Fitness Biasing to Produce Adaptive Gaits for Hexapod Robots

Gary B. Parker
Computer Science
Connecticut College
New London, CT, USA
parker@conncoll.edu

Abstract— **Anytime learning with fitness biasing was shown in an earlier work to be an effective tool for learning leg cycles for a hexapod robot. This learning system was capable of adapting to changes in the environment. Although the leg cycles were appropriate for rougher terrain, the gaits produced with them by a standard genetic algorithm were not capable of bearing the robot's load. In this paper, we present the use of anytime learning with fitness biasing to improve the gaits produced by allowing the learning system to adapt to unforeseen changes in the environment and the robot's capabilities. Training and tests were done in simulation, with the resultant gaits tested on the actual robot.**

Keywords- evolutionary robotics, genetic algorithms, locomotion, control, anytime learning, six-legged robot

I. INTRODUCTION

Learning the control programs that produce gaits for hexapod robots is a difficult problem. It is particularly challenging if one wants these programs to be altered to adapt the gaits for changes in the robot's capabilities or the environment. Gait generation can be broken down into two main parts: 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 [1] or they can be learned separately [2]. Learning together greatly increases the complexity of the learning algorithm so details are often lost in the abstraction necessary to keep the computations within reason. Since some detail is lost, the gaits produced by this method cannot fully exploit the capabilities of the robot. Individual leg learning can take into account the capabilities of the actuators and movement constraints of individual legs. This allows better use of each leg as long as the controllers are complex enough to handle the increased details. The subsequent learning of the coordination of these leg cycles results in a gait that takes full advantage of the capabilities of the individual legs.

Some form of evolutionary computation (EC) would work well in learning what signals are needed to produce the individual leg cycles and their coordination since EC is well suited for adapting a solution to the peculiarities of a problem. Randall Beer and John Gallagher [3,4] used genetic algorithms (GAs) to develop neural network (NN) controllers for a simulated hexapod robot. NN structures were defined for the leg cycles and the coordinators and the GA learned the weights required to generate gaits. Lewis, Fagg, and Solidum used incremental evolution to find the parameters of two neurons (forming an oscillator) to control the leg and then a network of these oscillators was evolved to coordinate the movements of the different legs [5]. In previous work on the incremental evolution of gaits, we used cyclic genetic algorithms (CGAs) to evolve a sequence of primitives that control servos on each leg [2]. The learning system took into consideration the peculiarities of each leg to evolve the best leg cycle for that leg. A standard GA was then used to evolve the coordination of these leg cycles to produce gaits.

A difficulty with these methods is that they do not alter their control programs to adapt to changes in the robot's capabilities or the environment. Grefenstette and Ramsey (1992) introduced the method of *anytime learning* that integrates the control system and the learning system [6]. This method allowed a learning component to continually compute a best control solution while the robot operated in the environment using the latest best solution. It could adapt quickly to changes in the robot or the environment by having the robot's sensors continually update the status of the environment and the robot's capabilities in the learning component's simulation. Unfortunately, this type of anytime learning would not work for ServoBot learning, which has to depend on global observation, instead of precise sensors, to determine the robot's capabilities. *Punctuated anytime learning* was developed to compensate for the ServoBot system's lack of quick feedback, precise sensors, and high computational power [7]. Training with the EC takes place off-line on a simple model while periodic checks on the actual robot help to improve the learning systems output. This type of anytime learning is referred to as punctuated because the learning, although it is continuous, has break points where it is working on newly updated information about the actual robot.

In previous work, we used *fitness biasing,* a form of punctuated anytime learning, to evolve adaptive leg cycles for the ServoBot [8]. The system adapted the leg cycles to be more appropriate for walking on carpet after they had been initially learned for a smooth surface. A standard GA used these leg cycles to evolve a high-stepping tripod gait for the hexapod robot. This gait was a significant improvement over the previous gait when the robot was tested on carpet. There were, however, still some issues with this gait. Although the leg cycles were better since

they lifted the legs high enough to avoid drag on the carpet, the ratio of time that they were in the air compared to on the ground increased. This resulted in a gait that had difficulty holding the weight of the robot.

In this paper, we present the second increment of punctuated anytime learning required for adaptive gait generation. We use fitness biasing to evolve gaits using the leg cycles that were learning using fitness biasing during the first increment of learning. The leg cycles changed due to a change in environment (going from a smooth surface to a carpeted surface), the gait needs to change in response to the leg cycle changes since the previous gait could no longer hold up the robot using the new leg cycles. The gaits produced using fitness biasing are still appropriate for the carpeted surface, but are also capable of bearing the weight of the robot.

II. The ServoBot

The ServoBot is a hexapod robot that has two degrees of freedom per leg. Twelve servos, two per leg, provide thrust and vertical movement. The servos 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 duration 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 and 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 duration 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 difference in the pulse durations is 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 durations. For example, moving a leg using consecutive pulse durations 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.

With a program that indicates the desired pulses, each leg's controller (BASIC Stamp II) can produce a sequence of pulses that direct the position of its two servos. The central stamp controller directs each leg stamp when to start its sequence (leg cycle) and if needed, when to cut short one cycle to start another in order to maintain leg coordination.

III. Incremental Gait Learning

A. First Increment: Evolving Leg Cycles

To produce leg cycles, a control program needs to be generated that controls the output of each BASIC Stamp. The output is a sequence of pulses that continually position the leg's servos. This sequence can be learned using a cyclic genetic algorithm (CGA), a variation of the standard GA that uses a chromosome made up of tasks to be completed in a set amount of time as opposed to traits of a solution [9]. The CGA can learn single loop control programs since some portion of the chromosome can be designated as the iterative section (a sequence of tasks that will be continually repeated). For this implementation, the tasks are sequences of pulses that are to be sent to the servos. Further details will not be discussed since the work reported in this paper deals with using punctuated anytime learning to assist in the second increment of gait learning, which uses a standard GA. The CGA has been used to generate reasonable leg cycles for each of the six legs [2]. These leg cycles were downloaded and observed on the actual robot where they produced efficient, useable leg cycles. The length (number of pulse commands) of the longest of the 6 optimal leg cycles was 36. A set of leg cycles using a range of desirable lengths were needed to produce a gait. Each leg was retrained for 500 generations to learn the optimal leg cycle with a desired length of 36. This population was then used to learn leg cycles with desired lengths from 21 to 52.

B. Second Increment:Evolving Gaits from the Leg Cycles

Gait generation required the coordination of the proper combination of leg cycles, one from each leg [2]. The gait learning algorithm would be able to choose leg cycles from anywhere in the range learned during the first increment for each leg to come up with the proper combination of leg cycles. In addition, the time to start each leg cycle needed to be learned in order for them to be coordinated in providing thrust and repositioning from other thrusts while bearing the weight of the robot.

Gait training was done using a standard GA. The chromosome (Figure 1) was made up of 7 parts. The gait cycle length (GCL) represented the number of pulses in the overall gait cycle. Information for each leg included its leg cycle length (LCL), which selected which leg cycle to use, and start time (START), which specified the time that the corresponding leg cycle should start.

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

Figure 1. Chromosome used for gait training.

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 of the individual leg cycles. 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 reached, 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.

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 used. 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.

Two types of crossover were used. In one, crossover was accomplished by randomly picking corresponding spots in the two selected parents. In the other, crossover was a gene-by-gene crossover that performed 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.

This method of incremental learning worked well for producing gaits for the ServoBot operating on a consistent (smooth) surface [2]. In all five test cases, near optimal tripod gaits were produced.

IV. Anytime Learning with Fitness Biasing

Fitness biasing [10] is a form of punctuated anytime learning, which employs off-line learning using evolutionary computation with the control program being downloaded to the on-line controller. The off-line learning does not require internal sensors but uses global observation to make the required adjustments to guide the evolutionary computation. The results of periodic tests, done on the actual robot, are used to bias the fitnesses calculated by the evolutionary computation, which uses a model of the robot. This allows the learning system to adapt to changes in the robot's capabilities to provide continually updated control programs.

Anytime learning with fitness biasing allows the system to modify the GA based on the performance of the robot. Figure 2 shows how it affects the learning. The model is not changed; its parameters were set before anytime learning began. The periodic checks on the actual robot alter the processing within the GA in an attempt to improve the result of training. This involves the biasing of evolved solution fitnesses by comparing their performance on the model to their performance on the actual robot.

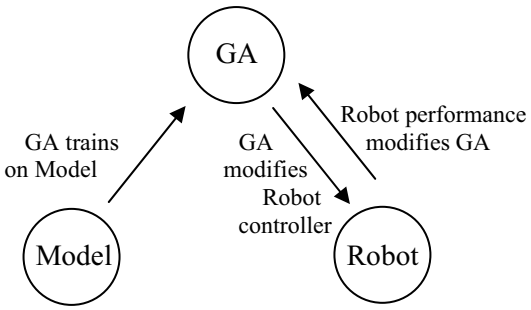

Figure 2. Fitness biasing affects the genetic algorithm.

Probability for selection is determined by computing each individual's fitness on the robot model. After each *n* generations all solutions of the GA population are tested on the actual robot. These measurements are used to bias the fitnesses found on the model to make them equal to the actual robot fitnesses. These biases are used as the GA continues training. In this way, solutions that are good on the actual robot but poor on the model have boosted fitnesses during training, which results in their production of more offspring. This solution requires *population-size* actual tests every *n* generations.

```
Model-Fitness = Compute-Model-
Fitness(Solution)
if absolute-value(Model-Fitness) < 1
   Bias = 1
else
   Actual-Fitness = Test-on-Robot(Solution)
   Bias = Actual-Fitness / Model-Fitness
```

Figure 3. Algorithm to compute bias.

A bias for each solution of the population is computed using the algorithm in Figure 3. This bias is stored with its corresponding solution. It is used in subsequent generations of the GA to alter the fitness of the solution computed on the model of the robot. This is done by multiplying the fitness computed on the model by the bias:

$$Corrected\text{-}Fitness = Model\text{-}Fitness * Bias \quad (1)$$

These *Corrected-Fitness*es are used for selection during the subsequent training being done by the GA. Pairs of individuals are stochastically selected for reproduction using the *Corrected-Fitness*es. The two individuals produce a single offspring for the next generation; combining their attributes by crossover with possible random variations caused by mutation. The new offspring's bias is computed by averaging the biases of its parents.

A. Learning Leg Cycles Using Fitness Biasing

The gaits learned using incremental learning discussed in Section III worked well on the smooth surface since they had minimal vertical movement in order to reduce the time for leg repositioning. This was effective because it provided time for more legs to simultaneously be on the ground, resulting in additional stability and consistent thrust, but it was inappropriate for rougher surfaces such as carpeting since returning leg's produced significant drag. In order to adapt the leg cycles to be able to handle rough surfaces, anytime learning using fitness biasing was employed.

The same model and CGA used in earlier results (Section III) were used as the learning part of the punctuated anytime learning system. Tests on the actual robot (simulated) were done after every ten generations. A list of 64 biases (one for each individual (chromosome) in the solution set), all being initialized to one, was produced. Training was continued for 200 CGA generations. This involved 200*64 evaluations on the model and 20*64 evaluations on the actual robot. This learning, using fitness biasing produced leg cycles, adapted to the new carpeted surface. The legs would raise more before and after they moved horizontally during the return portion of the cycle.

Leg cycles produced for rough surfaces were then used by a GA to evolve convolutions of leg cycles that formed tripod gaits. Since the new gaits had increased length, leg cycles with desired lengths from 36 to 67 were learned using the techniques discussed in Section III. Gait training, the chromosome used (Figure 1), and the model robot were all as described in Section III. In all five test cases, tripod gaits with high stepping leg cycles were produced [8]. These high stepping gaits were more appropriate for the rough surface, but now that more time was required for the leg to return for another thrust, there was less time with all legs on the ground. This resulted in a gait that worked well initially, but would slow down because the robot's weight, which was higher than normal because of the prototype board holding the controllers, was too much for the sustained support by three legs.

V. Anytime Learning with Fitness Biasing to Learn Adaptive Gait Cycles

Although the leg cycles adapted by anytime learning with fitness biasing to handle rougher surfaces were used to form new gait cycles, the results were not as good as desired. Nice tripod gaits evolved in the model, but due to the weight of the seven stamp control board, the robot could not statically hold itself up on three legs. In previous experiments where the leg cycles were optimized for a smooth surface, this was not as much of a factor since the robot spent minimal time on only three legs. With the leg cycles being optimized for conditions where increased vertical lift was required to reposition the legs, more time was needed for repositioning, so the robot spent more time on three legs. Although these gaits were useable on carpet, since the legs lifted high enough to avoid catching during their swing, the underbody servos were often being dragged along the ground. This worked in the lab, but was determined to be undesirable for field use. Anytime learning with fitness biasing was employed in evolving the gaits from the new leg cycles to rectify the problem.

A. Training

Training was done with the actual robot being simulated in a world where dragging the body was heavily penalized. Five populations of 64 randomly generated individuals as described in Section III were used to start the training. After each five generations of training on the model, the solution was checked on the simulated robot. The results of these tests were used to bias the fitnesses as described in Figure 3. A total of 100 generations on the model, which is equivalent to 20 generations on the robot, were used. The best individual solution from each of the five starting populations was stored after every ten generations of training on the model.

B. Results

Tests showed that anytime learning with fitness biasing could generate a gait for this new environment even though the genetic algorithm was working with an inaccurate model. Figure 4 summarizes the results done in simulation. Consider first the dashed lines only. They represent learning without anytime learning. The thinner line represents what the fitness would be if it were tested on the inaccurate model. The thicker line shows what the fitness really is on the simulated robot. The fitness improves quickly on the model as the genetic algorithm works to optimize a gait for it, but since robot tests are not part of the learning system, performance on the robot actually decreases. The learning system is learning a tripod gait that will have insufficient support for the robot to stay off the ground.

Now consider what happens when anytime learning with fitness biasing is part of the learning system (the solid lines). The thin solid line shows the fitnesses attained when tested on the inaccurate model. The thicker solid line is the resultant fitness on the robot. Comparing the thin solid line to the thin dashed line one can see that anytime learning reduced the resultant fitness on the model. This was caused by biasing the fitnesses away from solutions that were optimal for the model in favor of the robot. A comparison of the thicker lines shows that the anytime learning drastically improved the learning on the simulated robot (solid thick line versus dashed thick line).

Figure 5 shows a comparison of the learning curve when using anytime learning versus learning directly on the simulated robot. The x-axis shows the number of trials on the robot required for training when using either method. As can be seen, the anytime learning is an improvement over a genetic algorithm applied directly to the robot.

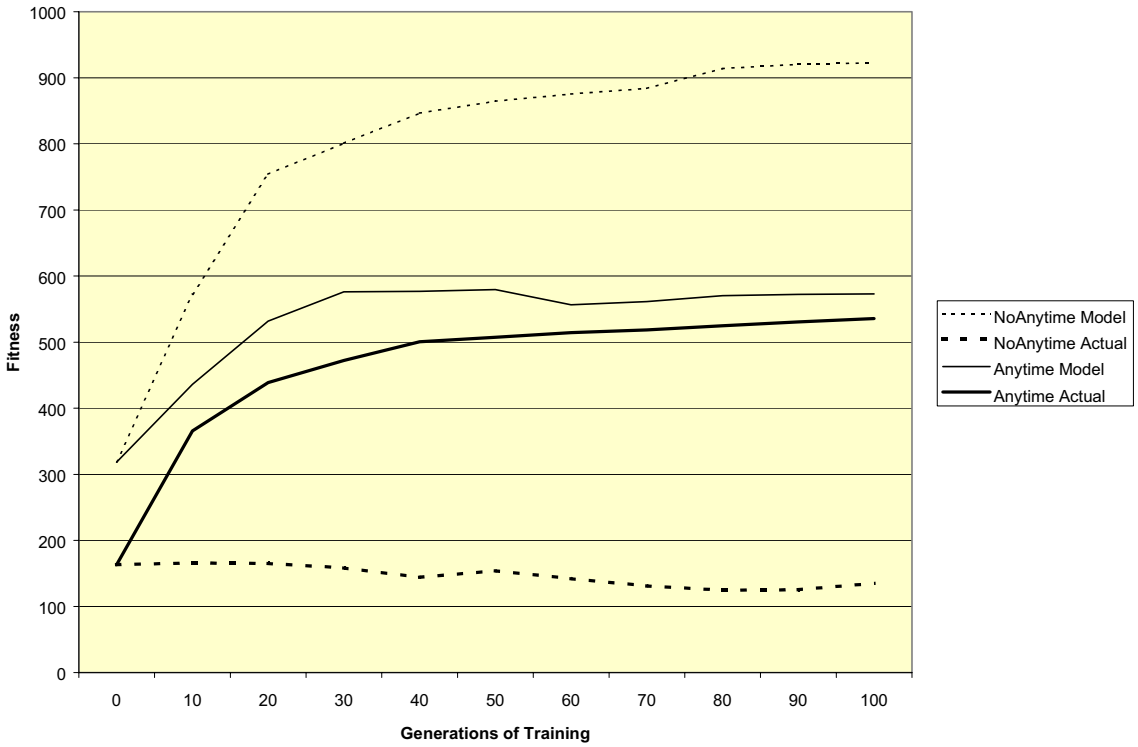

Figure 4. Comparing normal and anytime learning with fitness biasing on gait production from leg cycles. Fitness is the distance (mm) traveled in 500 pulses.

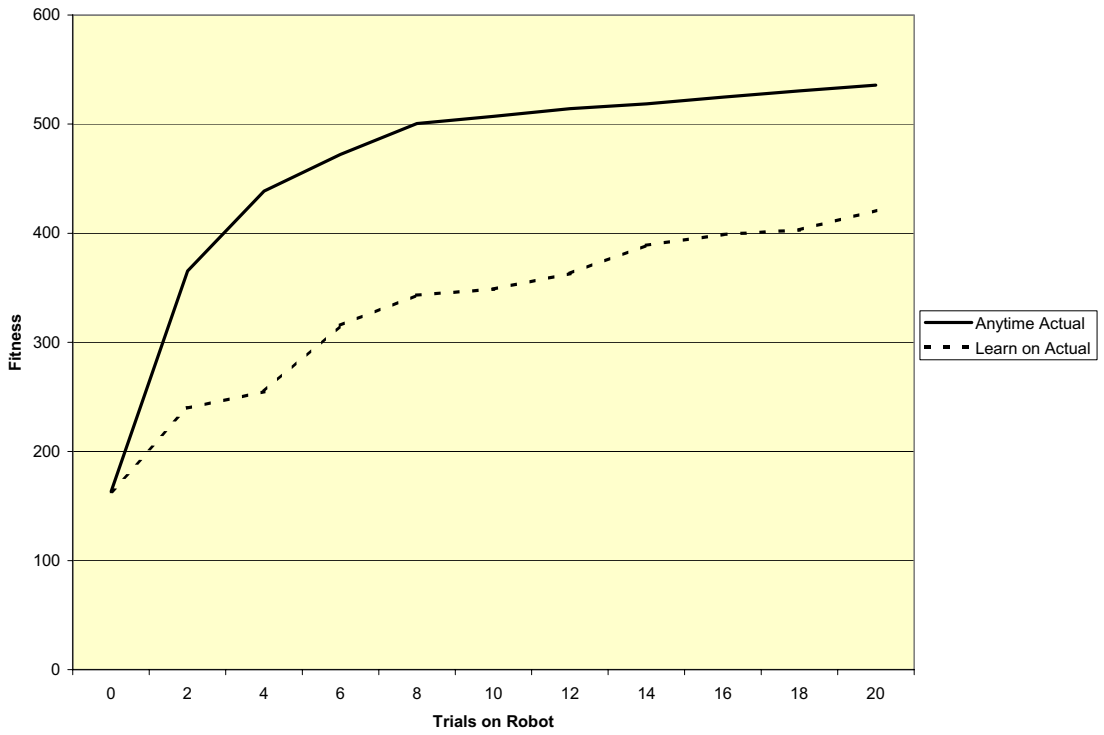

Figure 5. Anytime learning with fitness biasing versus learning directly on the robot.

Examination of the produced solutions revealed that the gaits formed were similar to those of insects using a slow moving metachronal wave. Tests on the actual robot showed that the gaits were an improvement over the ones learned without anytime learning; due to additional support, the robot body no longer touched the ground. They were not, however, that much faster. Using the five new gaits the average forward movement in 30 seconds was 136 cm; the old gaits' average was 124 cm. This was due to the fact that the ServoBot has six of its servo motors mounted on its underside giving it a clearance of only about 20cm. In addition, the underside of the servo motors is smooth. Both these factors allow the robot to comfortably drag itself without significant loss of speed. Penalties were introduced into the simulation to make dragging an undesirable option to accommodate robots without the low clearance and smooth underside. The gaits generated by anytime learning would be equally effective on high clearance robots since nothing except the feet of the robot touched the ground.

Figure 6 shows a comparison of two sample gaits produced during training. The top diagram is the tripod produced from Section IV. As discussed in the introduction to this section, this gait had minimal time when more than three legs were on the ground. The result was that the robot could not hold itself off the ground due to the excessive weight of the seven stamp controller board. The bottom diagram shows one of the gaits produced using anytime learning. As can be seen, there are almost always at least four legs on the ground resulting in the additional support required to hold up the heavy controller board.

Figure 6: Rough terrain gaits. The dark areas are times when the leg's foot is off the ground. The top diagram shows a sample tripod gait produced using the high stepping leg cycles. This gait lacked the support needed to keep the heavy robot off the ground at all times. The bottom diagram shows a sample gait produced after anytime learning with fitness biasing was applied to gait generation. The second gait seldom had more than two legs off the ground at one time, so it provided the support needed to hold the robot off the ground. This gait is very similar to the slow metachronal wave used by some slow walking insects.

VI. Conclusions

Anytime learning with fitness biasing was very successful at allowing the learning system to adapt the evolution of gaits to compensate for changes in the environment and the capabilities of the robot. Even though the model of the robot/environment being used by the learning system became incorrect, fitness biasing allowed it to adapt to the changes. As the simulated robot moved from a smooth surface to a rough surface the learning system adapted the leg cycles to be high stepping. This change revealed an inaccuracy in the model's parameters concerning the weight bearing capabilities of the robot. Fitness biasing adapted the gaits produced using these new high-stepping leg cycles despite the model's inaccuracies. Tests done on the actual robot showed that the produced gaits were appropriate for the robot operating on the rough surface. This successful use of fitness biasing on two levels of robot control helps to demonstrate its general applicability in this area.

References

[1] 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.

[2] Parker, G. (2001). "The Incremental Evolution of Gaits for Hexapod Robots." *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001).* pp. 1114-1121.

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

[4] 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.

[5] 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. 2618-23.

[6] Grefenstette, J. and Ramsey, C. (1992). "An Approach to Anytime Learning." *Proceedings of the Ninth International Conference on Machine Learning*. pp. 189-195.

[7] Parker, G. (2002). "Punctuated Anytime Learning for Hexapod Gait Generation." *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems.* pp. 2664-2671.

[8] Parker, G. (2003). "Learning Adaptive Leg Cycles Using Fitness Biasing." *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003).* pp. 100-105.

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

[10] Parker, G. and Mills, J. (1999). "Adaptive Hexapod Gait Control Using Anytime Learning with Fitness Biasing." *Proceedings of the Genetic and Evolutionary Computation Conference.* pp. 519-524.