Learning Adaptive Leg Cycles Using Fitness Biasing

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

Abstract -- This paper discusses the use of fitness biasing to alter the control of a seven microprocessor robot as it shifts from one environment to another. The robot was initially using a gait evolved to work on a smooth surface (tile). When tested on a rough surface (carpet) the learned gait was found to be inappropriate because the legs were causing drag as they repositioned. An efficient move to reposition on the smooth surface did not work on the rough surface. Anytime Learning with Fitness Biasing was applied to the continued evolution of the individual leg cycles as the simulated robot moved from an area of smooth to rough terrain. An actual robot was used to test the results. Following training using fitness biasing, the robot's gait was more appropriate for a rough surface as it learned to raise its leg more before initiating the return movement.

I. INTRODUCTION

Gait generation for multi-legged robots is a difficult problem. Each leg must learn a cycle that effectively produces forward thrust in the environment where it is expected to perform. These individual leg cycles must then be coordinated so that the robot maintains stability as the legs produce thrust and avoid causing drag. The problem is compounded by the fact that each leg has its own peculiarities in performance, which makes coordination more difficult. The use of incremental learning with evolutionary computation has been shown to be an effective way to learn gaits for hexapod robots [6]. These robots were equipped with seven microprocessors (one for each leg and one for coordination). The gaits learned took advantage of the capabilities of each leg to produce smooth forward locomotion.

The difficulty of the problem is increased when considering that there will be changes in the leg's capability or the environment while the robot is in operation. Adjustments to the control program must be made in real time. Anytime learning with fitness biasing [4] offered a means for the learning system to make adjustments after training was complete and the robot was in operation. This was done by using periodic checks on the robot to affect the solution fitnesses during evolution. Tests in simulation showed that this method was effective in providing adaptive gait control for a hexapod robot that went through a series of leg capability changes. This robot was equipped with a single microprocessor that could send signals to each leg to move backward or forward and up or down. Although these tests showed the effectiveness of fitness biasing, the controller used lacked the capacity to adjust the speed of movement of the individual legs. This limited the possible solutions available to the system and made adjustments for changes in the environment very difficult.

Other researchers have used incremental learning to produce gaits by first learning leg cycles. Beer and Gallagher used Genetic Algorithms (GA) to develop neural network (NN) controllers for a hexapod [1]. The controller had foot, backward swing, and forward swing motor neurons; two internal unspecified neurons and a single angle sensing neuron. The angle sensing neuron was connected to all 5 of the other neurons and those 5 were fully connected to each other. The 5 control neurons also had a threshold and time constant parameter. After an individual leg cycle was learned for a generic leg, six copies of the leg controller were made. Each of the leg controllers was connected to their neighbor leg controllers. A GA was used to learn the connection weights between the leg NNs. Lewis, Fagg, and Solidum used incremental evolution also [2]. 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, cyclic genetic algorithms (CGAs) were successfully used to evolve the pulse sequences that were needed to control servos on each leg [6]. These pulse sequences, which took into consideration the peculiarities of the leg's capabilities, were sent to the servos to generate leg cycles. A GA was then used to evolve the convolution of these cycles to produce a reasonable gait. This gait was a classic tripod that worked very well on the smooth surface of a tile floor. The leg cycles, since they were optimized for speed, 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. In addition, since the floor was smooth, short times of opposite direction or no leg movement while a leg was on the ground did not produce significant drag.

Tests determined, however, that these leg cycles would be inappropriate for rougher surfaces such as carpeting. In this paper we discuss the use of anytime learning using fitness biasing to adapt the leg cycles to be able to handle rough surfaces. With the fitness biasing method, the model used by the CGA is not altered; it is the same model used for smooth surfaces. Instead, periodic tests are done on the actual robot (simulated) and the results of these tests are used to bias the fitnesses of the corresponding solutions.

Leg cycles produced for rough surfaces are then used by a GA to evolve convolutions of leg cycles that formed tripod gaits. Tests showed that this method is an effective learning system that is adaptive to changes in the walking surface.

II. SEVEN STAMP SERVOBOT

This learning method 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 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 obtained 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 servos capability. Consecutive pulses of 40, 240, 440, etc. would probably result in the same speed as the consecutive pulses of 40, 340, 640, etc.

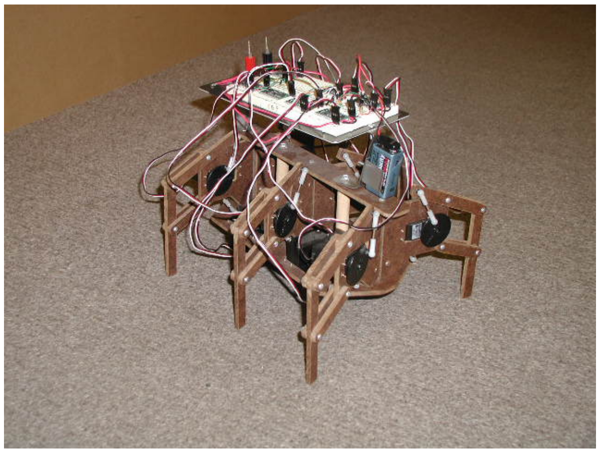

Fig. 1: ServoBot with seven BASIC Stamp II controllers; one for each leg and one to coordinate the legs.

Control for the ServoBot will be provided by seven BASIC Stamp IIs, one per leg and one working as the overall controller. Each leg's stamp takes in a sequence of pulses that indicated the position of its two servos. The central stamp controller tells 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.

A. Leg Model

A model of the leg was used to do the evolution. 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 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 the millimeters off the ground when the leg was fully lifted. Along with these positions the pulse width 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.

III. CYCLIC GENETIC ALGORITHMS FOR LEG CYCLES

Cyclic Genetic Algorithms were developed [3] to allow for the representation of a cycle of actions in the chromosome. They differ from the standard GA in that the genes represent solution tasks instead of traits and the chromosome is circular instead of linear in structure. The genes of the CGA chromosome can be one of several possibilities. They can be as simple as a set of actuator activations to as complex as cyclic sub-chromosomes that can be trained separately by a CGA. For our purposes, the genes represent a sequence of servo pulses. The trained chromosome contains the cycle of servo control pulses that will be continually repeated by the leg's controller to produce a leg cycle.

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 differing by only 20 microsecond 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 7 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 allow smooth horizontal movement, but smooth vertical movement was not needed since vertical movement does not affect the smoothness of the robot's movement over the ground.

((R1 HP1 VP1) (R2 HP2 VP2) ... (R8 HP8 VP8))

Fig. 2: 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 2 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 (*hp* is the horizontal pulse and *php* is the previous horizontal pulse) was used:

$$\textit{pulse increment} = (hp - php) \,/\, \textit{repetitions} \qquad (1)$$

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* affected 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.

CGA evolution took place on the leg model in the generation of a chromosome to control the leg cycle. 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.

IV. ANYTIME LEARNING WITH FITNESS BIASING

Anytime learning with fitness biasing is one of two methods of *punctuated anytime learning* [4,5,7]. It is a learning system that 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 system to modify the CGA based on the performance of the robot. The model is not affected; its parameters were set before anytime learning began. The periodic checks on the actual robot alter the processing within the CGA in an attempt to improve the result of training.

Probability for selection is determined by computing each individual’s fitness on the robot model. This fitness is computed for each individual in the population. After each *n* generations all solutions of the CGA 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 CGA 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.

Biases are computed for each solution by dividing the actual fitness by the model's fitness. Each bias is stored with its corresponding solution. It is used in subsequent generations of the CGA to alter the fitness of each solution computed on the model of the robot by multiplying this fitness by the bias. These corrected fitnesses are used for selection during the subsequent training being done by the CGA. Pairs of individuals are stochastically selected to form a new offspring through crossover and mutation. The new offspring’s bias is computed by averaging the biases of its parents.

V. FITNESS BIASING APPLIED TO LEG CYCLE GENERATION

The test, done in simulation, was designed to determine if fitness biasing could be used with a CGA to adapt leg cycle control as the robot moved from a smooth to a rough surface. The CGA used the original smooth surface model for evolution. The robot was represented by a second model that required additional vertical lift before a leg would be high enough to not influence the its horizontal movement. Each leg’s fitness was determined by the thrust that it produced while on the ground. The best leg cycle produced forward thrust (by moving the leg back) while the leg was on the ground and used minimal time to reposition the leg. But if reposition started (by moving forward) before the leg cleared the ground, negative thrust would result. The robot simulation model required more vertical separation between the ground and the leg before reposition could be done without producing negative thrust.

A. Training

Training began with the CGA operating on an inaccurate (smooth surface) model. Tests on the actual robot (simulated) were done after every ten generations (n = 10). A list of 64 biases (one for each individual (chromosome) in the solution set), all being initialized to one, was produced. Upon each generation of training (by the CGA) the fitnesses found by evaluating the 64 individuals of the population on the CGA model were multiplied by their corresponding bias to gain an adjusted fitness (*adjusted_fitness = model_fitness * bias*).

$$\begin{tikzcd}
I_i \arrow[dr] & & I_j \arrow[dl] & & B_i \arrow[dr] & & B_j \arrow[dl] \\
& I_k & & & & \dfrac{B_i + B_j}{2} &
\end{tikzcd}$$

Fig. 3: Bias calculation for offspring I_k from parents I_i and I_j.

This list of 64 adjusted fitnesses was used by the CGA for its standard selection followed by crossover and mutation. The biases also changed during evolution. The bias of each child was the average of its parents’ biases; see Figure 3. Individuals I_i and I_j (i and j being indexes between 0 and 63 corresponding to the 64 individuals in the population) are stochastically selected for reproduction. Their offspring I_k is the result of applying crossover and mutation to the parents. The bias assigned to I_k is the average of the two parents’ biases.

As training continued, the biases tend toward the average of all the biases. After each ten generations of CGA training, the original biases were disregarded and new biases were calculated (Figure 4).

$$(I_0, I_1, I_2,, \ldots I_n) \xrightarrow{\text{evaluate on model}} (M_0, M_1, M_2,, \ldots M_n)$$

$$(I_0, I_1, I_2,, \ldots I_n) \xrightarrow{\text{evaluate on robot}} (R_0, R_1, R_2,, \ldots R_n)$$

$$(B_0, B_1, B_2, B_3, \ldots B_n) = \left(\frac{R_0}{M_0}, \frac{R_1}{M_1}, \frac{R_2}{M_2}, \frac{R_3}{M_3}, \ldots \frac{R_n}{M_n} \right)$$

Fig. 4: Computation of biases. Each individual solution is tested on the model and the actual robot. The fitnesses that correspond to each individual are used to calculate the bias for that individual by dividing the actual fitness into the model fitness.

The biases were set so that the next generation of stochastic selection would be predicated on the fitnesses of the individual solutions on the actual robot. The system started out doing ten generations of training with the actual robot requiring only the minimal vertical movement for a good gait. After the ten generations, the environment changed to one where the robot required increased vertical movement. Training was continued for another 200 CGA generations. This involved 200*64 evaluations on the model and 20*64 evaluations on the actual robot.

B. Results

Anytime learning with fitness biasing was used over 200 generations of CGA training on each of the six legs. The results discussed are the average of these six legs.

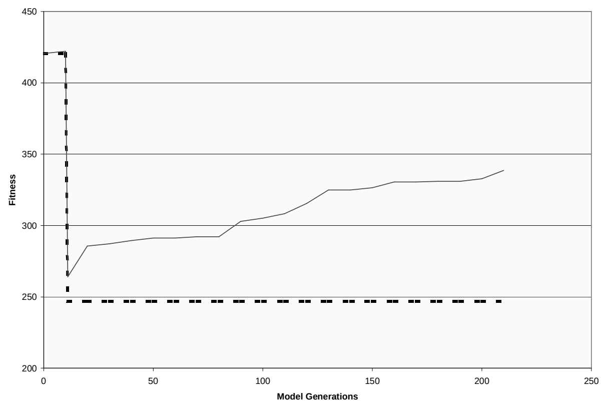

Fig. 5: Anytime learning (solid line) versus no anytime learning (dashed line).

Figure 5 shows a comparison of the actual robot leg performance when anytime learning was used versus when it was not. Without anytime learning, (dotted line) the actual robot leg fitness fell sharply at the tenth generation where the robot simulation entered an area where increased lift was required. No improvements were made to this since the training system was still using the outdated model (specifying that minimal lift is required) to find leg cycle solutions. The solid line shows the average fitness of the six legs when anytime learning was used. At the transition point, anytime learning gained an immediate advantage since the test on the actual robot leg helped to identify a better best solution. Its benefit did not stop there, as the anytime learning continued to increase the actual leg fitness over the next 200 generations.

This training took 20 generations of tests on the actual robot legs, which is the largest expense with anytime learning. Although anytime learning increased the solution over training without anytime learning, is it an improvement over doing the CGA directly on the robot? Figure 6 shows the same information as is shown in Figure 5, but a new line is added that shows what would happen if the training was done directly on the simulated robot. The generation numbers are shown in actual generations. The fitness again shows steady improvement, but this time the improvement is not as fast. Anytime learning with fitness biasing worked better than training on the model or directly on the robot.

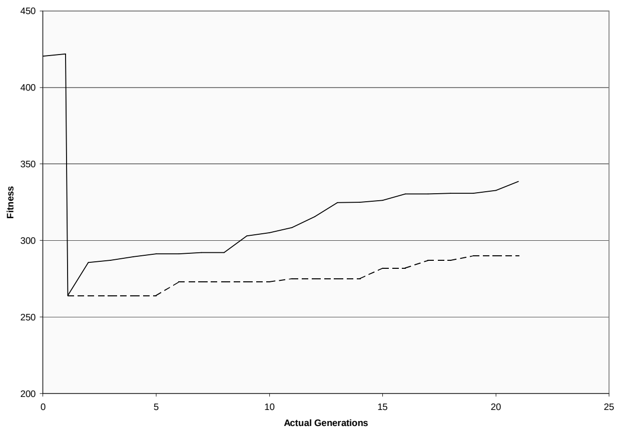

Fig. 6: Anytime learning (solid line) versus learning on the robot (dashed line).

VI. GAIT GENERATION

The previous section discussed the use of anytime learning to produce leg cycles that were optimized for rough terrain. The desired length (number of pulses in the cycle) for these leg cycles was not specified. The end results varied in length from 36 to 56 pulses. In order to use leg cycles for the generation of gaits a set of leg cycles using a range of desired lengths needed to be produced. The gait learning algorithm needs to be able to choose leg cycles from anywhere in this range for each leg to come up with the proper coordination of legs. The lowest "no length specified" optimal should be the lowest in this range since it would be unlikely that the best gait would use leg cycles that were all lower than their optimal. Gait cycles with desired lengths from 36 to 67 were learned using the techniques discussed in previous work [7].

The best individuals at generations that were multiples of 10 were stored. The results on the robot model are shown in Figure 7. Graphs of the fitness growth of the five distinct starting populations along with their average (dashed line) after 500 pulses are shown. There are two things to note from this graph. The start fitness at generation 0 is fairly high. This is because all the legs are already moving in a near optimal cycle; they simply need to be coordinated. The GA quickly learns adequate coordination and works to improve this solution to find the optimal leg cycle lengths and start spots for each leg.

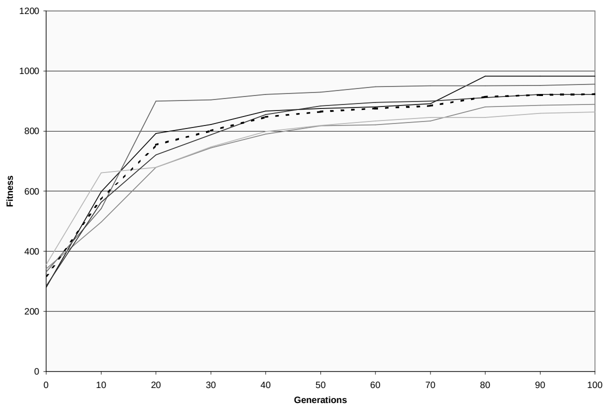

Fig. 7: Gaits formed from high-stepping leg cycles. Five tests are shown. The dashed line is the average performance of the five.

Tests on the actual robot confirmed the viability of this method. In all five cases, tripod gaits with high stepping leg cycles were produced. Figure 8 shows diagrams of two tripod gaits. The first is the gait produced with normal leg cycles from previous research [7]. The second shows the tripod gait produced using the higher stepping leg cycles formed from anytime learning. Now that more time is required in the leg swing, to allow it to lift over rough terrain, there is less time with all legs on the ground.

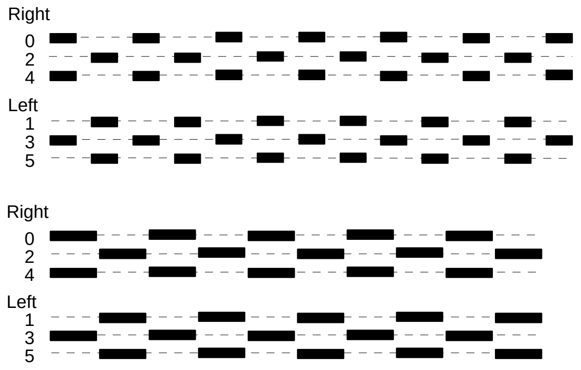

Fig. 8: Tripod gait for smooth terrain versus one for rough terrain. The smooth terrain gait (top diagram) takes minimal time to reposition the legs during their swing. For rough terrain, the stance (producing thrust) times are the same, but the swing times are longer as the leg must lift up further before it starts to move forward.

VII. CONCLUSIONS

Anytime learning with fitness biasing is an effective means of learning adaptive leg cycles, when used in conjunction with a cyclic genetic algorithm. It can assist in the correction of leg cycles to accommodate environmental conditions such as rough terrain by causing them to incorporate more lift into their cycles. Punctuated anytime learning has been shown, in simulation, to be more effective on these problems than direct learning on the robot. Further tests on anytime learning with fitness biasing should continue to reveal its general applicability to several problems in evolutionary robotics.

VIII. REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] Parker, G. (2000). "Co-Evolving Model Parameters for Anytime Learning in Evolutionary Robotics." *Robotics and Autonomous Systems*, Volume 33, Issue 1, pp. 13-30.

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

[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.