Evolving Neural Networks for Hexapod Leg Controllers

Gary B. Parker and Zhiyi Li
Computer Science
Connecticut College
New London, Connecticut 06320
parker@conncoll.edu, zli@conncoll.edu

Abstract -- The incremental evolution of neural networks to control hexapod robot locomotion can be separated into two main parts: the evolution of leg controllers that produce the cyclic action of single legs (leg cycles) and the evolution of the coordination of these individual leg controllers to produce a gait. In this paper, we use a genetic algorithm to do the first of these steps, to evolve the structure of an artificial neural network that produces leg cycles for a hexapod robot. The robot has 12 servo effectors; two per leg to produce horizontal and vertical movement. The servos are controlled by pulses that are provided by the leg's controller. A cycle of these pulses produces a leg cycle. With minimal restrictions on the structure of the neural network, a genetic algorithm was used to evolve in simulation the parameters of neurons and their connections. Neural networks were implemented on a BASIC Stamp II SX microcomputer and found to generate smooth leg cycles on the hexapod robot.

I. INTRODUCTION

Locomotion is a significant control issue for legged robots since the coordinated movement of all the legs is required to produce a good gait. Developing a system for learning gaits can eliminate the need for writing controllers for each specific robot, plus give the robot the ability to adapt to changes in its capabilities and/or environment. The problem of learning gaits can be divided into two parts, leg cycle learning and gait cycle learning, that can be implemented incrementally. In this way, near optimal controllers can be learned for each leg to produce effective cycles of movement that produce forward thrust. These controllers can then be coordinated to work together in producing a gait that maintains stability while propelling the robot forward at maximum speed. In this paper, we discuss a method for learning the controller that can produce a leg cycle for an individual leg. This is done using a genetic algorithm (GA) that learns the connections and their weights for a neural network (NN) controller. This controller, in addition to the ones for the other five legs, can be used in future work to produce a gait for a hexapod robot.

A number of researchers have used evolutionary computation to evolve controllers for legged robots [1,2,4,5,6,7,8,9]. Most related to the work presented in this paper is that of Beer and Gallagher [1] and Lewis et al [7].

Beer and Gallagher used genetic algorithms to develop neural network controllers for a hexapod robot [1]. The controller had three motor neurons (foot, backward-swing, and forward-swing), two internal unspecified neurons, and a single angle sensing neuron. The three motor neurons and two internal neurons were fully connected. They each had a threshold, a time constant parameter, and an input from the angle sensing neurons. The GA learned the weights of these eight input connections for the set of five neurons. After a leg cycle was learned for a generic leg, six copies of the leg controller were made. Each of the leg controllers was connected to its neighbor leg controllers. A GA was used to learn the connection weights between the leg NNs.

Lewis, Fagg, and Solidum used this staged evolution also [7]. In their model, the position of each of the two leg joints was driven by the state of a neuron. The two neurons controlling the leg were the leg swing neuron and the leg elevation neuron. The two neurons formed an oscillator, and the oscillators were mapped to a pulse width modulated signal that controlled the position of the motors. A GA was used to find the parameters of this two-neuron network. Initially, the NN started at random values but within several cycles the two neurons fell into an oscillatory pattern, with a phase difference of 90 degrees. Then a network of these oscillators was evolved by using a GA to coordinate the movements of the different legs.

In previous work to learn hexapod gaits, we used a cyclic genetic algorithm (CGA) [8] to learn the pulse sequences that were needed to directly control servomotors on each leg to generate leg cycles [9]. These pulse sequences took into consideration the peculiarities of the leg's capabilities. After each leg was trained, a GA was used to evolve gaits.

In this paper, the weights of a fully connected six neuron NN are learned. This NN takes in two sensor inputs and outputs a sequence of pulses to the two servomotors of the leg. This work is distinct from the work discussed previously. In the Beer and Gallagher model, the neural controller was designed at a macro level in which the output of neurons represented complete actions such as forward swing, backward swing and foot-down. The GA was used to find the appropriate associated weights and thresholds. Our approach is similar to this except that we have six neurons with two producing control pulses to the leg's two servomotors and two sensor inputs marking horizontal and vertical leg extremes. The outputs of the neurons are pulse widths that control the angular positions of each leg servo. In Lewis, Fagg, and Solidum's work, they specifically defined the controller of each leg as a simple two-neuron oscillator that generated outputs that were mapped into pulse widths and then sent to servos. The GA was used to learn the weights and thresholds.

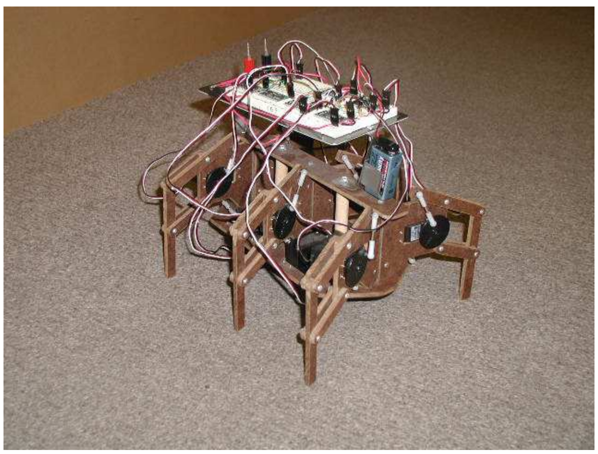

Fig. 1: ServoBot with seven BASIC Stamp II controllers; one for each leg and one to coordinate the legs.

In the work presented in this paper, the neural structure will evolve, including the connections, associated weights, and neuron thresholds. In addition, this work differs from these works in that the leg cycle learned is for a specific leg taking into account its particular capabilities. The resultant NN controller is then implemented on a BASIC Stamp II SX (sold by Parallax, Inc.). The output of the NN running on the BASIC Stamp is a cycle of pulses sent to the servos of the leg that produces a viable leg cycle.

II. SERVOBOT

The NN controller was designed to work on the ServoBot robot (Figure 1). It is an inexpensive hexapod robot that has two degrees of freedom per leg. Twelve servos, two per leg, provide thrust and vertical movement. The servomotors have three wires; one for power, one for ground, and a third provides control. They can be set to specific angular positions by providing a control pulse. This pulse should be repeated every 25 ms for the servo to maintain a constant position. The length of the pulse determines the position. Pulses from 0 to 3000 μs cover the full range of movement for each leg, although each servo is unique in its pulse to position ratios. Some may have a full down position at 20, on others it may be 80. There is the same variance in the full up position. In addition, the right and left side servos are mounted differently to ensure consistent mechanical capabilities, so in some cases the full down position is at a pulse length of 10 and in some cases it's at 2400.

Neuron	Pulse	X-Posit	Y-Posit
0	0	0	0
1	200	1	0
2	400	7	3
3	600	16	8
4	800	28	14
5	1000	41	21
6	1200	52	28
7	1400	63	34
8	1600	71	39
9	1800	75	43
10	2000	77	45
11	2200	76	48
12	2400	73	49
13	2600	73	49
14	2800	73	49
15	3000	73	49

Fig. 2: The neuron output with its corresponding pulse width. This pulse will affect the horizontal servo by moving the foot to the X position shown. This pulse sent to the vertical servo would move the leg's foot to the designated Y position.

By measuring the results on the actual robot, applied pulses can be converted into leg positions for both the vertical and horizontal servos. Figure 2 shows the relationship between pulse widths and the position they determine for a particular leg. These are measured capabilities of a robot's specific leg. The Neuron Output column will be explained in the next section. The Pulse Width column shows several possible pulse widths that can control the servo. The X and Y positions show the resultant leg position due to the movement of the horizontal and vertical servos associated with each pulse width. The full forward position is defined as 0 for X; the full downward position is defined as 0 for Y. A pulse with a width of 600 μs sent to the horizontal servo will drive it to an angle that will result in the leg moving to a position 16 mm back from the 0 position. The same pulse sent to the vertical servo will result in the leg lifting/lowering the foot to 8 mm off the ground.

The servo cannot move the leg fast enough to reach the desired position within one pulse if the differences in pulses are too much. This results in the fastest leg movement as the servo attempts to get to its desired position as soon as possible. Through experimentation, it was found that this fastest speed results in a maximum leg movement of 5mm in one pulse. For example, if the original pulses are (0, 0), corresponding to (horizontal servo, vertical servo), putting the leg at x, y position (0, 0) and the next pulses sent are (600, 0), the associated position (from Figure 2) should be (16, 0). However, because this exceeds the servo's capability, the leg will only move to (5, 0) in that one pulse.

Control for the ServoBot is provided by seven BASIC Stamp II SXs, one per leg and one working as the overall controller. Each leg's stamp should produce a sequence of pulses that will direct the position of its two servos. The central stamp controller will be used in future work to maintain the timing so all the neurons of the leg controller BASIC Stamps will fire at the same time.

III. THE NEURAL NETWORK CONTROLLER

The NN controller for the robot was designed to be implemented on seven BASIC Stamp II SX controllers. One of the BASIC Stamps will be the timer to control the coordinated execution of the other stamps. It will release a control pulse every 25 ms (the timing is predicated on the requirements of the servomotors that require pulses every 25ms to designate servo location). The six other BASIC Stamps will each control a leg. They will serve as nerve ganglia, with sensor inputs and actuator outputs, for that leg. These individual leg ganglia will be connected to each other to form a complete NN for locomotion control. Although the six BASIC stamps can do their computations in parallel, the computations within each stamp will be sequential. The appearance of parallel processing of every neuron will be achieved because the computations within each stamp will take place within the 25ms timer duration.

Each BASIC Stamp will simulate a neural ganglion where all of the neurons are firing simultaneously. This leg ganglion has six neurons (Figure 3). Each neuron can read the input from the two BASIC stamp pins that will be connected to sensors and the pins connected to other leg ganglia. In addition, each neuron has six input connections from each of the internal neurons (including itself), plus two thresholds. Two of the neurons are directly connected to the servomotors.

The coordinator controller will send a signal to all of the leg controllers every 25ms (we refer to this as an iteration). Upon receipt of this signal, internal computations using actual external connections and internal connections will be computed and the new internal values will be set as external signals are sent. A more detailed description of the NN follows.

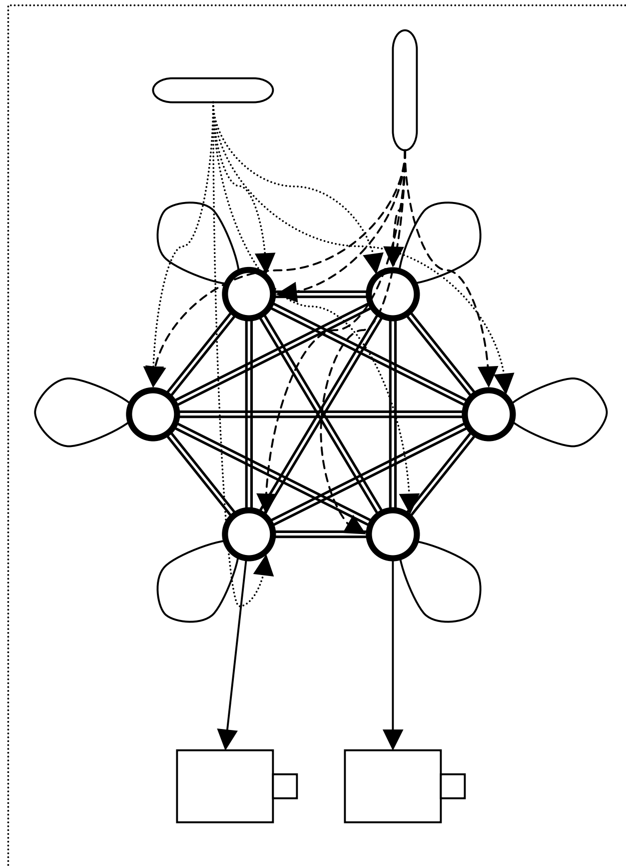

Fig. 3: The neural network controller for a single leg. The controller is made up of 6 neurons that each have 6 output (one that goes to each neuron including one that loops back to the sending neuron). This means that each neuron will also have 6 inputs from neurons. In addition, each neuron has an input from the horizontal sensor and one from the vertical sensor. The weights of all of these connections are learned by the genetic algorithm. One of the neurons is preset to send a control pulse to the horizontal servo and one is preset to send a control pulse to the vertical servo.

Sensors. The two sensors are simulated at this point. Sensor-1 monitors horizontal extreme positions and sensor-2 monitors vertical extreme positions. When the leg hits its full backward position, sensor-1 will fire 15 and keeps firing it until the leg hits its full forward position, where the sensor starts to fire 0. When the leg hits its highest position, sensor-2 will fire 15 until the leg hits its lowest position where the sensor starts to fire 0. The sensors are connected to all of the neurons. The connection weights, which are within the range [-15, 15], are left to the GA to find.

Neural connections. Due to memory limitations within a BASIC Stamp II SX, the number of neurons is limited to six (Neuron-0 to Neuron-5). Six should be sufficient for the GA to evolve freely. It is not known what role each neuron will play in the controlling process and there is no concern with whether a neuron is useful or not, just that enough neurons are made available to solve the problem. Each leg controller will have 6 neurons that are fully connected to each other and connected to the possible external connections. External connections are actual, through wiring between pins, and internal connections are simulated through variables holding the output. Each neuron's connections with other neurons and associated weights are left for the GA to find. It is possible that some neurons will not be used in the final result. The connection weights are within the range [-15, 15]. If it is 0, there is no connection in between; if it is positive, the effect from the source to target neuron is positive; if it is negative, the effect from the source to target neuron is negative.

Thresholds. Each neuron has two thresholds: threshold-low and threshold-high. During each iteration the NN changes its state, each neuron will accumulate the effects of those that are connected with it and of the sensors. This is called the accumulation value. The output of a neuron in each iteration is determined by its accumulation value and its two thresholds. The maximum output for a neuron is 15 and the minimum value is 0. If the accumulation of a neuron is smaller than threshold-low, it will fire a 0. If the accumulation of a neuron is more than threshold-high, it will fire a 15. If the accumulation of a neuron is between the two, a linear function will transform it to a value between 0 and 15 that will be fired. This linear function is shown in Equation 1.

$$Output = 15 \times \left(\frac{(A - T1)}{(T2 - T1)} \right) \tag{1}$$

`Output` is the output of a neuron; `A` is the accumulation of a neuron; `T1` is the lower threshold; `T2` is the higher threshold. In case threshold-high is smaller than threshold-low, the neuron will fire 15 when the accumulation is larger than threshold-low and fire 0 when the accumulation is smaller than threshold-low. The selection of the 0-15 range is to allow efficient storage in the BASIC Stamp II SX. With this range, we can use a nibble (4bits) to represent the output from a neuron.

Output. Neuron-0 is connected directly to the horizontal servo; Neuron-1 is connected directly to the vertical servo. At each iteration, outputs from Neuron-0 and Neuron-1, which are within the range [0, 15], will be mapped to associated pulse widths [0, 3000] by multiplying it by 200μs and sent to the leg's servos to control their angular position and subsequently set the linear positions of the leg.

Execution. The NN will start at an initial state (defined by the GA) and then it will run for 500 iterations, i.e. it will change its state 500 times. Each iteration is started by a signal from the coordinator controller (the 7th BASIC Stamp II SX). The signal will be sent every 25ms. After the NN gets this signal, it will evaluate itself according to its present state and calculate what the next neuron values should be.

IV. GENETIC ALGORITHM

A genetic algorithm (GA) was used to find the connection weights and the thresholds of the ganglia of neurons. The software package GENESIS5.0 [3] was used to perform the functions of the GA. A population of 80 chromosomes was randomly generated. Each chromosome represented a NN structure. The NN ran 500 iterations and thus generated a sequence of 500 pulses for each of the two servos. The two sequences were evaluated with a fitness assigned. The individual chromosome used was a series of numbers that describe the NN controller. It is shown in Figure 4.

```
((n0 t01 t02 w00 w01 w02 w03 w04 w05 ws10 ws20)
 (n1 t11 t12 w10 w11 w12 w13 w14 w15 ws11 ws21)
 (n2 t21 t22 w20 w21 w22 w23 w24 w25 ws12 ws22)
 (n3 t31 t32 w30 w31 w32 w33 w34 w35 ws13 ws23)
 (n4 t41 t42 w40 w41 w42 w43 w44 w45 ws14 ws24)
 (n5 t51 t52 w50 w51 w52 w53 w54 w55 ws15 ws25))
```

Fig. 4: The chromosome used to represent a neural network. `ni` is the original value (value between 0 and 15) of the ith neuron. `ti1` is the lower threshold (value between -400 and 400) of the ith neuron. `ti2` is the higher threshold (value between -400 and 400) of the ith neuron. `wij` is the connection weight (value between -15 and 15) from the ith neuron to the jth neuron. `ws1i` is the connection weight (value between -15 and 15) from sensor-1 to the ith neuron. `ws2i` is the connection weight (value between -15 and 15) from sensor-2 to the ith neuron.

The general GA functions were performed by the GENESIS package, but a fitness function was provided to run the program. Fitnesses consisted of three factors: forward movement, number of times raising the leg, and drag generated. Forward movement was calculated by the movement generated when the leg was on the ground. The number of times the leg was raised and lowered was penalized because it wasted energy and reduced the effect of the forward movement generated. The drag generated was a penalty due to a leg staying on the ground with its full backward position, which would just cause drag.

V. IMPLEMENTATION ON THE BASIC STAMP

Tests were done using five different (randomly generated) starting populations. Learning was done in simulation with the final best solution tested on a leg of the actual robot. Figure 5 shows the learning curve with the average of the five separate tests. As can be observed, the GA quickly learned and by about 60 generations all of the trials resulted in a leg cycle that could produce sustained forward movement. In the remaining generations, the GA continued to refine the solution to produce a better leg cycle. Figure 6 shows the chromosome of the individual that resulted in the fittest NN controller (compare with Figure 4 to see what each field represents).

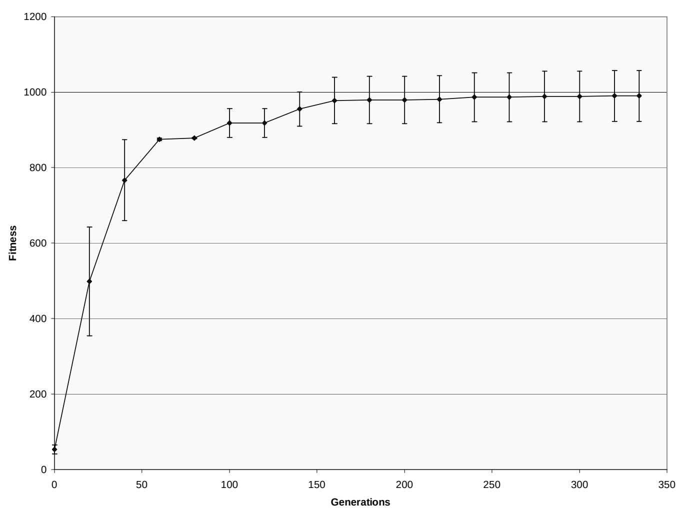

Fig. 5: Single leg training, the average populations (the error bars show standard error) of the five starting. The fitness of the best individual at every 20 generations of training is shown.

The artificial NN, which was found to be the fittest in simulation, was used for implemented on the hardware. Tests were run using the BASIC Stamp code development software, in the debug mode, to observe the pulse outputs. These were determined to be reasonable so it was used to attempt implementation on the BASIC Stamp. Initially the standard BASIC Stamp II was used. The weights were stored in the EPROM, an array of 6 nibbles was used to store the values of the neurons, and an additional array of 6 nibbles was used to store the resultant values of the 6 neurons (values ranged from 0 to 15).

The program used subprograms to sequentially compute the new values for each of the neurons. During these computations the old neuron values were held constant in the current neuron value array. After the computations were complete, the new values were copied over the old and pulse values were sent for the two servos.

This program was downloaded/executed and the results observed. In these initial tests done using the standard BASIC Stamp II, its processing speed was observed to be too slow to deliver timely control pulses to the servos. The movements of the leg looked correct, but the pulse signals sent to the servos were spaced at long intervals that resulted in very slow and rough leg movement. Implementation was switched to the BASIC Stamp II SX and it was determined that this microprocessor could make the calculations fast enough to deliver pulses to the servos in time to produce smooth leg movement.

```
((4 -186  125  14 -12  -8  3  6 -3 -1 -10)
(10 167  398 -14   0  12 -14  1  0  6 -13)
(11 104   15   1  -2  15 -14  3  0  5  13)
(5  173 -362 -11  -8 -14  -5 -9  7 -13  7)
(10  88  274 -12  -8   2  -1 14  0  -5  1)
(3  -79 -135  -8  15  -7   8  3 -4  6  2))
```

Fig. 6: The resultant NN chromosome that produced the fittest controller for the leg.

The pulses produced by the NN controller resulted in a leg cycle that used the full horizontal movement doing thrusts. The return was done immediately upon reaching the full back position with an up and down phase that put the leg back on the ground in position for the next thrust. Figure 7 shows the movement of the actual leg by a plot of the X and Y positions of the foot (the full up, down, forward, and back positions were measured; the slopes were approximated).

The leg starts out moving back from the 0 position to 73 mm back while the foot stays on the ground (this takes 15 pulses). When the leg gets to the full back position, the legs starts moving forward and at the same time it starts lifting the foot (this can be seen at pulse 15). This lift continues for about eight pulses and then the leg quickly lowers the foot back down to the ground. At approximately this point, the NN directs the horizontal servo to start moving the leg back again. There is an instant where the leg is moving back while the foot is not all the way on the ground. This causes no problem because the horizontal movement at this point is minimal so minimal thrust is lost and when the foot hits the ground the leg produces immediate thrust. The cycle continues with the foot moving vertically from 0 to 23 mm and horizontally from 14 to 73 mm. The speed of the foot is fastest in between the extremes as it slows down to reverse direction.

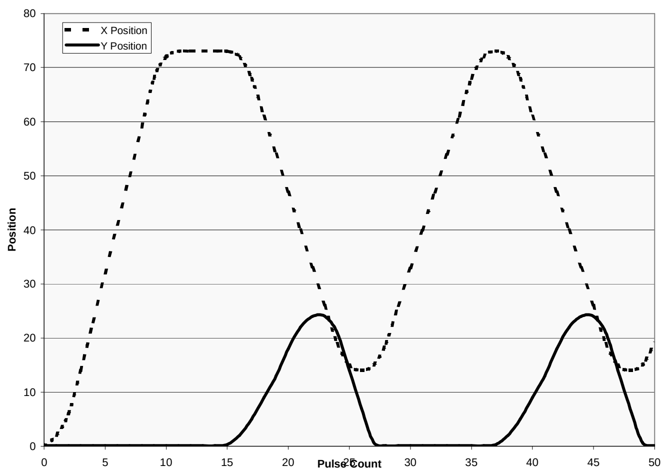

Fig. 7: Observed movement of the leg when the best NN controller was downloaded and ran on the BASIC Stamp II SX. The graph shows the X and Y position of the foot as the leg moves in the first 50 pulses. After the initial 15 pulses, a cycle of 22 pulses starts and is continually repeated.

VI. CONCLUSIONS

NN controllers for the BASIC Stamp II SX are capable of producing useable leg cycles for the ServoBot robot. A GA can effectively evolve the structure of these NN controllers. The resultant NN controllers are specialized for the task and the learning system will have the flexibility to adapt to changes in the environment or robot capabilities. Tests on the SX confirmed the viability of this method as the produced leg cycles worked well on the actual robot. Future work will involve the generation of NN controllers for each of the legs followed by the evolution of a central controller. Wires between the I/O pins of the BASIC Stamp will connect the leg controllers to each other. Alternatively, the six NN ganglia can be co-evolved to learn coordination with no central controller (except to send timing signals for synchronization).

VII. REFERENCES

[1] Beer, R. D. and Gallagher, J. C. (1992). "Evolving Dynamic Neural Networks for Adaptive Behavior." *Adaptive Behavior*, vol. 1, pp. 91-122.

[2] Fujii, A., Ishiguro, A., Otsu, K., Uchikawa, Y., Aoki, T., and Eggenberger, P. (2000). "Evolutionary Creation of an Adaptive Controller for a Legged-Robot: A Dynamically-Rearranging Neural Network Approach." *Proceedings of the International Symposium on Adaptive Motion of Animals and Machines.*

[3] Grefenstette, J. J. GENESIS5.0 Copyright © 1990.

[4] Gruau, F. (1995). "Automatic Definition of Modular Neural Networks." *Adaptive Behavior*, vol. 3, num. 2, pp. 151-183.

[5] Hornby, G., Fujita M., Takamura S., Yamamoto T., and Hana O. (1999). "Autonomous Evolution of Gaits with the Sony Quadruped Robot." *Proceedings of the Genetic and Evolutionary Computation Conference*.,pp. 1297-1304.

[6] Kodjabachian, J. and Meyer, J-A. (1998). "Evolution and Development of Modular Control Architectures for 1-d Locomotion in Six-Legged Animats." *Connection Science*, vol. 10, pp. 211-254.

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

[8] Parker, G. and Rawlins, G. (1996). "Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots." *Proceedings of the World Automation Congress (WAC'96), Volume 3, Robotic and Manufacturing Systems*, pp. 617-622.

[9] Parker, G. (2000). "Evolving Leg Cycles to Produce Hexapod Gaits." *Proceedings of the World Automation Congress (WAC'00), Volume 10, Robotic and Manufacturing Systems*, pp. 250-255.