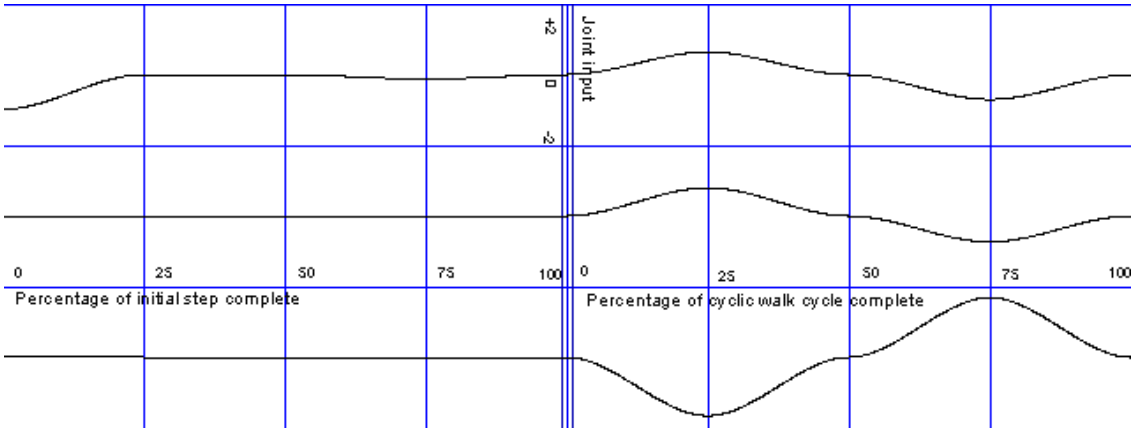

Figure 7 – Simple spline controller

The number of control points required for the simple spline controller is given by Equation 11.

$$a \cdot (i + c) \tag{4.3}$$

where a is the number of actuators

i is the number of control points in the initialization spline

c is the number of control points in the cyclic spline

Equation 11 – Control points required for simple spline controller

Cubic Hermite splines were implemented in the controller as they offer a number of desirable characteristics over other splines. The major advantage of the Hermite spline is that the curve passes through all the control points. As the position and tangent are explicitly specified at the ends of each segment, it is easy to create a continuous curve. This allows for a control representation that is far simpler for humans to manipulate using a keyframing-styled approach than B-Splines, whilst still possessing the desired properties to allow fast and flexible evolution.

There is a large collection of evidence that supports the proposition that most gaits for both animals and robots feature synchronized movement [56]. That is, when one joint alters its direction or speed, this change is likely to be reflected in another limb. Enforcing this form of constraint is computationally simpler with Hermite splines than with B-Splines. In order to force synchronous movement with a Hermite spline, all actuator control points must lie at the same point in cycle time, since the control points represent the critical points of the control signal. The alteration of control points within a B-spline influences the location of the control signals critical points in a non-uniform manner, thus enforcing such a constraint with a B-spline system is far more computationally intensive.

In order to incorporate sensor feedback information into the spline controller, another dimension was added to the controller. The extended control points could now specify their locations within both the gaits cycle time, and the feedback value. This resulted in a set of control surfaces for each actuators cyclic information. The initialization splines were not extended as it was assumed that any robot would be initialized on flat

terrain. However, the initialization splines can be easily extended to contain information dependent on the terrains gradient through a similar method to that used to extend the cyclic splines. The number of control points required for the extended controller is given by Equation 12. Extending the controller in this form significantly increased the number of control points required. Figure 8 illustrates a resulting control surface for one actuator.

$$a \cdot (i + c \cdot f) \qquad 4.4$$

where f is the number of control points used to sample the feedback

Equation 12 – Control points required for extended spline controller

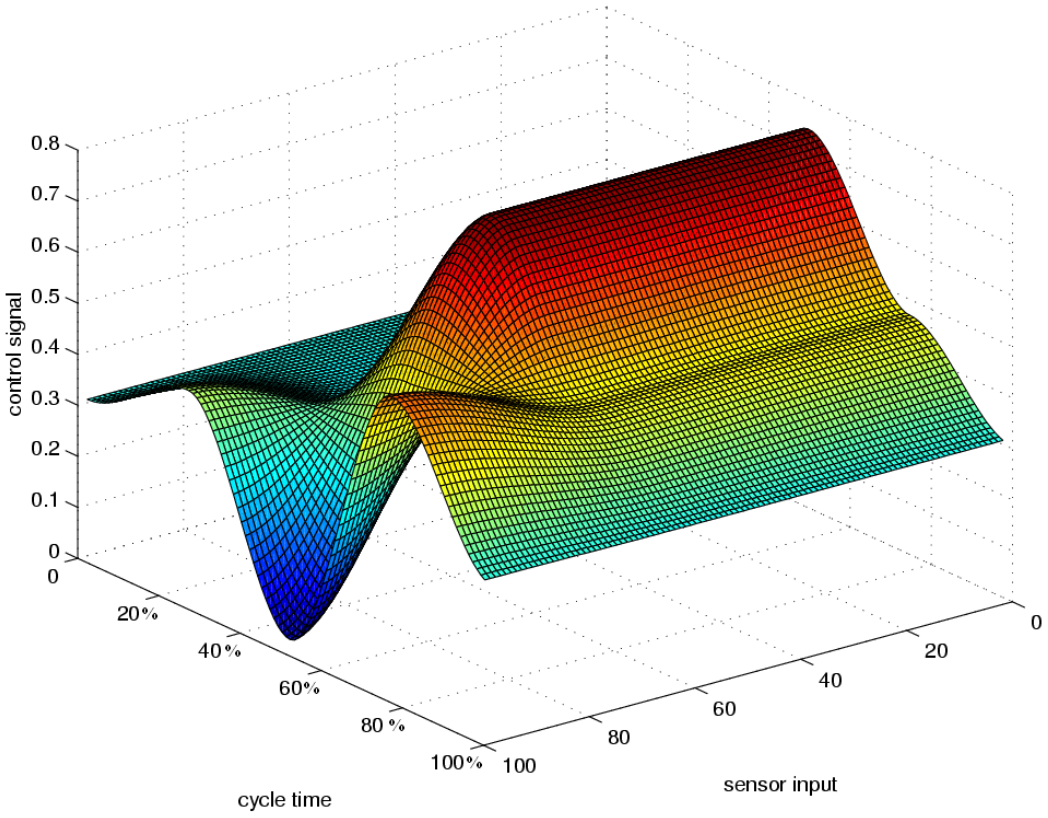

Figure 8 – Generic extended spline controller

The actuator evaluates the desired output value from the enhanced controller as a function of both the cycle time and the input reading from the sensor. The most appropriate sensory feedback was found to be an angle reading from an inclinometer placed on the robots central body (torso). Thus, the resultant controller was expressed in terms of the percentage cycle time, the inclinometers angle reading, and the output control signal.

4.3.2 Spline Controller Encoding

In order to evolve the spline controller with a genetic algorithm, the controller's parameters needed to be encoded into chromosome representations. Two different encoding schemes were implemented. Initially, a direct encoding method was utilized, as the use of this method is straightforward. After some initial analysis of results, indirect encoding was attempted to take advantage of the symmetrical nature of the control signals generated. However, this requires previous knowledge of the structure and desired walking pattern from the robot, and hence was later removed.

Staged Evolution

Support for staged evolution can only be achieved if the controller is specifically designed such that the evolution can proceed in this manner. The encoded spline controller supports staged evolution by progressively adding dimensions and reducing restrictions to the controllers control points.

Direct Encoding

Each control point value was treated as an 8-bit fixed point value. In the initial phase of evolution the control points locations within the walking cycle time were equally distributed. This provided each control point with only one degree of freedom, and reduced the available solution space but also significantly reduced the complexity of the chromosome required to describe the controller. In the following stage of evolution, the equally distributed time constraint was dropped, providing the control points with an additional dimension of freedom. Finally, the tangent values for the control points in the spline were added to the chromosome, allowing final refinement of the solution.

Indirect Encoding

There is a large amount of evidence that neural controllers for bipedal locomotion exhibit a high degree of symmetry [13,16]. Evolution periods required to generate the controllers have been significantly reduced by using indirect encoding schemes which take advantage of the symmetries found within the controller, and within the resulting walking motions. After initial analysis of preliminary results from the spline control, symmetric output could be observed in the resulting control signals. Thus, an indirect encoding method was applied in an attempt to reduce evolution periods. The control points for each actuator within the first half of the cyclic section were mirrored to correspond with the control points for the ending half of the actuator that was symmetrically opposite to the actuator. For example, the left knee's starting control points would be mirrored to the right knee's ending control points. This significantly decreased the complexity of the required chromosome structure.

Evolution Phase	Direct Encoding Complexity	Indirect Encoding Complexity
Phase 1	$a(s+c)$	$a(s+\frac{c}{2})$
Phase 2	$2a(s+c)$	$2as+c$
Phase 3	$3a(s+c)$	$3a(s+\frac{c}{2})$

Table 2 – Increasing chromosome complexity

Direct Encoding With Symmetry

Successful application of an indirect spline encoding scheme requires that the controller is developed with knowledge of the robot's kinematics and structure. Furthermore, the robot's physical design must also be symmetric. In order to allow the evolution of non-symmetric robots whilst still attempting to obtain the benefits available from the compact representation, the symmetric information was seeded into the directly encoded controller. This provides the opportunity for reduced evolution periods, whilst still allowing the full parameter set to be specified. As a result, any robot controller can be evolved regardless of its structure, and symmetric robots

benefit from the reduced evolution periods, as well as the ability to further refine the control parameters.

4.3.3 Neural Controller

Although there have been a number of neuron models proposed for locomotion control [24,43,13] the neuron model implemented was based on state-variable point model proposed by MacGregor[42]. This model was applied as MacGregor provided detailed explanations to the theory and the implementation details behind this model.

The neuron is based on a simplified version of Hodgkin and Huxley's equations [42], and uses only four state variables (potassium conductance, membrane voltage, threshold, and a binary action potential spike). The equations used to describe the model were given in Equation 6. MacGregor demonstrated the use of this neural model for repetitive firing (the PTNRN10 model), and thus the model could be easily extended to produce neural networks with oscillatory output.

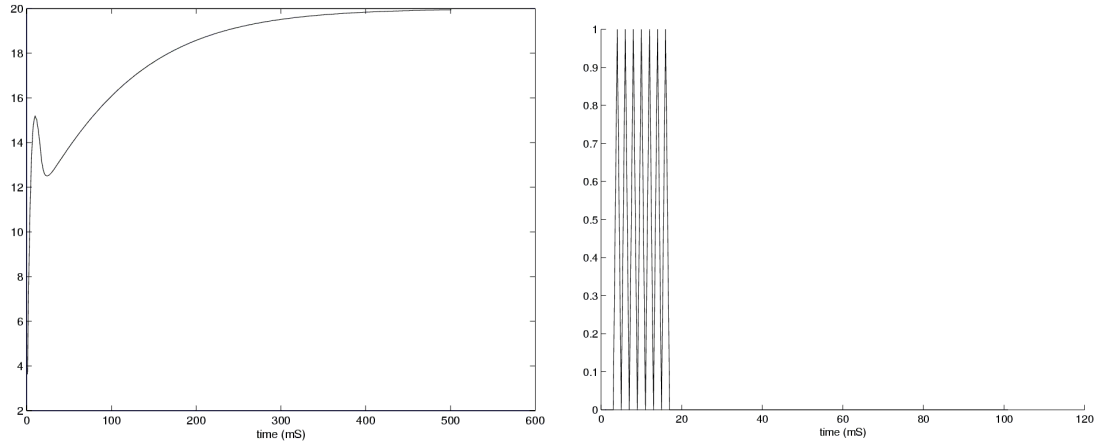

Figure 9 – Output Voltage and Spike for PTNRN

Constructing an Oscillator

Oscillators are typically constructed in neural networks using two coupled neurons with delayed inhibition and self-activation [57]. However it is possible to construct an oscillator from MacGregor's neuron model by taking the spike output

and directly connecting it to the neuron as a positive feedback current. This allowed oscillatory signals to be constructed from a single neuron. These individual neurons were then used to control an individual actuator, provided with a time input value. Sample output from the neural oscillator is shown in Figure 10

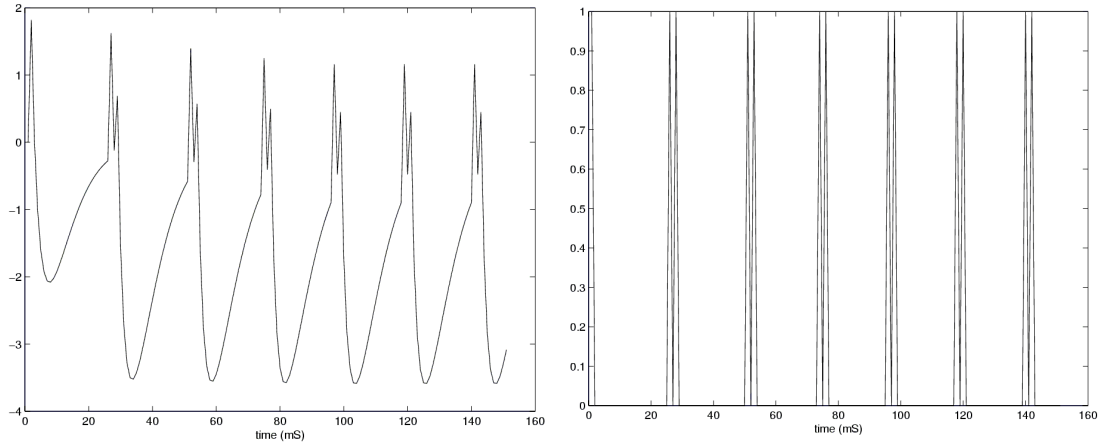

Figure 10 – Oscillator Output

Encoding

Although a number of optimal encoding schemes for neural controllers have been proposed [58], the neural controller was encoded using fully direct encoding methods. As the neurons were directly coupled to the actuators, there were no neural network parameters that needed to be encoded or evolved. The encoding for the PTNRN neurons is shown in Table 3.

Parameter Name	Range
ampGK	[0..32] (5.3 fixed point value)
EK	[-8..0] (3.5 fixed point value)
amp	[0..128] (7.1 fixed point value)
tMemb	[0..16] (4.4 fixed point value)
Multiplier	[-1..1] (1.7 fixed point value)
Integration Step Size	[0..4] (2.6 fixed point value)

Table 3 – Evolved Neuron Parameters and Encodings

4.3.4 PID Control

PID controllers provide a simple means of adding robust feedback control [28]. Coupling PID controllers to the neural and spline controllers enables a simple form of incorporated feedback. The PID controller interprets the control signals provided by the higher level controllers and transforms the higher level signals to the signals that drive the robot's actuators. Through the cascading of control systems a simple form of feedback can be incorporated, and the resulting gaits remain stable indefinitely.

The PID controllers were hand designed to provide a general form of control. The controllers were designed such that they would allow the robot to remain in a standing state when no other control signals were provided.

4.4 Parallel Computation

Further reductions in evolution time can be achieved by exploiting distributed computing methods. Since the genetic algorithm is inherently parallel, individual solutions can be evaluated on distributed workstations in parallel. The large diversity in processing power between workstations created the need for an efficient workload allocation procedure, which allowed faster workstations to evaluate more individuals than slower ones.

The parallel computation design followed a client-server methodology. The server is responsible for maintaining the GA and scheduling the simulation evaluations to the clients. Each client is given one individual to evaluate, then returns the result to the server. The client continues to wait for the server to allocate it another individual for evaluation. This scheduling algorithm allowed faster workstations to evaluate more individuals than slower workstations, and provides an effective means for load balancing.

The operating system, processing power, and processor architecture varied greatly between clients. This required that the communications system between the variety of client configurations and server be fully portable. As a result, the implemented communications system was based on network file sharing technology to allow for maximum portability across platforms.

4.4.1 Server

The server is responsible for executing the genetic algorithm and scheduling the workload evenly between clients. The server loads the robot configuration information from the simulator configuration files, and allows the user to adjust various control parameters. Any clients attempting a connection are detected and added to the available set of workstations.

The server begins distributed processing by passing the robot and simulation configuration information to the clients. Once the clients have configured the simulation environment, the server allocates a simulation task to each client. The server decodes the chromosome information and sends the controller parameters to the client for evaluation.

Upon client completion of the simulation, the results are recorded on the server and another task is delegated to the client. If the server has received all the required results for the current generation, the server will calculate the new individuals for the next generation, and then repeat the process of scheduling tasks to clients. Fault tolerance is integrated into the system by allowing any client to be connected, disconnected or reconnected to the server at any stage. The server automatically disconnects any client that fails to respond within a given time frame, and removes them from the available processing pool.

4.4.2 Clients

On startup, the clients attempt to connect to the server and indicate their availability. They then wait for the server to send the robot and simulation configuration information. The clients then configure and spawn the simulated

environment. Once the simulated environment is initialized, each client waits to receive controller parameters. The gait described by these parameters is then simulated and evaluated according to the fitness function provided by the server. The resulting fitness value is returned to the server, and the client continues to wait for the next set of control parameters to be sent.

4.5 System Summary

The software package developed consists of three major components. There are a number of issues that needed to be addressed in managing communications between each software module. The overall software system is illustrated in Figure 11.

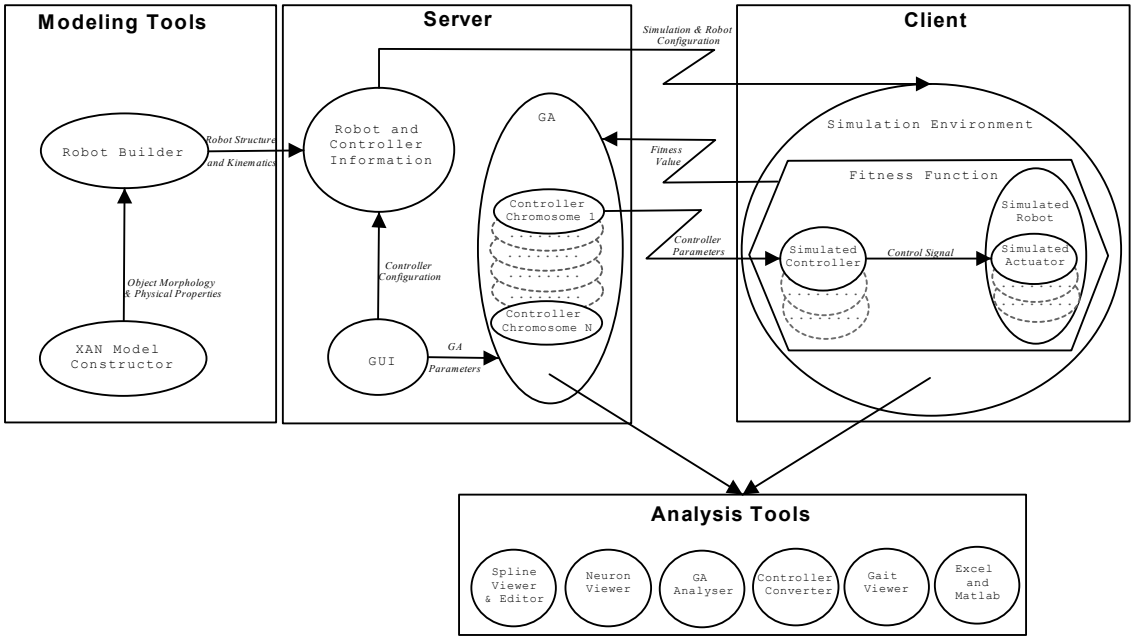

Figure 11 – System Overview

The system has been divided into four software sub-sections:

- Modeling Tools – This includes the software required to design the overall robot structure, and the software to construct an objects morphology and generate its physical characteristics.

- Server – Contains the main interface to the user, allowing them to configure the robots controller, the simulator environment, and genetic algorithm. The server executes the actual genetic algorithm and performs individual fitness evaluations in parallel on the clients.
- Client – Contains the simulation environment and evaluates individual controllers and returns the fitness value to the server.
- Analysis Tools – This is a set of software utilities created in order to analyze and visualize various aspects of the control systems, genetic algorithm, and generated gait.

The steps required to generate a gait for a specified robot design are given below:

1. Construct individual robot objects based on the objects shape, dimensions and mass.
2. Combine the individual objects in order to realize the overall robot design.
3. Load the robot's structural information into the main GA server and configure the robots control system parameters.
4. Configure the genetic algorithm parameters and fitness function.
5. Evolve the robot controller
6. Investigate the generated results using the provided set of analysis tools to confirm the desired gait has been achieved.

5. Investigations

5.1 Genetic Algorithm

A number of factors influence the performance of the genetic algorithm. These include chromosome complexity, the operators used, selection weightings for the operators, the selection scheme, the fitness function utilized and the problem type. The investigations undertaken reveal interesting results about the performance enhancement gained from the operators, selection schemes, and fitness functions employed.

5.1.1 Method

Four differing configurations of the genetic algorithm were examined. The first was a traditional GA using roulette wheel selection with two operators: a one-point crossover, and a mutate operator. This configuration was employed as this is a commonly implemented version of the genetic algorithm [17], and hence provides an adequate benchmark figure.

The second configuration used roulette wheel selection with more advanced operators: inversion, non-binary average combination, and random replacement. The third GA was based on the second configuration, except that half of the chromosomes were generated using random selection. The fourth configuration extended the third and contained a non-binary creep operator, as well as supporting higher values of elitism.

The parameters used for each GA configuration are shown in Table 4. The percentages indicate the percentage chance each item has of being used in the creation of a new chromosome.

Genetic Algorithm	Selection Schemes	Operators
1. Traditional	Roulette Wheel	Crossover (70%) Mutate (30%)
2. Enhanced	Roulette Wheel	Crossover (30%), Mutate (10%), Random Replacement (25%), Average (30%), Inversion (5%)
3. Enhanced with random selection	Roulette Wheel (50%) and Random Selection (50%)	Crossover (30%), Mutate (10%), Random Replacement (25%), Average (30%), Inversion (5%)
4. Enhanced with high elitism, random selection, and non-binary creep	Roulette Wheel (50%), Random Selection (25%) and Elitist (top ten percent) Selection (25%)	Crossover (35%), Mutate (10%), Random Replacement (10%) , Average (30%), Inversion (5%), Creep(10%)

Table 4 – Genetic Algorithm Parameters

Each version of the genetic algorithm ran through three different trials involving different fitness function terminating conditions:

- Non-terminating fitness function. This fitness function did not include any premature termination conditions, and simply continued evaluation of the robot until the specified simulation time was reached. The evolution was completed without an intermediary stage.
- Premature-terminating fitness function. Two termination conditions were added to the fitness function. The system would terminate if the robot fell over, or if both feet were off the ground for too long (jumping etc.)

- Premature-terminating fitness function with one stage of evolution. The GA ran as above, except that the system progressed through one phase of evolution.

5.1.2 Results and Discussion

The graph shown in Figure 12 shows the performance alteration when using early terminating and staged evolution techniques, relative to non-terminating conditions. From the results obtained, early termination appears to produce reduced evolution periods averaging at 39.8% of the original evolution times. Inclusion of staged evolution reduces the times are by a further 14.5%. Thus, by using both staged evolution and early termination, the evolution period is on average reduced to 34% of the original evolution time.

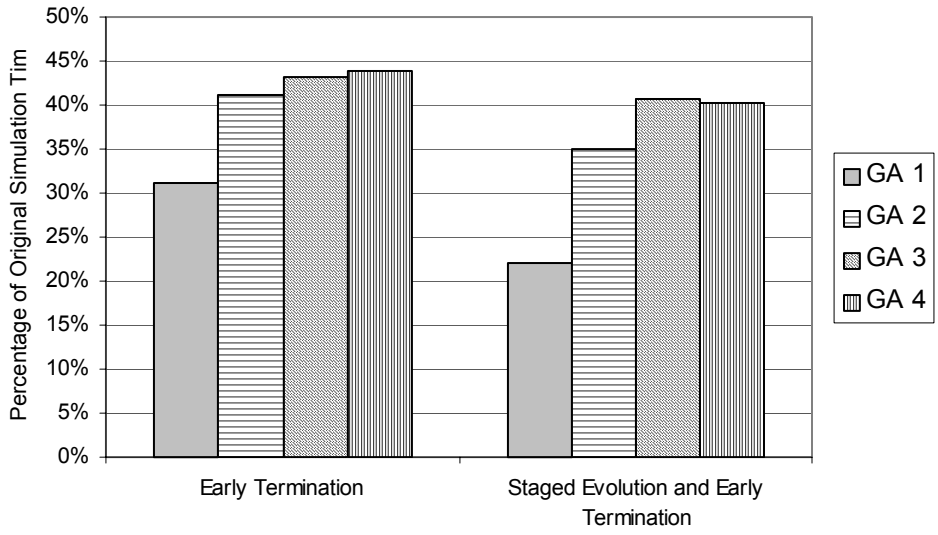

Figure 12 – Change in required simulation time

The average increase in fitness of the population per generation is shown in Figure 13. Early termination causes the conventional genetic algorithm to decrease its average increase in fitness. However, other forms of genetic algorithms appear to gain in fitness. A possible explanation for the abnormal result from the conventional algorithm is that it relies on the randomness present in other chromosomes in order to generate new chromosomes with increased fitness. Each of the other genetic

algorithms contain either an operator which can introduce randomness (random replacement), or a selection scheme which increases randomness (random selection).

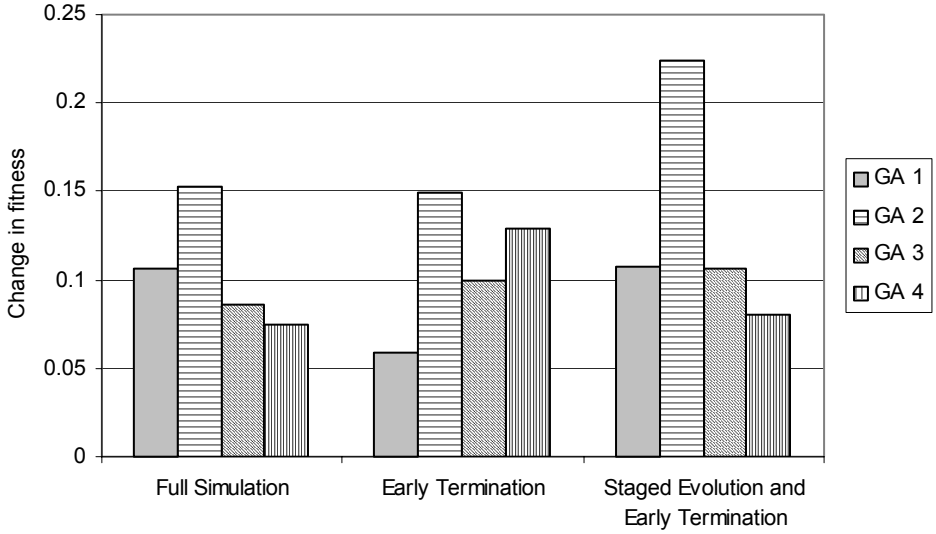

Figure 13 – Average increase in fitness per generation

For staged evolution with early termination, the fitness per generation greatly increases for the first two forms of the genetic algorithm (conventional & enhanced). However, this increase in performance is not realized in the GAs which involve random replacement and random selection. The GA configurations using random selection take longer to converge towards a solution. Thus, when the staged evolution takes place in the tenth generation, the population is not close enough to a solution to be able to benefit from moving to the next stage. The other two forms of GA have somewhat converged by this time, and hence are able to take advantage from the next stage.

The performance of each operator, in terms of the percentage of positive fitness chromosomes generated relative to other operators, is depicted in Table 5 below.

Operator Name	Early Phase	Late Phase
Crossover	16.61%	28.46%
Average	26.25%	25.18%
Random Replace	19.81%	16.89%
Mutate	15.52%	5.81%
Creep	17.90%	22.90%

Table 5 – Relative chromosome generation success rate

The results indicate that the performance of the non-random operators remains unchanged or improves during later phases of evolution, whereas random based operators decrease their performance. This result confirms that increased randomness in the initial phase of evolution is beneficial as it broadens the search space of the algorithm. Thus, as would be predicted, the random operators do not provide as much benefit in later stages of evolution. An interesting outlier from this generalization is the non-binary creep operator. Although the random replacement, and mutate operators decrease their percentage success rate, the creep operator increases its success rate in later phases of evolution. This result can be explained from the basis of the non-binary nature of the creep operator.

During later phases of evolution, small alterations to the chromosome are more likely to produce improved fitness ratings. As the population converges towards the optimal solution, any large alterations to the chromosome will result in a solution further from the optimal value. Since the creep operator performs only small modifications to the chromosome, it is far more likely to produce fit chromosomes within the converging solution space.

5.1.3 Conclusion

From the results obtained it would indicate that early simulation termination can provide significant reductions in evolution times, and genetic algorithms which produce enough randomness will not be adversely affected by having the worse chromosomes terminated. The ability to take advantage of the staged evolution

process appears to depend on the current status of the population of the GA. If the population has begun to converge, then the staged evolution opens up a new prospect for higher rating fitness chromosomes. Random operators and selection schemes appear to be beneficial during early phases of evolution, whereas the crossover-based operators perform consistently during the entire evolution period.

5.2 Evolving Gaits with Splines

The two robots models utilized for the investigation of the spline control system consist of two regular, geometrically identical legs, with three controlled joints. The first robot (Figure 14 - Left) had large feet, and lacked a complete torso. The second robot provided a more realistic representation of a robot and has dimensions comparable to a human being (Figure 14 - Right). Both robots have 3 controlled joints per limb, and thus a total of 6 joint controllers for each gait were evolved.

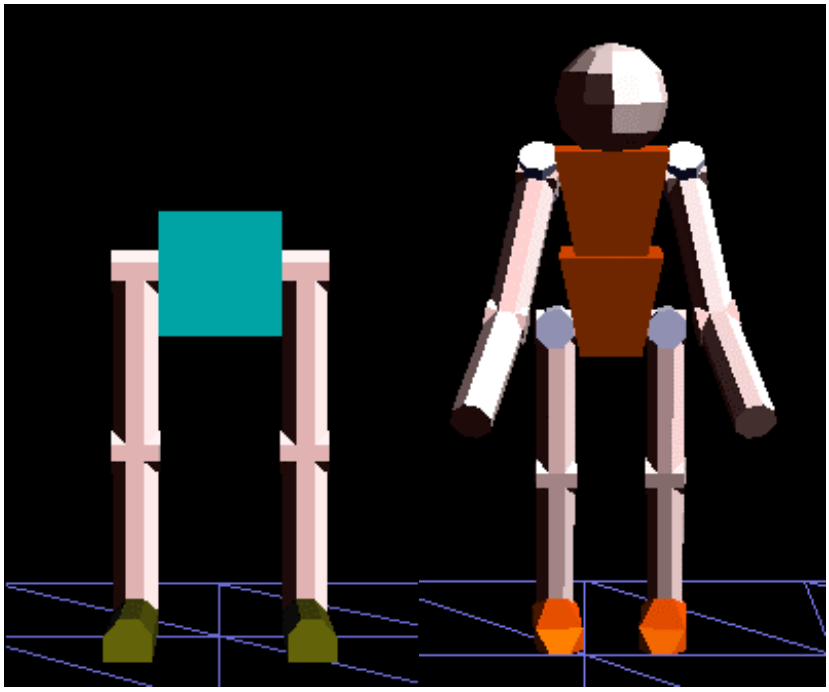

Figure 14 – Biped Robot Models

5.2.1 Spline Controller

The standard spline controller successfully described a full walking gait for a simple bipedal robot. An early successful gait is depicted in Figure 15. The spline controller consisted of four cyclic control points and four initialization control points. An indirectly encoded chromosome representation for this controller could be evolved to produce adequate walking patterns within 12 hours on a 500MHz Athlon PC. Although there were only four control points, these were sufficient to describe the complete walking cycle for the simple robot.

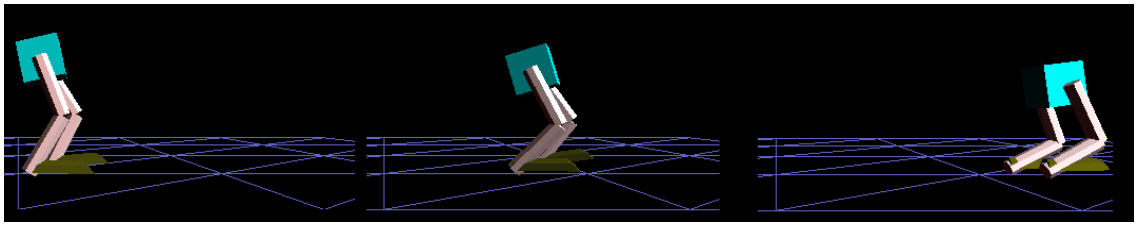

Figure 15 – Early Biped Gait

The resulting controllers did not produce gaits that could maintain locomotion for an extended period of time. As is clearly indicated in the above sequence of images, the torso of the robot slowly lowers towards the ground as the gait progresses. To overcome this, sensory feedback can be integrated into the control system. This feedback information can be used to correct deviations from the desired signals.

5.2.2 Spline and PID Control

Incorporation of PID controllers with the spline control system enabled a simple form of feedback that allowed the robot's walking cycle to be maintained indefinitely. The PID controller ensured that the robot's joints reached the joint position specified by the spline control system. Figure 16 illustrates the spline control system coupled with a PID controller providing continued locomotion.

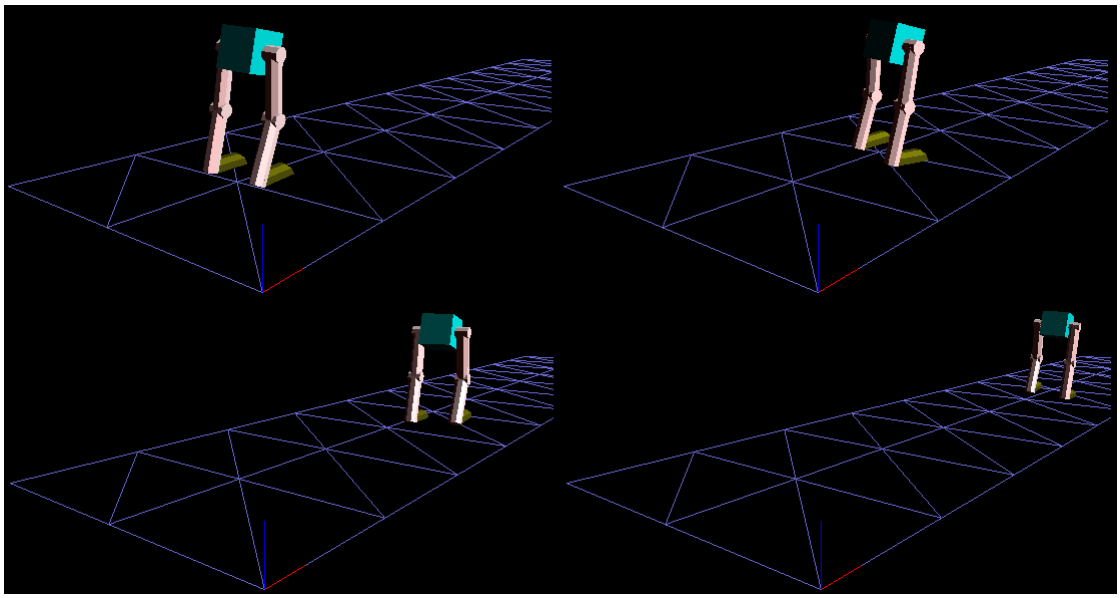

Figure 16 – Sustained Robot Walk

The extended control system was capable of expressing complex control signals responsible for dynamic gaits with a robot that was far more complicated to balance. This allowed the control of the less stable android robot, and enabled the control of complicated movements, such as the jumping gait illustrated in Figure 17. The resultant control system depicted was evolved within 60 generations and began convergence towards a unified solution within 30 generations.

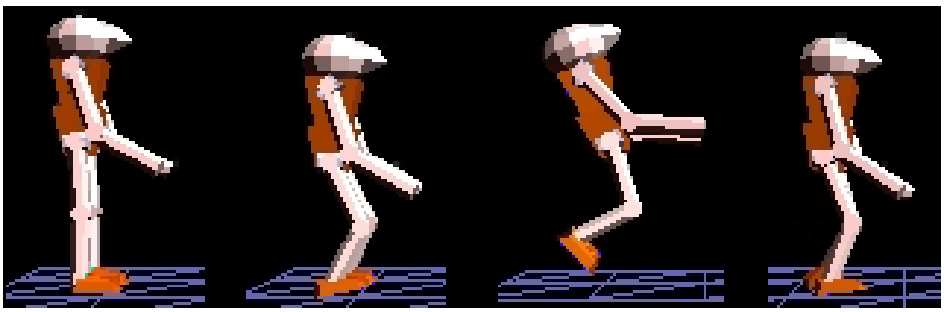

Figure 17 – Jumping Robot

5.2.3 Spline Control with Integrated Feedback

Extending the spline controller to incorporate sensory information provided a far more adaptable and robust control system. The inclinometer reading was successfully interpreted by the control system to provide an added level of feedback

capable of sustaining the generated gait over non-uniform terrain. An example of the resultant gait is illustrated below. The controller required over 4 days of computation time on a 800MHz Pentium 3 system, and was the result of 512 generations of evaluation.

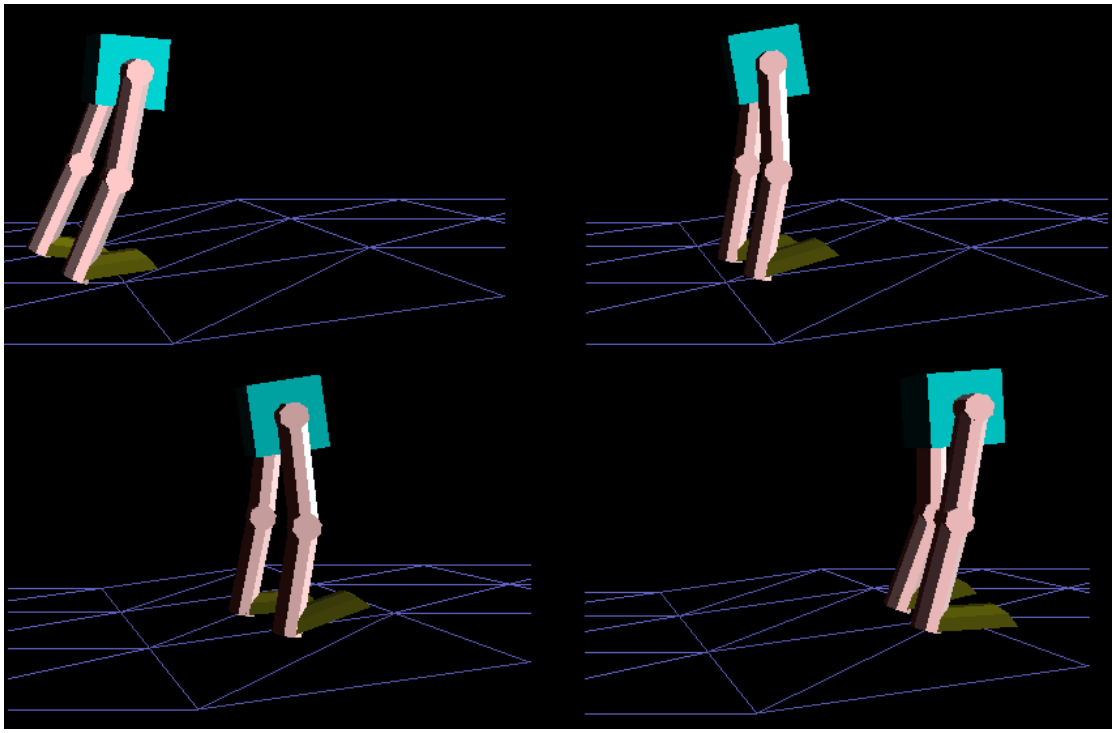

Figure 18 – Robot walking over terrain

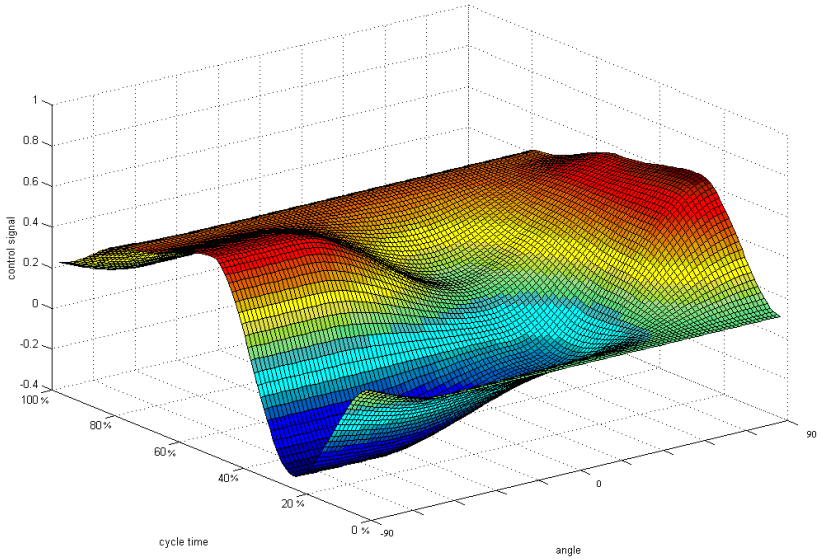

Figure 19 – Extended Spline Controller

An example control surface for one of the actuators generated by the spline controller is depicted in Figure 19. The change in the output signal generated when the inclinometer is providing different readings can be clearly seen. The fitness values during the evolution are illustrated in

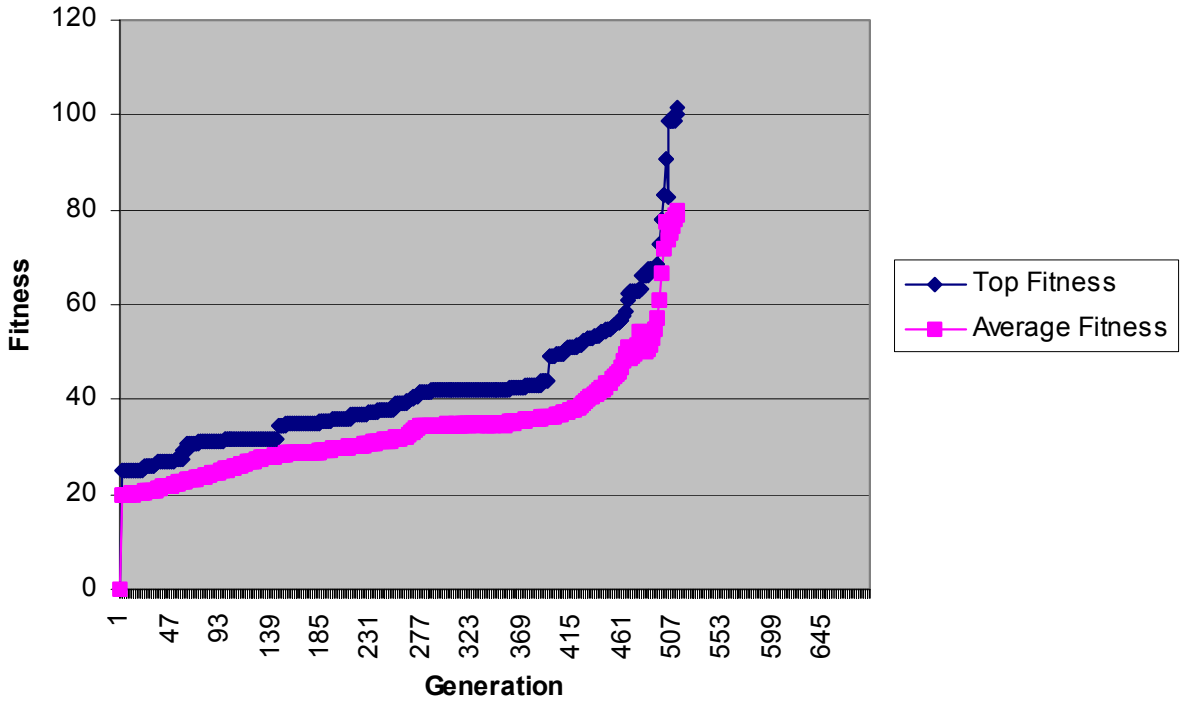

Figure 20. A rapid increase in fitness values can clearly be observed at around 490 generations. This corresponds to the convergence point where the optimal solution is located.

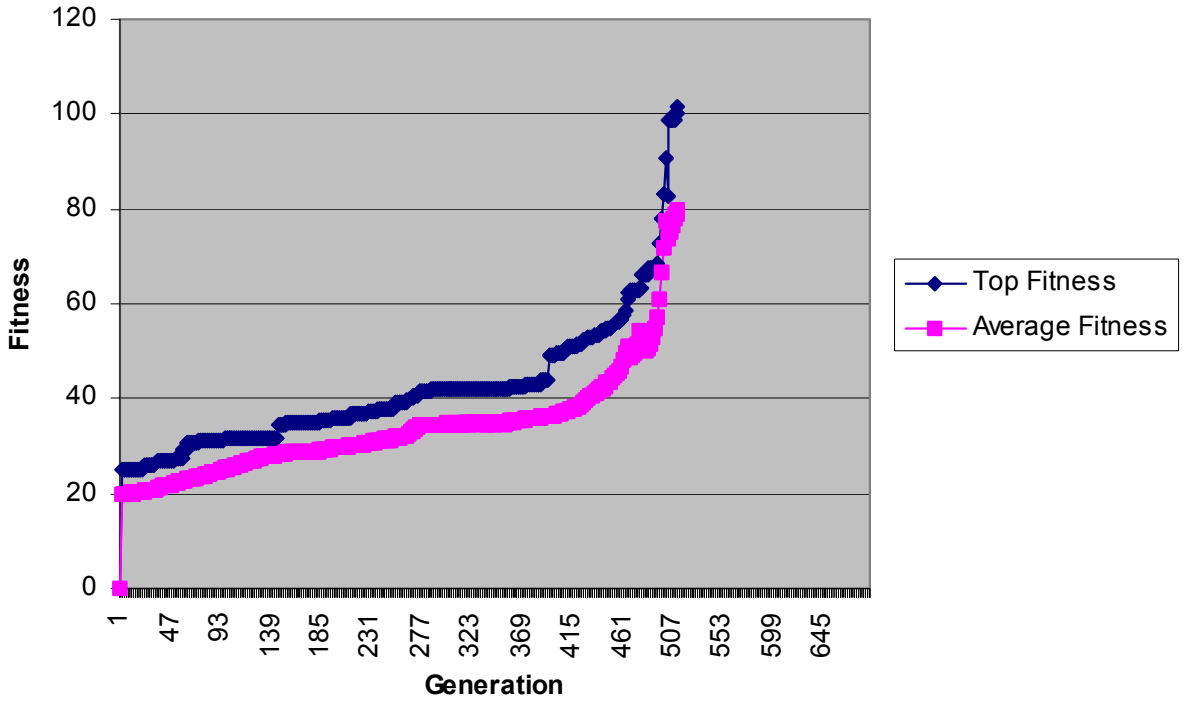

Figure 20 – Fitness vs Generation for Extended spline controller

5.2.4 Non Bipedal Robots

The spline control system also successfully demonstrated control of non-legged robots. The snake robot model shown in Figure 21 consists of five joints, each with only one degree of freedom (sideward motion). Figure 21 clearly illustrates the snake's sinusoidal movement, as evident in real snakes.

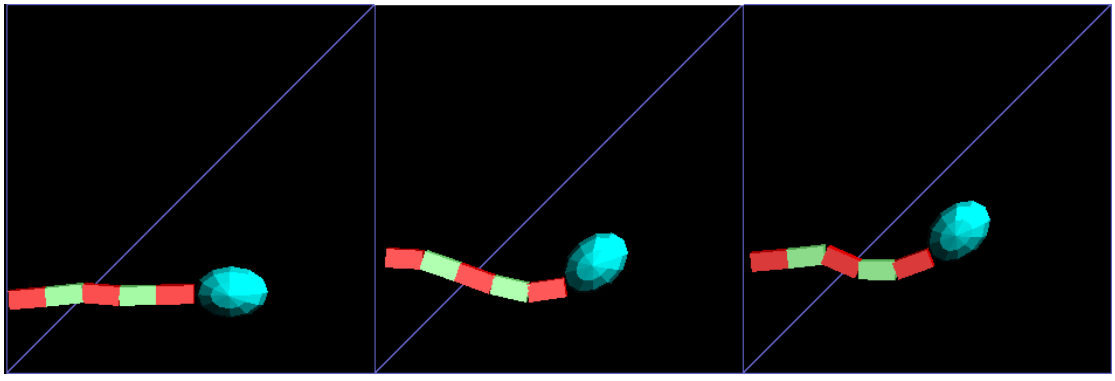

Figure 21 – Snake Gait

Gaits for non-bipedal legged robots were also successfully evolved utilizing the spline control system. The most successful gait evolved for the tripod robot is illustrated in Figure 22. The robot achieves forwards motion by thrusting its rear leg towards the ground, and lifting its forelimbs. The robot then gallops with is fore limbs to produce a dynamic gait.

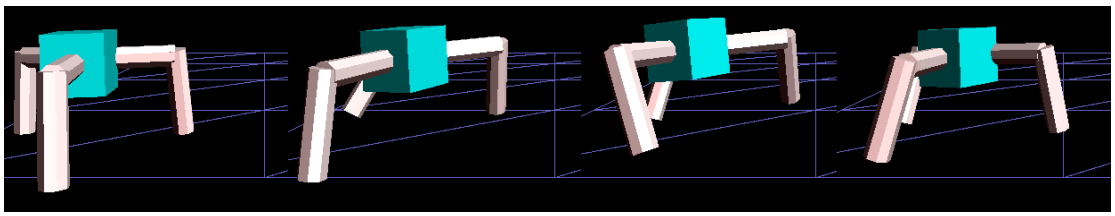

Figure 22 – Tripod Gait

5.2.5 Conclusion

The spline controller was applied to a number of robots with differing morphology. The controller expressed a number of diverse gaits for various robot designs, and proved to be readily extendible. The expanded control system successfully controlled a bipedal robot over rugged terrain, and demonstrated the powerful capabilities of this simple control system.

5.3 Neural Controller

Neural net controllers have already successfully demonstrated control of legged robots [4,16]. The neuron model implemented demonstrated oscillatory output when its output was directly fed back into one of the neurons input channels [42]. The investigation undertaken examines the ability for a single neuron to provide actuator control, the evolution periods required to evolve a gait, and the ability of the software to support a different control mechanism.

5.3.1 Method

Each individual neuron was directly coupled to an actuator or an actuators controller. Thus the encoded chromosome consisted entirely of the string of neuron parameters, and did not need to evolve any network weightings. The parameters evolved for the neural controller were given in Table 3.

The neural controller was evaluated on the simple biped robot illustrated in Figure 14. Three separate versions of controller evolution were attempted, each with differing configurations of the PID controller. The first version contained a PID controller configuration that was successfully employed with the spline control system. This controller was initialized to provide the correct control signals to maintain the robot in a standing state when no further input was provided. The second version was similar to the first, except that the DC value for each actuator was not set. Finally, the neural controller was directly connected to the actuator with no form of intermediary control. Each of these versions was permitted to evolve over a period of 24 CPU hours on an 800MHz Pentium 3 system.

5.3.2 Results and Discussion

The neural controller did not provide lifelike gaits for the simple robot within the constrained evolution time. The directly connected neural controller did not produce any successful gaits at all. The neural controller connected with the PID controller did produce gaits which resulted in forwards motion, however none of the gaits generated within the 24 hours produced lifelike results. The most successful gait generated is illustrated in Figure 23.

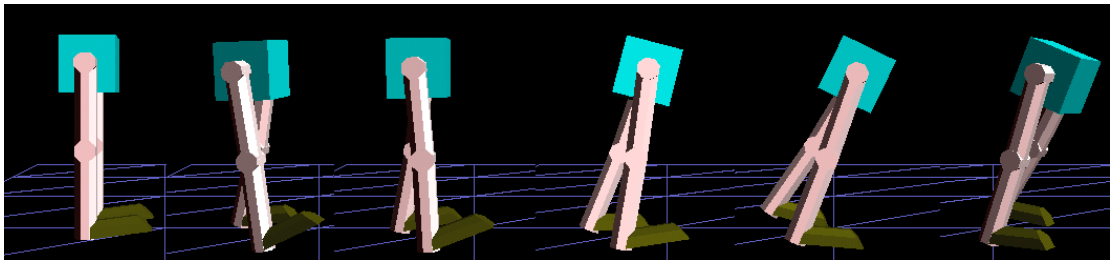

Figure 23 – Gait produced from Neural Controller

Given the severe constraints placed on the neural network it is not surprising that the controller could not express any adequate gaits. All previously successful neural controllers utilized four or more neurons combined to control each actuator [4,21,16,13]. Furthermore, the neural net was usually provided with a range of sensory input values, and not simply constrained to time as its only input. The neuron models used by most researches were typically more complex and usually based from direct observations of neuron models responsible for locomotion in animals [23,24].

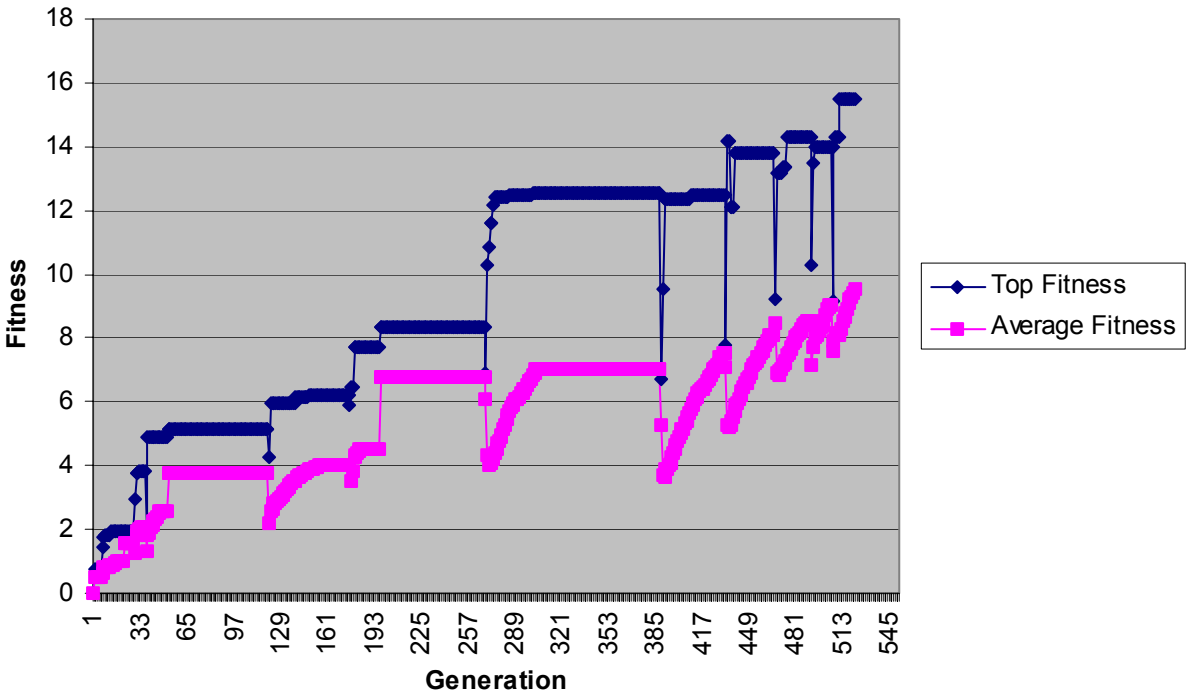

Figure 24 – Fitness vs Generation for Neural Controller

Figure 24 shows the change in fitness per generation during the evolution of the neural controller. The controller required 521 generations to generate a gait that resulted in adequate forward motion. A similar number of generations were required to evolve the robust spline control system previously described. The graph indicates that the neural controller's performance stagnates, until the genetic algorithm is reset and new random chromosomes are introduced.

5.3.3 Conclusion

The neural controller implemented did not generate the desired results in its presented configuration within the permitted time frame. Previous evidence suggests that MacGregor's model could produce better results if sensory feedback information was incorporated and the neural network was expanded [14]. Lewis et al [4] indicate that the evolution periods for directly evolving a neural controller can be extremely large. Thus, it is possible that the proposed neural control system could successfully generate lifelike gaits given lengthier evolution periods. Although the neural control system was generally unsuccessful, the system did prove that the design was capable of evolving, simulating, and evaluating differing controller types.

6. Conclusions

6.1 Summary of Contributions

This thesis presents a flexible architecture for controller evolution. A new form of spline locomotion control was proposed and investigated. The Hermite spline controller was shown to be capable of describing complex dynamic walking gaits for robots with differing morphologies. The spline controller offered a number of advantages over a neural based approach. Directly encoded fully connected neural networks have prohibitively long evolution periods [13]. Indirectly encoded networks significantly reduced the evolution periods but cannot be automatically evolved. However, the spline controller is sufficiently simple that the directly encoded controller can be evolved in an acceptable time frame.

The software developed provides a number of useful tools for robot designers. The modeling tools and libraries simplify the robot design task, and enable a simple form of information exchange between various utilities. The genetic algorithm is implemented in a flexible and extendible manner.

This thesis investigated a number of aspects of genetic algorithms used for evolution of robot controllers. It was found that randomness present in the genetic algorithm provided advantages during the early phases of evolution, but degraded performance in later phases. Optimizations of the fitness function and evolution strategies were also determined to be dependent on the genetic algorithms configuration, and current state.

Two control systems for robot locomotion were investigated and discussed. Extension of the basic spline controller to incorporate feedback information was proven to be viable, but required a significant increase in the evolution period. A sample neural network controller was presented, but found to be too simple to provide adequate control for acceptable gaits.

6.2 Future Outlook

The implementation of the neural controller was very rudimentary and did not achieve acceptable results. In order for any comparison between control methodologies to be adequately made an improved neural control system must be implemented. One of the major disadvantages when evolving neural controllers is the need for designer input [13]. Initially evolving a spline control system, and then training the neural network with the spline controllers output could provide a method for conversion between control systems, and also allow full automation of the neural evolution.

Further investigations into the spline control system are necessary to establish its extensibility and applicability. Expansion of the controller to provide for n-dimensional input would allow the controller to receive input from any number of sensors. However, the performance impacts of such extensions would have to be carefully examined. The current system requires the designer to specify the number of control points utilized to describe the control spline. This process could be automated by having the system add control points to a spline if the system was incapable of describing a successful gait within a given time frame.

Executing the control systems on real robot hardware would facilitate a direct evaluation to the practical performance of the spline control system. This would also allow investigation of how the controller copes with being passed across from simulation onto an actual robot.

Finally, evolution of the robot morphology and the control system in unison would enable the robot's design to be improved, such that the robot's structure was optimally designed to suit its desired purpose. Further extensions of this could be to automatically construct the designed robots using 3d printing technology, removing the human designer completely from the robot design process [59]

References

[1] M. H. Raibert. *Legged Robots That Balance* Cambridge, MA: The MIT Press, 1986

[2] G. S. Hornby, S. Takamura, J.Yokono, O. Hanagata, T. Yamamoto, M. Fujita, "Evolving Robust Gaits with AIBO" in *IEEE International Conference on Robotics and Automation*. pp. 3040-3045, 2000.

[3] D. Wettergreen, C. Thorpe, W.L. Whittaker. "Exploring Mount Erebus by Walking Robot" in *International Conference of Intelligent Autonomous Systems*, pp. pp72-81. 1993.

[4] M. Lewis, A. Fagg, G. Bekey. "Genetic Algorithms for Gait Synthesis in a Hexapod Robot." in *Recent Trends in Mobile Robots* (World Scientific, New Jersey), pp. 317-331. 1994.

[5] A. Takanishi, M. Ishida, Y., Yamazaki and I. Kato. "The realization of dynamic walking by the biped walking robot WL-10RD." in *Proceedings of the 1985 ICAR*. pp. 459-466, 1985.

[6] G.B. Parker, D.W. Braun, I. Cyliax, "Learning Gaits for the Stiquito" *Proceedings of the 8th International Conference on Advanced Robotics (ICAR'97)*. pp. 285-290. 1997.

[7] S. Kim, J. Oh, K. Lee, "Design of 4 Joints 3 Links Biped Robot and Its Gaits", in *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp 523, 2000.

[8] Adams, B., Cynthia Breazeal, Rodney A. Brooks, Brian Scassellati, "Humanoid Robots: A New Kind of Tool", *IEEE Intelligent Systems and Their Applications: Special Issue on Humanoid Robotics,* Vol. 15, No. 4, July/August, pp. 25-31, 2000.

[9] D. Arnold “Evolution of Legged Locomotion”, MSc. thesis, Simon Fraser University, School of Computing Science, 1997.

[10] E. Muybridge. *Animals in Motion*. 1887.

[11] J.H. Holland, *Adaptation in Natural and Artificial Systems*, Cambridge, MA: MIT Press, 1975.

[12] D. Beasley, D. R. Bull, and R. R. Martin, “An Overview of Genetic Algorithms:Part 2, Research Topics” *University Computing*, vol. 15, no. 4, pp. 170-181, 1993.

[13] M. Krueger “Genetic Algorithms can Evolve Neural Network Controllers for Robot Locomotion”, BE thesis, University of Western Australia, 2001

[14] R. Reeve “Generating walking behaviors in legged robots”, Ph.D. thesis, University of Edinburgh, 1999.

[15] J.Ventrella,“Explorations in the Emergence of Morphology and Locomotion Behavior in Animated Characters “ in *Artifical List IV*, (R.A. Brooks and P. Maes Eds.) MIT Press, Cambridge, MA pp. 436-441, 1994.

[16] A. J. Ijspeert, “Evolution of neural controllers for salamander-like locomotion” in *Proceedings of Sensor Fusion and Decentralised Control in Robotics Systems II* pp. 168-179, 1999.

[17] D. Beasley, D. R. Bull, and R. R. Martin, “An Overview of Genetic Algorithms: Part I, Fundamentals” *University Computing*, vol. 15, no. 2, pp. 58-69, 1993.

[18] F. Busetti, ”Genetic algorithms overview”. [Online] Available: http://citeseer.nj.nec.com/464346.html

[19] C. Ledger. “Automated Synthesis and Optimization of Robot Configurations: An Evolutionary Approach” Ph.D. thesis, Carnegie Mellon University, 1999.

[20] T.S. Hussain. Methods of Combining Neural Networks and Genetic Algorithms, ITRC/TRIO Tutorial, Queens University, 1997.

[21] M. Lewis, A. Fagg, A. Solidum, “Genetic Programming approach to the Construction of a Neural Network for Control of a Walking Robot” in *IEEE International Conference on Robotics and Automation* (Nice, France) pp 2618-23, 1992.

[22] J. Busch, J. Ziegler, C. Aue, A. Ross, D. Sawitzki, W. Banzhaf, “Automatic generation of Control Programs for Walking Robots Using Genetic Programming” in *European Conference on Genetic Programming (EURO GP),* pp. 259-268, 2002.

[23] P. Wallen, O. Ekeberg, A. Lansner, L. Brodin, H. Traven, and S. Grillner. “A computer based model for realistic simulations of neural networks. II. The segmental network generating locomotor rhymicity in the lamprey.” *Journal of Neurophysiology*, pp. 1939-1950, 1992.

[24] R. Beer. “On the dynamics of small continues-time recurrent neural networks”. *Adaptive Behaviour*, vol 1, pp91-122, 1992

[25] K. Gurney, *Neural Nets*, UCL Press Ltd, 2002.

[26] Bartels, R. H., Beatty, J. C. and Barsky, B. A. *An Introduction to Splines for use in Computer Graphics and Geometric Models,* Morgan Kaufmann, 1987.

[27] J. Ziegler, W. Banzhaf, “Evolution of Robot Leg Movements in a Physical Simulation” *in Proceedings of the Fourth International Conference on Climbing and Walking Robots*, (K. Berns and R. Dillmann, Eds.), Professional Engineering Publishing, 2001.

[28] R.C. Dorf, R.H. Bishop. *Modern Control Systems*. 9th Edition. Upper Saddle River, NJ. Prentice Hall, 2001.

[29] S. Bennett, "The Development of the PID Controller" in *IEEE Control Systems*, pp 58-64, 1993.

[30] H.R. *Everett Sensors for Mobile Robots - Theory and Application*. Wellesley, MA. 1995

[31] E.J. Marey. *Animal Mechanisms: a treatise on terrestrial and aerial locomotion*. Appelton, New York, 1874.

[32] R. A. Liston, R. S. Mosher. "A versatile walking truck." *Proceedings of the Transportation Engineering Conference.* London: Institution of Civil Engineers. 1968

[33] M. H Raibert, I. E. Sutherland. (1983). Machines that walk. *Scientific American* 248(2): 44–53.

[34] K. DeJong. "The analysis of a Class of Genetic Adaptive Systems" PhD thesis, University of Michigan, 1975.

[35] M.F. Bramlette. "Initialization, mutation and selection methods in genetic algorithms for function optimisation." in *Proceedings of the Fourth International Conference on Genetic Algorithms*, (R.K. Belew and L.B. Booker Eds.), pp 100-107. Morgan Kaufmann, 1991.

[36] L. Davis *Handbook of Genetic Algorithms* Van Nostrand Reinhold, 1991.

[37] J.R. Koza. *Genetic Programming: On The Programming of Computers By Means Of Natural Selection* Cambridge, MA: MIT Press, 1992.

[38] P. Nordin, W. Banzhaf, "An On-Line Method to Evolve Behaviour and to Control a Miniature Robot in Real Time with Genetic Programming" in *Adaptive Behaviour*, vol 5 no 2 pp 107-140, 1997.

[39] A. J. Ijspeert, J. Kodajabachian, "Evolution and Development of a Central Pattern Generator for the Swimming of a Lamprey" in *Artifical Life* 1998.

[40] H. de Garis. "Genetic Programming : Building Nanobrains with Genetically Programmed Neural Network Modules." in *Proceedings of the International Joint Conference on Neural Networks*, July 1990

[41] J. Zhang, K.V. Le, A.Knoll "Unsupervised Learning of Control Surfaces Based on B-Spline Models" in *Proceedings of IEEE Int. Conf. on Fuzzy Systems*, 1997.

[42] R.J. MacGregor. *Neural and Brain Modeling*, San Diego, CA: Academic Press, 1987

[43] G. Taga, "A model of the neuro-musculo-skeletal system for human locomotion." *Biological Cybernetics*. vol 73. pp 97-111 1995.

[44] MathsEngine [Online] http://www.mathengine.com/

[45] Open Dynamics Engine [Online] http://www.q12.org/ode/ode.html

[46] T. Taylor, C. Massey. "Recent Developments in the Evolution of Morphologies and Controllers for Physically Simulated Creatures". *Artificial Life* vol. 7(1). pp. 77-87. (2001)

[47] C Ledger. Darwin2K: simulation and automated synthesis for robotics [Online] http://www.darwin2k.com/

[48] H. Keller, H. Stolz, A. Ziegler, T Bräunl, AERO: A Physically Based Simulation and Animation System. [Online] Available: http://www.aero-simulation.de/, [13th June 2002]

[49] DynaMechs (Dynamics of Mechanisms): A Multibody Dynamic Simulation Library [Online] http://dynamechs.sourceforge.net/

[50] S. McMillan, *Computational Dynamics for Robotic Systems on Land and Under Water*, Ph.D. Thesis, The Ohio State University, Columbus, OH, Summer 1994.

[51] S. McMillan, "Dynamechs v3.0 Refernce Manual." [Online] Available: http://dynamechs.sourceforge.net/ref_manual.ps

[52] G-MAX [Online] Available: http://www.discreet.com/products/gmax/

[53] S. Rodenbas, RobotBuilder [Online] http://eewww.eng.ohio-state.edu/~rodenbas/

[54] K. Entacher. Linear congruential generator: LCG [Online] http://random.mat.sbg.ac.at/~charly/server/node3.html

[55] Z. A. Karian and R. Goyal "Random Number Generation and Testing" *Maple Technical Newsletter,* vol 1, no 1, 1994

[56] J. Rose, J.G. Gamble. *Human Walking* Baltimore: Williams and Wilkins, 2nd Edition, 1994.

[57] B. Ermentrout, "An Introduction to Neural Oscillators" in *Neural modelling and neural networks*, pp 79 – 110. Pergamon Press. 1994.

[58] F. Gruau, Automatic definition of modular neural networks. *Adaptive Behavior*, 2, pp.151-183, 1995.

[59] H. Lipson, J. B. Pollack. "Evolving Physical Creatures" [Online] Available: http://citeseer.nj.nec.com/523984.html

Appendix 1. Evolved Gait

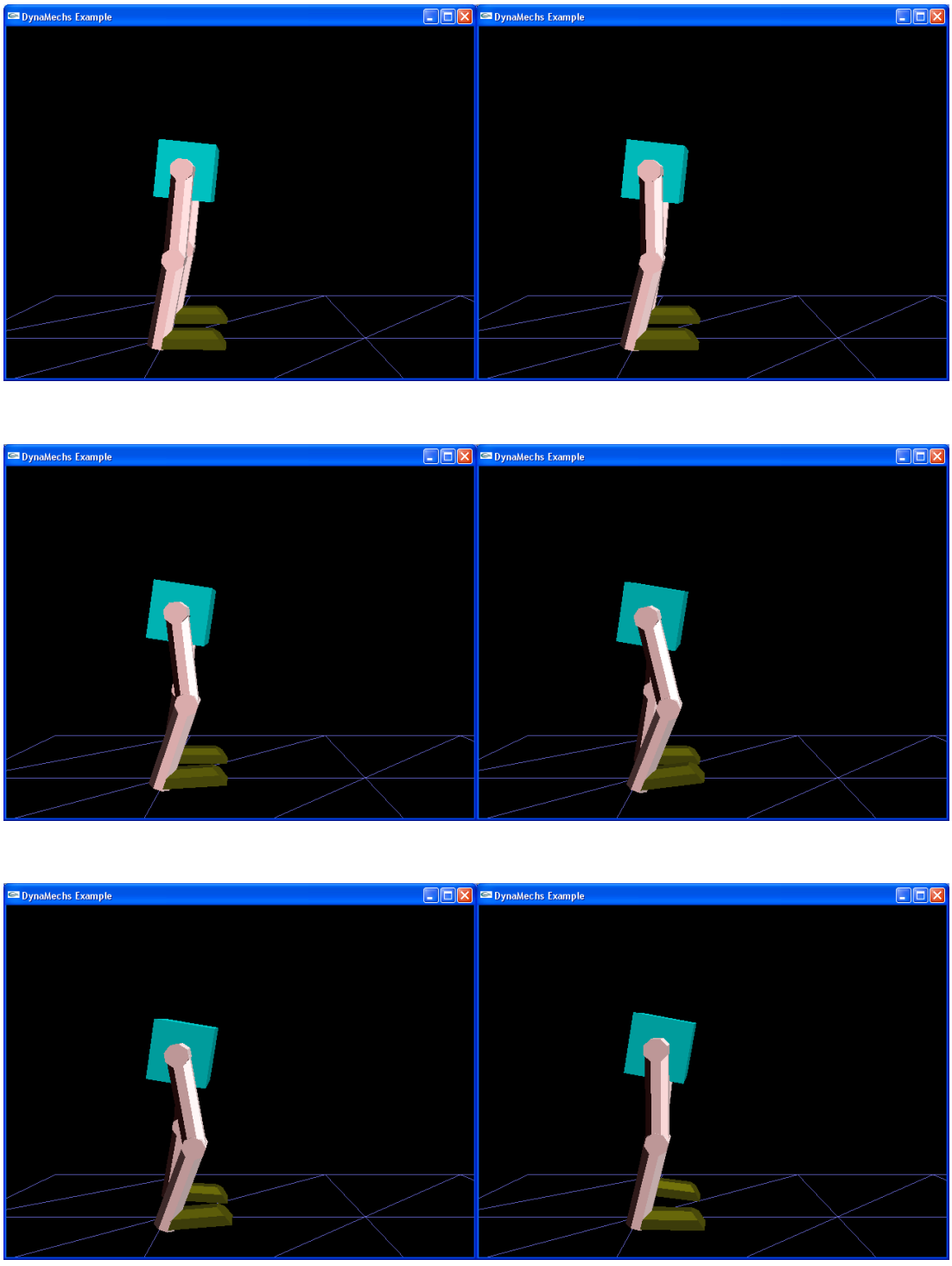

Figure A1.1: Sample biped gait. Images to be read from left to right, top to bottom.

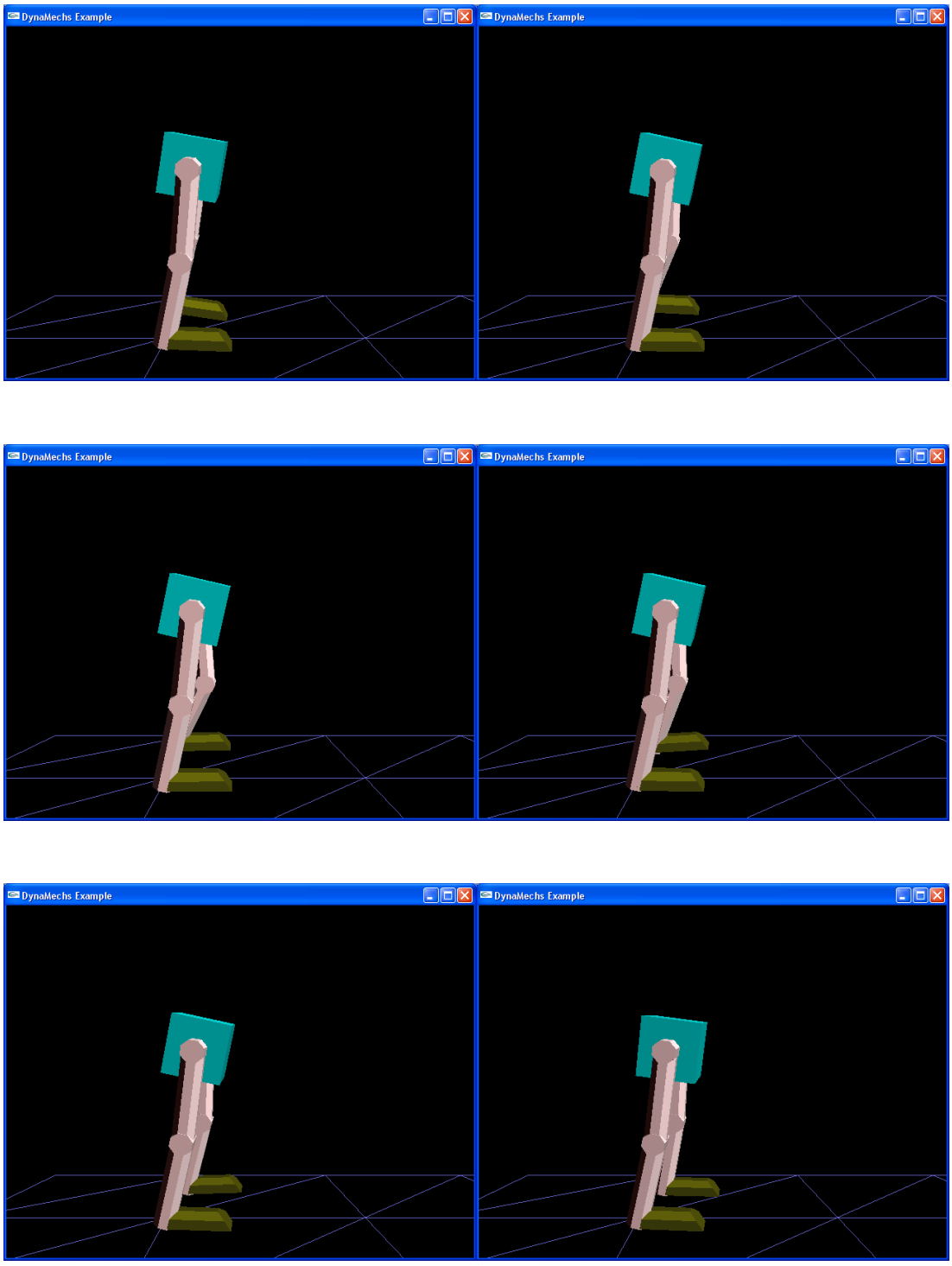

Figure A1.2: Sample biped gait. Images to be read from left to right, top to bottom.

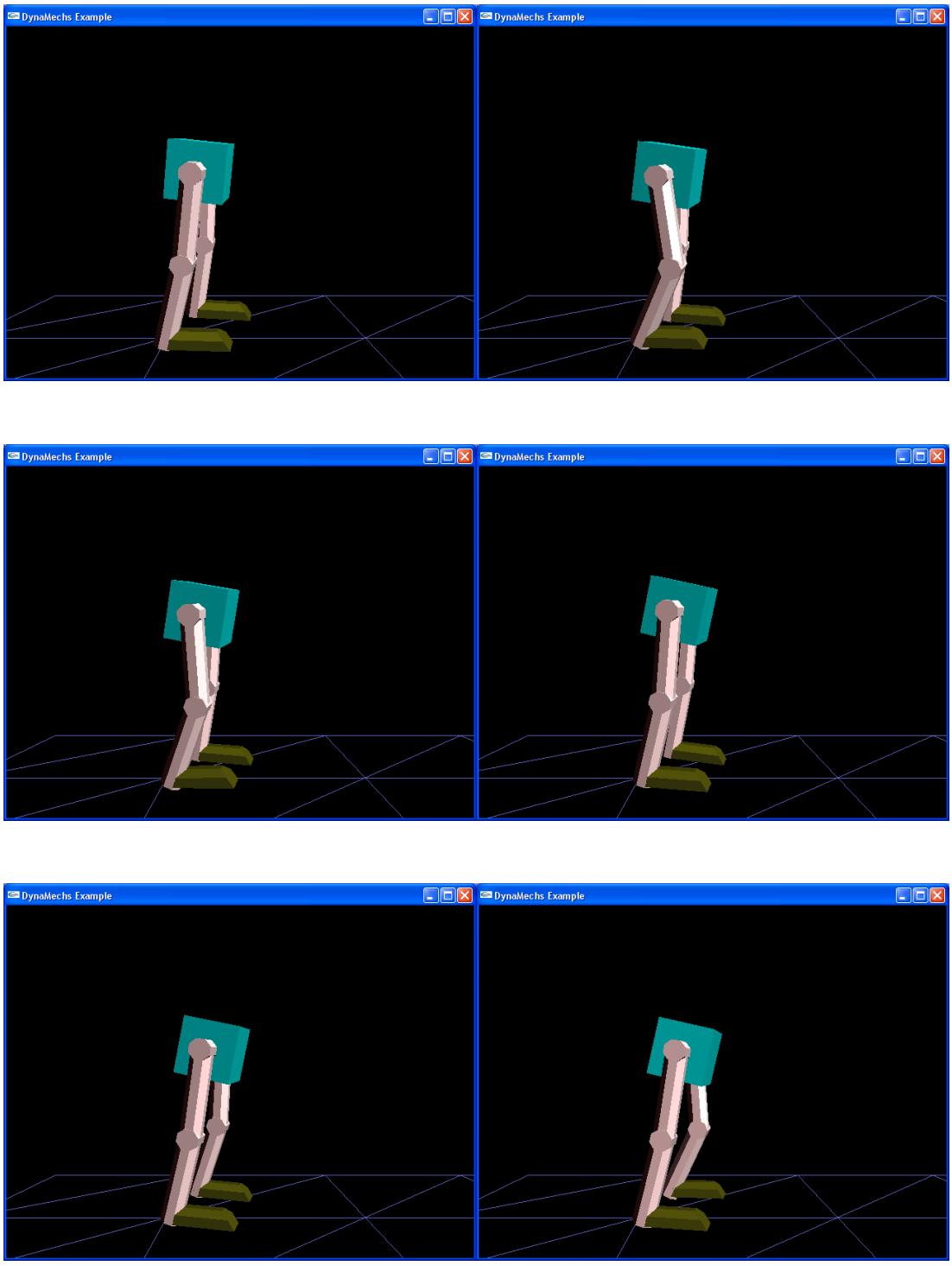

Figure A1.3: Sample biped gait. Images to be read from left to right, top to bottom.

Appendix 2. Sample Robot Model

This is the configuration file for the robot shown in Figure A2.1 This robot model was developed by Kevin Judd.

Figure A2.1: An example bipedal robot

```
# DynaMechs V 3.0 ascii

Articulation {
    Name              "cameron3"
    Graphics_Model    ""
    Position       7.3 -2.5 -1.0
    Orientation_Quat  0.0 0.0 0.0 1.0

    MobileBaseLink {
       Name "refmember"
       Graphics_Model       "models/obj_cube_center.xan"

       Mass                 0.5
       Inertia                    1.1   0.0  0.0
                            0.0   1.7  0.0
                            0.0   0.0  1.7
       Center_of_Gravity    0.0 0.0 0.0

       Number_of_Contact_Points   8
       Contact_Locations   -1.0  1.0  1.0
                            1.0  1.0  1.0
                           -1.0 -1.0  1.0
                            1.0 -1.0  1.0
                           -1.0  1.0 -1.0
                            1.0  1.0 -1.0
                           -1.0 -1.0 -1.0
                            1.0 -1.0 -1.0

       Position       0.0   10.0   7.3
       Orientation_Quat  -1.5 0.0 0 1.5
       Velocity       0.0 0.0 0.0 0.0 0.0 0.0
    }
    Branch {
       Branch {
           ZScrewTxLink {
              Name "LScrew"
              ZScrew_Parameters    1.4 0
           }
```

```
RevoluteLink {
   Name "LThigh"
    Graphics_Model     "models/obj_cylinder_x.xan"

    Mass                0.5
    Inertia                    0.1   0.0  0.0
                        0.0   1.7  0.0
                        0.0   0.0  1.7
    Center_of_Gravity   1.5 0.0 0.0

    Number_of_Contact_Points  1
   Contact_Locations          3.0  0.0  0.0

    MDH_Parameters                0.0 0 0.0 1.57
    Initial_Joint_Velocity        0.0
    Joint_Limits                  -0.8 2
   Joint_Limit_Spring_Constant  5000.0
   Joint_Limit_Damper_Constant  500.0
    Actuator_Type             0
    Joint_Friction            10.0
}

RevoluteLink {
   Name "LCalf"
    Graphics_Model     "models/obj_cylinder_x.xan"

    Mass                0.5
    Inertia                    0.1   0.0  0.0
                        0.0   1.7  0.0
                        0.0   0.0  1.7
    Center_of_Gravity   1.5 0.0 0.0

    Number_of_Contact_Points  1
   Contact_Locations          3.0  0.0  0.0

    MDH_Parameters                3.0 0.0 0.0  0.0
    Initial_Joint_Velocity        0.0
    Joint_Limits                  0.1 2
   Joint_Limit_Spring_Constant  5000.0
   Joint_Limit_Damper_Constant  500.0
    Actuator_Type             0
    Joint_Friction                10.0
}

RevoluteLink {
   Name "LFoot"
    Graphics_Model     "models/obj_half_cylinder_x.xan"

    Mass                0.5
    Inertia                    0.1   0.0  0.0
                        0.0   1.7  0.0
```

```
                          0.0   0.0  1.7
        Center_of_Gravity   1.5 0.0 0.0

        Number_of_Contact_Points  4
       Contact_Locations          2.3  0.2  0.0
                                  2.3 -0.2  0.0
                                  0.0  0.2  0.0
                                  0.0 -0.2  0.0

        MDH_Parameters                3.0 3.1416 0.0 1.7
        Initial_Joint_Velocity        0.0
        Joint_Limits                  0.3 2.9
       Joint_Limit_Spring_Constant  5000.0
       Joint_Limit_Damper_Constant  500.0

        Actuator_Type             0
        Joint_Friction                10.0
    }
  }

Branch {
  ZScrewTxLink {
      Name "RScrew"
      ZScrew_Parameters       -1.4 0
  }
  RevoluteLink {
      Name "RThigh"
      Graphics_Model  "models/obj_cylinder_x.xan"

      Mass                    0.5
      Inertia                 0.1   0.0  0.0
                              0.0   1.7  0.0
                              0.0   0.0  1.7
      Center_of_Gravity             1.5 0.0 0.0

      Number_of_Contact_Points      1
      Contact_Locations             3.0  0.0  0.0

      MDH_Parameters          0.0 0 0.0 1.57
      Initial_Joint_Velocity  0.0
      Joint_Limits           -0.9 2
  Joint_Limit_Spring_Constant  5000.0
  Joint_Limit_Damper_Constant  500.0

      Actuator_Type           0
      Joint_Friction          10.0
  }
  RevoluteLink {
      Name "RCalf"
      Graphics_Model_Index   "models/obj_cylinder_x.xan"

      Mass                    0.5
```

```
            Inertia                 0.1   0.0  0.0
                                    0.0   1.7  0.0
                                    0.0   0.0  1.7
            Center_of_Gravity              1.5 0.0 0.0

            Number_of_Contact_Points       1
            Contact_Locations              3.0  0.0  0.0

            MDH_Parameters          3.0 0.0 0.0  0.0
            Initial_Joint_Velocity  0.0
            Joint_Limits            0.1 2
              Joint_Limit_Spring_Constant  5000.0
              Joint_Limit_Damper_Constant  500.0

            Actuator_Type           0
            Joint_Friction          10.0
        }
        RevoluteLink {
            Name "RFoot"
            Graphics_Model_Index   "models/obj_half_cylinder_x.xan"

            Mass                    0.5
            Inertia                 0.1   0.0  0.0
                                    0.0   1.7  0.0
                                    0.0   0.0  1.7
            Center_of_Gravity              1.5 0.0 0.0

            Number_of_Contact_Points       4
            Contact_Locations              2.3  0.1  0.0
                                           2.3 -0.1  0.0
                                           0.0  0.1  0.0
                                           0.0 -0.1  0.0

            MDH_Parameters          3.0 3.1416 0.0 1.7
            Initial_Joint_Velocity  0.0
            Joint_Limits            0.3  2.9
              Joint_Limit_Spring_Constant  5000.0
              Joint_Limit_Damper_Constant  500.0

            Actuator_Type           0
            Joint_Friction         10.0
              }
     }
    }
}
```

Appendix 3. Software Overview

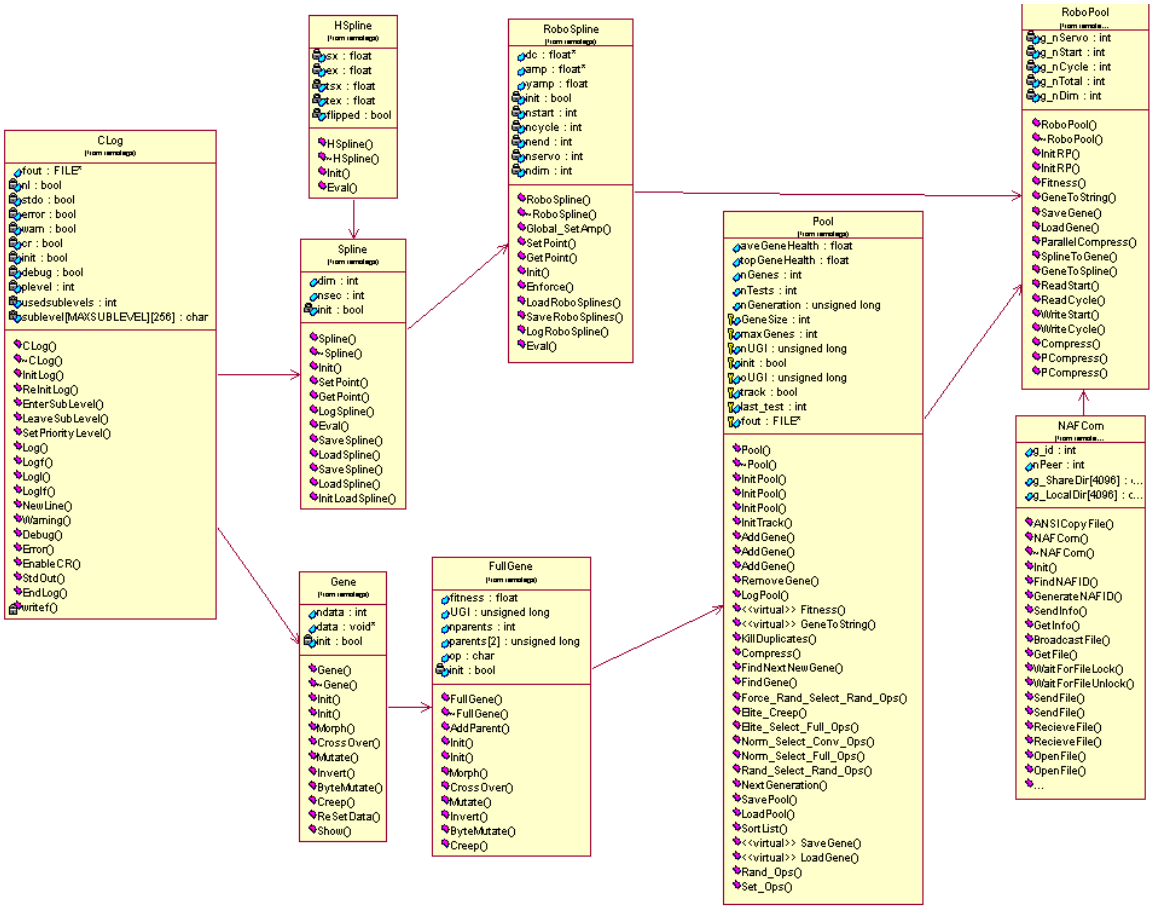

Figure A3.1 : UML Diagram of a subsection of the remotega program

The above figure provides a general description of the classes used in the construction of an application. The Clog class acts as an optional base class, which can be used for logging and debugging information . The controller used in the illustration above was the spline controller. Hence the spline, Hspline and RoboSpline classses were included. The Hspline provides the basic functionality behind a single cubic hermite spline. The Spline class emulates a multi point spline, and the RoboSpline contains the actual robot spline controller information. The base genetic algorithm functionality is encapsulated in the gene, fullgene and pool classes. The gene class contains the data structres required to manipulate a chromosome. The fullgene extends this functionality to enable tracking information within the GA. The pool class is a base class for a population. The RoboPool class is a specific instance of the

gene pool class and contains functionality to encode the spline controller (robospline) into a format usable in the genetic algorithm (pool). The NAFCom class provides the network file communications functionality required for parallel evolution. This class diagram was extracted from the remotega program, a standalone version of the architecture described in this thesis.