The Incremental Evolution of Gaits for Hexapod Robots

Gary B. Parker

Computer Science
Connecticut College
New London, CT 06320
parker@conncoll.edu

Abstract

Gait control programs for hexapod robots are learned by incremental evolution. The first increment is used to learn the activations required to generate a single leg cycle. At this level the control program is required to produce the proper sequence of pulses needed to generate smooth movement by the servos. The learning program needs to take into account the peculiarities of the servo, its mounting and the capabilities of the leg. The second increment of the learning process is used to learn the best combination of individual leg cycles to produce a gait. This part requires the learning system to choose the best leg cycles for each leg and to coordinate their movement. In this paper, we describe an application of this method to learn gaits for an actual hexapod robot. A cyclic genetic algorithm is used to learn efficient gait cycles for each leg. A genetic algorithm is used to combine these leg cycles in such a way that coordinated gaits result. Tests are conducted on the actual robot to confirm the method's viability.

1 INTRODUCTION

The generation of gaits is important for the effective use of hexapod robots. Proper gaits are needed to ensure that the robot moves quickly and efficiently. Gaits need to be custom designed specifically for the individual robot to make the best use of its capabilities. There are two main parts to gait generation; the cyclic action of the individual legs and the coordination of all the legs to make effective use of their cycles. These can be learned together by finding the sequence of concurrent movements required by all the actuators as was done in previous work [3,4,5]. Or, they can be learned separately. Learning together greatly increases the complexity so details are often lost in the abstraction necessary to keep the computations within reason. This method can produce reasonable gaits that can operate on simpler controllers, but since some detail is lost, they cannot fully exploit the capabilities of the robot. Learning the leg cycles separate from their coordination allows the system to better use each leg as long as the controllers are complex enough to handle the increased details.

Individual leg learning can take into account the capabilities of the actuators and movement constraints of individual legs. In our work, we use a robot that has servos for actuators. These servos require a pulse to designate their desired position. The pulse length for each position is distinct for each servo and is dependent on its placement during installation. A single pulse does not guarantee proper positioning since it may be asking the servo to move further then it can in the time between pulses. A sequence of pulses with small changes in pulse length is required to get rate control. This sequence of control signals needs to be repeated to get the cycle of activations required to produce a cycle of movement for the leg.

Since evolutionary computation (EC) is well suited for adapting a solution to the peculiarities of a problem, some form of EC would work well in learning what signals are needed for the leg cycle. The difficulty comes in that most forms of evolutionary computation are not naturally equipped to handle the cyclic nature of these leg cycles. One exception is with genetic programming, which can be used to evolve programs and programs can have loops. Graham Spencer [6] had some success in generating programs for hexapod gaits using genetic programming. His programs worked concurrently on all the actuators to produce gaits for hexapod robots. His programs, tested only on robot simulations, resulted in gaits that maintained sustained forward movement but could not obtain the optimal tripod gait.

Randall Beer and John Gallagher [1,2] used genetic algorithms (GAs) to develop neural network controllers for a simulated hexapod robot. In this work, the structure of the neural networks (NNs) was pre-defined and the GA learned the weights required to generate gaits. This work makes more of a division between the leg cycles and the coordination of legs. NN structures are separately defined for the leg cycles and the coordinators, but the weights are learned concurrently. The nature of the leg cycles are somewhat defined by the structure of the NNs and further learned by the GA. The coordination is controlled by central NNs with defined structures; the weights of these are learned at the same time as the leg NN weights.

In previous work [3,5], we used Cyclic Genetic Algorithms (CGAs) to generate the sequence of primitive instructions that produced a gait. CGAs were developed to allow for the representation of a cycle of actions in the chromosome. They differ from the standard GA in that the chromosome is in the form of a circle with two tails. The tails of the CGA chromosome are provided to allow for pre and post-cycle procedures. They provide a means for completing tasks before and after entering the cycle. For gait sequence generation, the pre-cycle can position the legs in a ready to walk posture and the post-cycle can return the robot to a stable at rest posture. In our application, we used only the pre-cycle tail. The CGA genes can be one of several possibilities. They can be as simple as normal genes that represent traits of the individual or they can be as complicated as cyclic sub-chromosomes that can be trained separately by a CGA. For our purposes, the genes represent tasks that are to be completed in a set amount of time. The trained chromosome will contain the cycle of servo control pulses that will be continually repeated by the leg's controller to produce a leg cycle.

Tests showed that CGAs could produce tripod gaits on robot simulations that were transferable to an actual autonomous hexapod robot [4]. This was accomplished by creating a model with specific information taken from an individual robot. The CGA used this model to develop an optimal gait that was specific to the robot's capabilities. This gait was subsequently downloaded into the actual robot where its performance was confirmed to correspond to the performance of the model. The primitive instructions used in these experiments were not designed to take advantage of the full capabilities of the servo motor actuators. Each servo had 2 possible states; either full forward or full back for horizontal servos or full up or full down for vertical servos. Each servo was given a control pulse that would drive it to the extreme. This was necessary to accommodate the limited capabilities of the single BASIC Stamp II controller.

In this paper, we use incremental evolution to learn control for 7 controllers. One controller is used to coordinate the other 6, which are each used to control a leg. These 6 additional controllers allow the system to take advantage of the capabilities of the servos. Each leg controller controls that leg's vertical and horizontal servo. Cycles of pulses are learned using a CGA that produces individual leg cycles optimizing for time on the ground and forward movement. These individual leg cycles are then combined, using a standard genetic algorithm to produce gaits for the robot. Tests in simulation and on the actual robot confirm the viability of this method for producing gaits.

2 THE ROBOT

The robot used was the ServoBot, which is a hexapod robot that has two degrees of freedom per leg. Twelve servos, two per leg, provide thrust and vertical movement. 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 20 to 2400 microseconds 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 20 and in some cases it's at 2400.

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. Varying speeds of movement can be attained by incrementally changing the pulse lengths. For example, moving a leg using consecutive pulse lengths of 40, 45, 50, etc. will move the leg at a slower speed than 40, 50, 60, etc., unless, of course, the increments are already more than the servo's capability. Consecutive pulses of 40, 240, 440, etc. would probably result in the same speed as the consecutive pulses of 40, 340, 640, etc.

Control was provided by BASIC Stamp IIs, one per leg and one working as the overall controller. Each leg's stamp could take in a sequence of pulses that indicated the position of its two servos. The central stamp controller told each leg stamp when to start its sequence and if needed, when to cut short one cycle to start another in order to maintain leg coordination.

3 THE FIRST INCREMENT: EVOLVING LEG CYCLES

In order to produce leg cycles, each stamp needs a sequence of pulses to continually position its servos. This sequence must by variable in length to accommodate the differing capabilities of each leg and its servos. Fixed length chromosomes offer distinct advantages when using CGAs since like areas of each chromosome are more likely to correspond to similar tasks. In order to formulate the problem in such a way as to be able to use a fixed length chromosome, some observations of a leg cycle had to be made. Pulses within 20 microseconds of each other result in positions that are only slightly distinguishable from each other (usually within 1 mm). This level of position accuracy is sufficient for our problem, so we can represent all pulses from 0 to 2400 by the numbers 0 to 120 considering each to be in increments of 20 microsecond pulses. This allows us to use a seven bit number to represent each pulse. It takes 14 bits to represent pulses for both servos.

Smooth movement is required by the horizontal servo, especially while on the ground. A sequence of pulses such as 100, 120, 140, 160 would move the leg smoothly from the position corresponding to 100 to the position corresponding to 160. The sequence 100, 110, 150, 160 would result in the same final position, but the movement would not be as smooth. The chromosome representation needed to be such that smooth movement would be possible for horizontal movement, but was not needed for vertical movement since vertical movement does not affect the smoothness of the robot's movement.

$$((R_1\ HP_1\ VP_1)\ (R_2\ HP_2\ VP_2)\ (R_3\ HP_3\ VP_3)\ (R_4\ HP_4\ VP_4)\ (R_5\ HP_5\ VP_5)\ \ldots\ (R_8\ HP_8\ VP_8))$$

Figure 1: Leg cycle chromosome. Each gene of the chromosome was made up of three parts: repetitions, horizontal pulse, and vertical pulse.

In order to accommodate these considerations, the chromosome representation shown in Figure 1 was used. The chromosome was made up of 8 genes. Each gene consisted of 3 parts. The first was called the *repetitions*, the second was the *horizontal pulse*, and the third was the *vertical pulse*. The *horizontal pulse* and *vertical pulse* numbers were each multiplied by 20 microseconds to calculate the actual pulse width sent to the servo. The effect of the *repetitions* was different on the two types of pulse. For the *horizontal pulse* the repetitions number was used to calculate the increments required to move from the servo's last pulse length to the new pulse length. The following formula was used:

pulse increment = (horizontal pulse - previous horizontal pulse) / repetitions

This *pulse increment* was then added for *repetitions* number of consecutive pulses until the end servo pulse was at *horizontal pulse*. For example, if the *previous horizontal pulse* was 40 and the gene was (5, 60, 100) then the following pulses would be sent to the horizontal servo over the next 5 inputs : 44, 48, 52, 56, 60. *Repetitions* effected the *vertical pulses* only by telling the controller how many times to repeat this *vertical pulse*. The extra computation was not required since smoothness was only a factor for horizontal movement.

The contents of the chromosome representation were used directly by the BASIC Stamp II and upon execution it would do the calculations required to direct its two servos. An example of the resultant sequence of pulses that would be produced is shown for a shortened chromosome in Figure 2.

Genes	Horizontal Pulses	Vertical Pulses
(4 25 127)	100	127
	75	127
	50	127
	25	127
(2 55 43)	40	43
	55	43
(5 125 38)	69	38
	83	38
	97	38
	111	38
	125	38

Figure 2: Sequence of pulses resulting from example genes. The 125 from the last gene is used to calculate the increments (-25) from 125 to 25. The first of this is added to calculate the first pulse. As can be observed, the pulse of 125 is again reached and the cycle continues with smooth horizontal movements.

3.1 LEG MODEL

Each leg was represented by a simple data structure that held the information required to produce a leg cycle. Each servo's maximum throw positions were stored as x, y, coordinates. The horizontal servo's full forward position was defined as x = 0, the full back position was x = the measured number of millimeters distance from the full forward. The vertical servo had a y = 0 if it rested on the ground when all the legs were full down and the max up was y = the millimeters off the ground when the leg was fully lifted. Along with these positions the pulse required to attain each was recorded. The model data structure also included a lookup table for each servo. This table listed the corresponding leg position of 13 different pulse lengths (1,200, 400,…2400). These figures were attained by applying consistent pulses to each servo and measuring the leg's response. The final data kept in the model was the current position and pulse of each servo.

3.2 TRAINING

Evolution of a leg cycle started by taking accurate measurements of the leg's capabilities. This information was fed into the model data structure used for training. A population of 64 chromosomes (each representing a leg cycle) was randomly generated and trained for 500 generations on the model of the robot. Fitness was calculated using three factors: forward movement, down count, and smoothness. Forward movement was calculated by determining the movement generated while the leg was on the ground. To attain the maximum forward movement, the leg should be on the ground throughout the length of its effective throw. The effective throw is usually less than the full throw. As the leg reaches its extremes of movement, the distance moved per pulse reduces significantly, so in the optimal solution the leg is repositioned before it reaches its full extreme. Another facet of the leg fitness is the down count. This factor gives more fitness to leg cycles where the leg is on the ground for a high proportion of the time. A third contributor to fitness is smoothness. This is calculated for movement on the ground. Leg cycles where the horizontal movement over the ground is consistent score higher smoothness. These three fitness indicators were added together to get the total fitness. The fitness for each chromosome was used to stochastically select individuals to produce each new population.

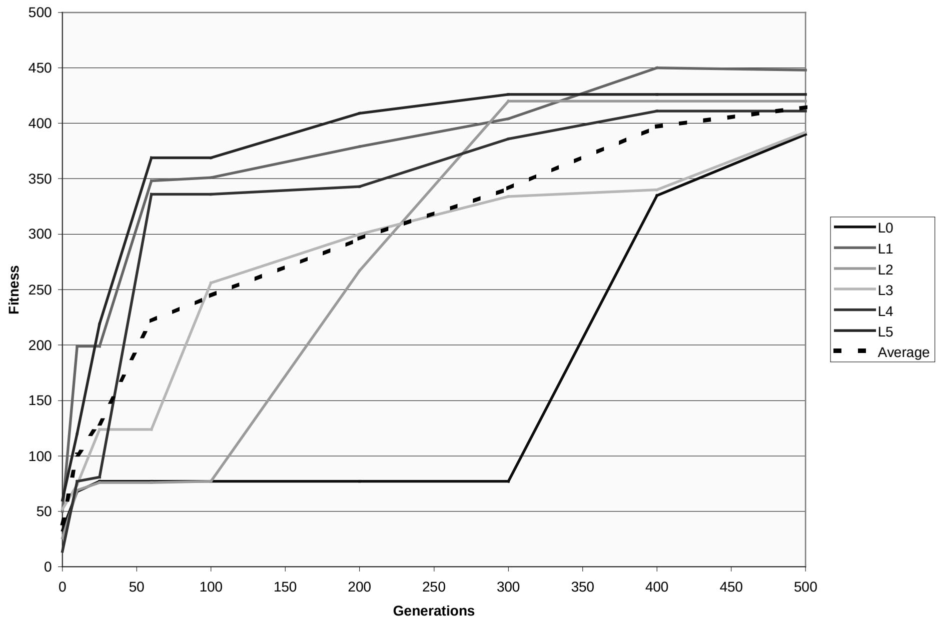

Figure 3: Single leg training for each leg.

Crossover was accomplished by randomly picking corresponding spots in the two selected parents. In the pre-cycle tail, a single point in both chromosomes was picked. In the cyclic section, since it could be considered a circle, crossover was performed at two points. The effect was to swap sections within the circle. An alternate type of crossover was a gene-by-gene crossover that performs crossover in each of the corresponding genes of the two chromosomes. Crosses could happen between the individual members of the list or within the bits of the specific numbers in the list. There were two types of mutation used and selected randomly after each recombination. In one, each gene had a random chance of being replaced by a new completely random gene. In the other, each part of the gene had a random chance of having one of its bits flipped.

Gene-by-Gene Evaluation, a genetic operator peculiar to CGAs, was used to clean up the chromosome by randomly picking one or two individuals from the population on each set of trails and examining each gene one at a time. Genes were evaluated move-by-move by comparing the previous move fitness to the present. Genes that performed poorly in their current position were eliminated. Genes that were good in the execution of their early repetitions and subsequently dropped in the later repetitions were modified by reducing their repetitions. Genes that had zero repetitions were moved out so that only active genes were at the start of the cyclic section.

3.3 RESULTS

Training was done for 500 generations with the fittest individual chromosome saved at 0, 10, 25, 60, 100, 200, 300, 400, and 500 generations. The results of this training, done for each leg, is shown in Figure 3. Both the optimal length (number of pulses in the cycle) and content of the cycle had to be learned. Each solid line represents a leg. The dashed line is the average. Three of the 6 legs learned quickly. One of the legs was stuck for some time, with a suboptimal length, which precluded it from further growth until it evolved to a different length. At this time it also improved rapidly. The optimal lengths found for the six legs varied from 29 to 36 pulses per cycle.

Training was repeated, in preparation for the gait training discussed in the next section, but this time an additional fitness calculator was used. *Desired length* reduced the fitness if the chromosome's length was different than a predesignated desired length. A second test was performed using 5 randomly generated populations, but this time the *desired length* factor was included. The length used was 36 pulses, which was the maximum optimal length found in the previous test. The results of this test are shown in Figure 4. With pressure to conform to a specified near optimal length, all six legs grew quickly in their fitness.

The resultant leg cycles were downloaded and observed on the actual robot where they appeared to produce efficient, useable leg cycles. No mechanism was constructed to test individual legs, so actual quantitative tests were not possible until the individual leg cycles were used together to form a gait.

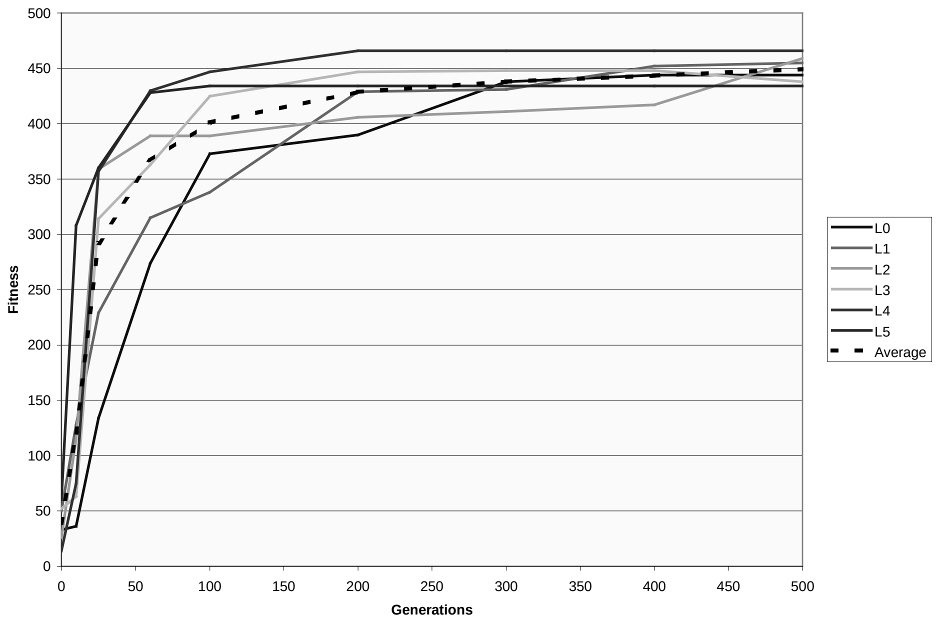

Figure 4: Single leg training with a desired length specified.

4 THE SECOND INCREMENT: EVOLVING GAITS FROM THE LEG CYCLES

A hexapod gait can be looked at as the coordination of 6 legs with each leg performing its own cycle. The proper combination of the six correct leg cycles will produce the desired gait. In addition to finding the best gait for the ServoBot using optimal leg cycles, this task also shows how to combine several cycles to form a single cyclic behavior.

4.1 EVOLVING FIXED LENGTH LEG CYCLES

Section 3 discussed the evolution of gait cycles that were optimal when they were at a desired length of 36. This length was chosen since it was the longest of the 6 no-length-restriction optimal leg cycles found previously. A set of leg cycles using a range of desired lengths would be needed to produce a gait. The gait learning algorithm would be able to choose leg cycles from anywhere in this range for each leg to come up with the proper coordination of legs. The no-length optimal should be in the center of this range but there was more likelihood that longer length leg cycles would be of use in further experimentation so the longest gait cycle length found in the no-length tests was chosen to be the middle desire length.

Each leg trained for 500 generations to learn optimal leg cycle with a desired length of 36. This population was then used to learn gait cycles with desired lengths from 21 to 52. Starting with the 36 length population, the desired length was changed to 35 and training continued for 200 generations. This continued down to a desired length of 21. Similarly, training up to 52 was done starting from the 36 length population. These learned leg cycles were stored in 6 files, which were called up when gait training began.

Gait training was done using a standard GA. The chromosome (Figure 5) was made up of 7 parts. The gait cycle length (GCL) represented the number of pulses in each gait cycle. Information for each leg included its leg cycle length (LCL) and start time (START). Each of these values is described in the next section.

```
(GCL
 (LCL START)
 (LCL START)
 (LCL START)
 (LCL START)
 (LCL START)
 (LCL START))
```

Figure 5: Chromosome used for gait training.

4.2 ROBOT MODEL

The single stamp that acted as the central controller was to coordinate the individual leg cycles. It needed to know the length (in pulses) of the gait cycle and which leg cycles to use for each leg. In addition, it needed the start time for each gait cycle. This was where the coordination took place. Upon execution the controller program would count through the total number of pulses 0, 1, 2, 3…. When the start time for each leg was researched, its leg cycle began. The central controller ensured that all the stamps executed their pulses together. When the gait cycle length was reached, the count started again at 0. When each leg's start number was reached they begin their cycle again. To simulate the effect of this on the robot, each of the leg cycles was run separately for the number of designated pulses used for training (500 in this case). They were then considered to be running simultaneously in a simulator that would determine at each pulse what the result of the 6 leg pulses would be.

4.3 TRAINING

A population of 64 randomly generated chromosomes was produced to start training, which was done for 500 generations. Each individual's fitness was calculated by determining the effect of the 6 leg-cycles running simultaneously as specified by the gait cycle chromosome. In addition to calculating the fitness produced by the legs, additional factors such as balance and drag were introduced. Balance was a determination of the robot's stability. Drag was used to penalize the fitness of the gait when the legs were on the ground but not producing thrust. Using these fitnesses, individuals were stochastically selected to be the parents of the next generation. Crossover and mutation were done both at the gene level and at the bit level as described in section 3.2.

4.4 RESULTS

The best individual at 0, 10, 25, 60, 100, 200, 300, 400, & 500 generations was stored. The results on the robot model are shown in Figure 6. Graphs of the fitness growth of the 5 distinct starting populations along with their average (dashed line) is shown. There are three things to note from this graph. The start fitness at generation 0, in most cases, is fairly high. This is because all the legs are already moving in a near optimal cycle; they just need to be coordinated. The GA quickly learns adequate coordination by 100 generations. After that, the GA works to improve this solution to find the optimal leg cycle lengths and start spots for each leg. In all 5 cases, near optimal tripod gaits are produced.

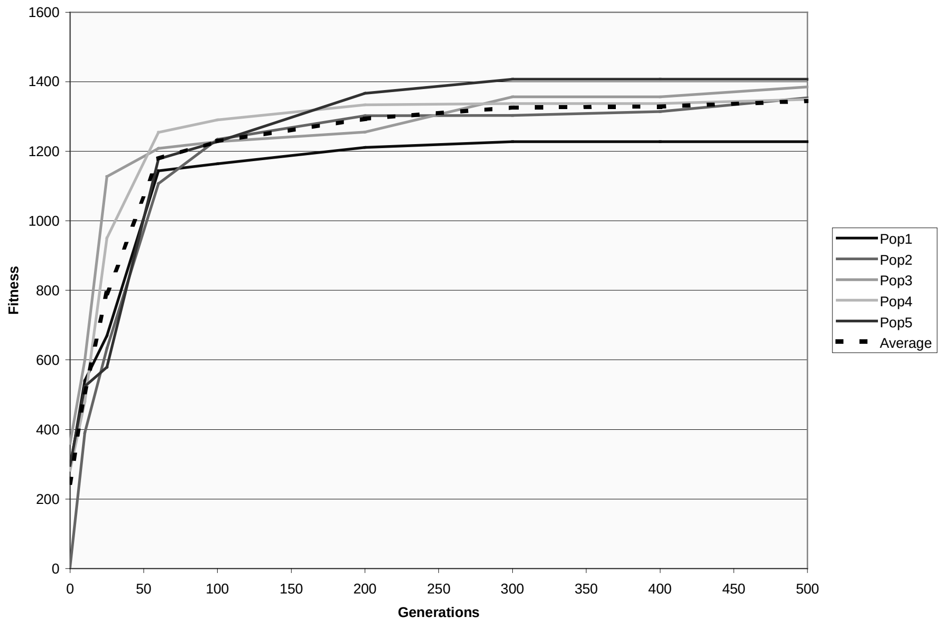

Figure 6: The results on a model of GA training to coordinate the legs.

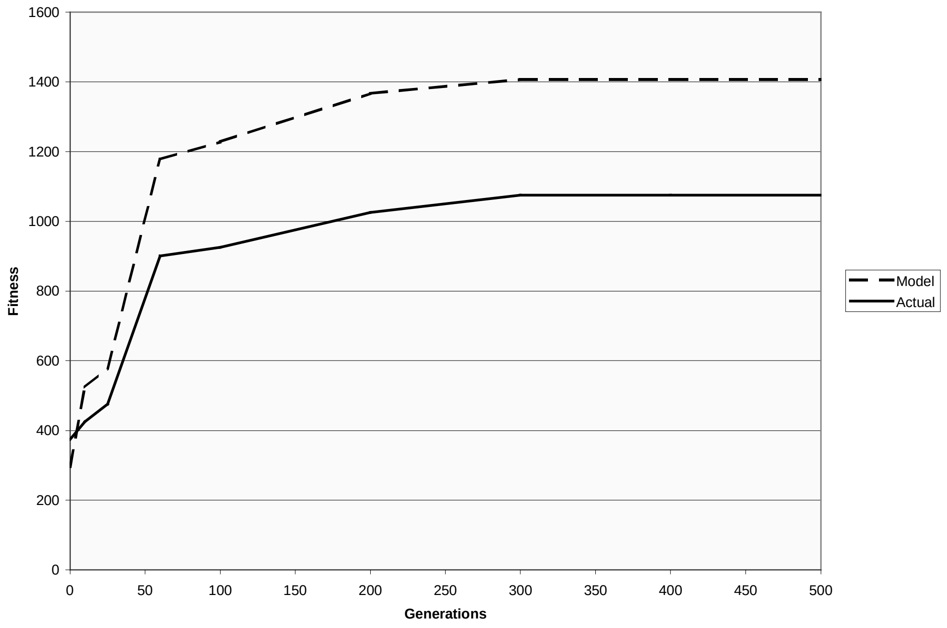

Figure 7: Comparison of the model to actual robot performance for a single population over the 500 generations of training.

Tests on the actual robot confirmed the viability of the produced gaits. Figure 7 shows the results of actual tests on the robot using the 0, 10, 25, 60, 100, 200, 300, 400, and 500 generations of one of the populations. As can be observed, the system consistently overestimates the fitness in the higher ranges. The model is purposely simple to reduce computation time and does not take into account lost speed due to slippage and actuators moving slower due to resistance. Both of these factors have more of a negative effect at high speeds. In addition, the model does not compensate sufficiently for the weight of the robot. Observations of actual tests on the robot show that the legs need to be lifted higher to avoid some drag during leg repositioning. The gait is fast enough, however, that there is minimal time with only three legs on the ground. The result is a steady forward movement with little time wasted. Figure 8 shows a comparison of the end products of the five trials. In all cases the gait produced was a fast tripod.

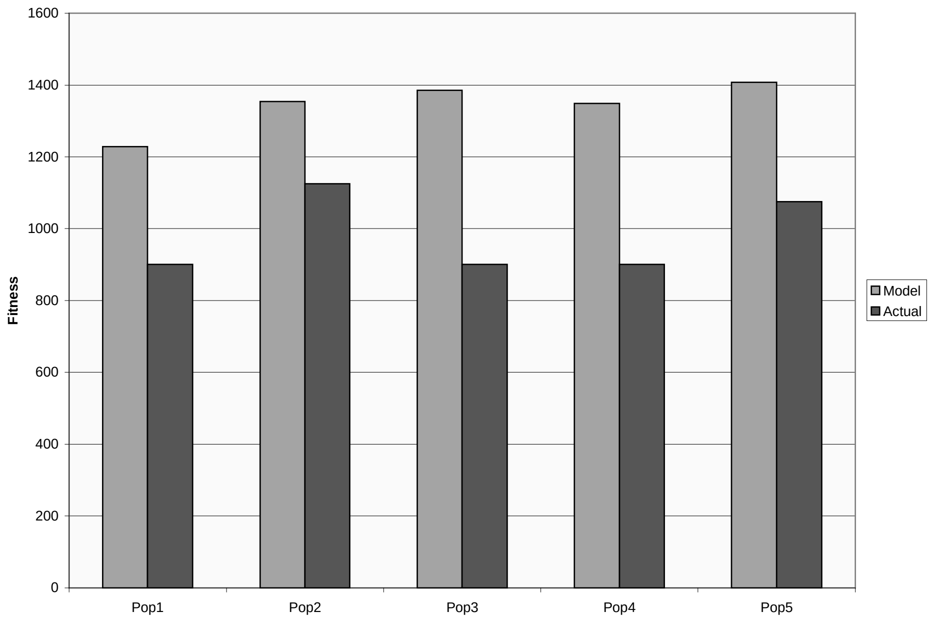

Figure 8: Comparison of the model/actual fitness for all five populations after training is complete.

5 CONCLUSIONS

Incremental evolution is an effective means of evolving gaits for hexapod robots. In the first increment CGAs can be used to generate the cycles of pulses required to produce a leg cycle for a two servo leg. Tests in simulation showed that they improve performance significantly over training and observation of the results on an actual robot confirmed the viability of the produced cycles. In the second increment, these leg cycles can be combined in such a way that their concurrent execution can produce a gait. Using a GA to coordinate the 6 leg cycles, with fitness predicated on maximum forward movement, the leg cycles can be combined to form a near-optimal gait cycle. Tests in the simulation and the actual robot confirm the viability of this method.

References

1. Beer, R. D., and Gallagher, J. C. (1992). "Evolving Dynamical Neural Networks for Adaptive Behavior." Adaptive Behavior, 1 (pp. 91-122). Cambridge: MIT Press.

2. Gallagher, J. C. and Beer, R. D. (1994). "Application of Evolved Locomotion Controllers to a Hexapod Robot." Technical Report CES-94-7, Department of Computer Engineering and Science, Case Western Reserve University.

3. 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).

4. Parker, G., Braun, D., and Cyliax I. (1997). "Evolving Hexapod Gaits Using a Cyclic Genetic Algorithm." Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC'97). (pp. 141-144).

5. Parker, G. (1998). "Generating Arachnid Robot Gaits with Cyclic Genetic Algorithms." Genetic Programming 1998: Proceedings of the Third Annual Conference. (pp. 576-583).

6. Spencer, G. (1994). "Automatic Generation of Programs for Crawling and Walking." Advances in Genetic Programming. (pp. 335-353) K. Kinnear, Jr. (ed.), Cambridge, Ma: MIT Press.