LOCOMOTION CONTROL CYCLES ADAPTED FOR DISABILITIES IN HEXAPOD ROBOTS

GARY B. PARKER and INGO CYLIAX
Department of Computer Science, Indiana University, Bloomington, IN 47405
gaparker@cs.indiana.edu, cyliax@cs.indiana.edu

ABSTRACT

Inherent in robot control and behavior is cyclicity. Sequences of actions taken by the robot tend to repeat. This is no more apparent than in the locomotion of hexapod robots. The movement of the separate legs must be coordinated in such a way that smooth forward motion will result. The individual legs each have their own cyclic nature and the combination of leg movements forms a cycle of over all movement. In primitive robots, with simple controllers that require a string of activations for locomotion, an appropriate cycle of these activations needs to be provided. The problem is further complicated by the robot's uniqueness and accompanying variance of leg capabilities. In previous work we introduced Cyclic Genetic Algorithms which have successfully been used to generate gaits for actual hexapod robots. In this paper, we extend on that work by showing that the CGA can adapt to disabilities in the robot and adjust the gait accordingly.

KEYWORDS: genetic, disabled, robot, hexapod, gait, control, cyclic

INTRODUCTION

Autonomous legged robots can be very useful in performing tasks in hazardous environments where the terrain precludes the effective use of wheels. Control of these robots can be difficult. The use of learning algorithms capable of adjusting to the intricacies of the robot's capabilities can be of significant use in development and can add the adaptability required to let the robot maintain autonomy.

Locomotion is an important part of legged robot control. The capabilities of the separate legs, such as rate and range of movement, must be accounted for when developing a gait. Changes in leg capabilities further complicate the situation as the controller tries to maintain a viable gait for continuous locomotion.

The complication of this task suggests that some form of Evolutionary Computation would be appropriate. Randall Beer and John Gallagher used genetic algorithms to develop a neural net to control a simulated cockroach [1] and showed that it was capable of controlling an actual hexapod robot [2]. This technique was successful for the robots used, which had relatively complex controllers. Small inexpensive robots that require a repeatable sequence of activations for control need to be able to generate gaits using minimal *a priori* knowledge of tripod gaits. Graham Spencer [3] used genetic programming and minimal *a priori* knowledge to evolve programs that could produce gaits. Although the programs learned were capable of producing gaits with sustained forward movement, they did not result in the optimal tripod gait. In addition, his tests were done only on simulations and not on an actual robot.

In previous work, we developed [4] Cyclic Genetic Algorithms (CGAs), which generate gaits using minimal *a priori* knowledge. It has been shown that the use of CGAs can be an effective means of gait generation for hexapod robot simulations. They can, with only low-level primitives, produce reasonable gaits in minimal time. Their output requires little in intermediate controller complexity, as it is a sequence of these primitives, which can be fed directly into the robot. In addition, the applicability of these algorithms has been tested on an actual robot [5]. A model for simulation was produced based on the measured capabilities of the ServoBot robot. This model was trained with the CGA and the evolved gaits were transferred to the actual semi-autonomous robot. They were found to be tripod (considered optimal for speed) in nature and comparable to the best designed by human engineers.

An important next step is to ensure that the CGA is adaptable to changes in the capabilities of the robot and that it can develop gaits for even the most severely disabled robots (legs missing). In this paper, we test the CGA’s ability in this regard by doing two types of test. In one, we slowly disable one or two legs by restricting their horizontal movement (leaving the vertical enabled). In the other, we completely disable one or two legs by making their horizontal movement zero. Tests are done starting with the healthy population where a tripod gait is being produced and with new random populations where the CGA must evolve the gait from scratch. The results confirm the adaptability of the algorithm on simulations and on the actual hexapod robot.

ROBOT AND MODEL

The robot used was the ServoBot, which is a small, inexpensive hexapod robot that was developed for individual robot and colony experimentation. Control is transmitted through a cable connected to a Sparc workstation. Once the sequence of activations is transmitted, the cable can be disconnected allowing autonomous movement. The control sequence consists of a list of activations, which the on-board controller will continually repeat. Each activation controls the instantaneous movement of the 12 servo actuators. For example: an activation of 001000000000 results in the lifting of the left front leg; 000001000000 results in the pulling back of the second right leg. 001001000000 would activate both at the same time. Activations are held active by the controller for 100 msec. A repeated sequence of these activations can be evolved by a CGA so that the robot walks forward.

The model (Figure 1) was a data structure that holds the essential information needed to determine the state of the legs and the subsequent movement calculated from the control activation input. Fields for each leg were included to store the leg's capabilities and current position. To determine each leg's resting vertical position, the robot was placed on a level surface. Legs touching the ground were at position 0; all others were relative to where their 0 position would be if they were on the ground. All future references to each leg's vertical position were relative to this initial 0 position. Each leg's horizontal position was measured relative to its at rest full forward position.

Model data structure fields for each leg:
current up -- current vertical position of leg
max up -- position off ground when full up
max down -- position off ground when full down
current back -- current horizontal position of leg
max back -- position relative to full forward when full back
Fields applicable to all legs:
rate up/down -- rate of vertical movement when actuator excited/relaxed
rate back/forward -- rate of horizontal movement when actuator excited/relaxed

Figure 1: Model Data Structure

Measurements to fill the position fields were taken by activating each control individually and recording the leg's maximum throw. An average rate per activation was calculated for horizontal and vertical movement by dividing the maximum throw by the minimum number of activations required to attain it.

CYCLIC GENETIC ALGORITHMS

Cyclic Genetic Algorithms [4] solve the problem of evolving repetitive behavior that requires continual cycles of sequential actions. They are based on Genetic Algorithms, which were introduced by John Holland [6], and use the standard selection, crossover and mutation operators. They differ in that the CGA chromosome has a cycle of genes that can accommodate the cyclic nature of gaits. In addition, the genes can represent tasks that must be completed in a predetermined segment of time. For our purposes the tasks to be completed are a sequence of activations to the robot that will result in a reasonable gait.

The chromosome structure used for evolution was made up of four parts (Figure 2). The coordinators affected individual legs. The back-down coordinator ensured the leg was down if moving back. The up-forward ensured that the leg was moving forward if it was up. A 12-bit number was used to represent the coordinators as there were two possible per leg. The inhibitors prevented designated pairs of legs from being moved back or forward at the same time. A single 15-bit number represented the inhibitors. Each bit position corresponded to a specific pair of legs (15 possible). If the bit was 1 it did not allow both legs to move up at the same time even if the activation commanded it. The lower numbered leg could move while the higher numbered leg was inhibited.

The start-section was the part of the chromosome that held the activations that were executed only once. The purpose of this section was to allow the robot to transition from a standing state to a state that was part of the gait cycle. The iterative-section was repeated as many times as desired. This section was designed to form a sequence of instructions that when repeated would result in sustained cyclic behavior. The start-section was made up of a single gene and the iterative-section was made up of 12 genes.

The genes had 2 parts (Figure 2). The *activations* part was a 12-bit number that contained the encoding required to activate two possible primitives per leg. One controlled whether the servo moved the leg up or down, the other controlled whether the servo moved the leg back or forward. The *moves* part was an 8-bit number that designated the number of times to repeat the *activations* part. This *moves* part was what gave the CGA the ability to vary the length of the sequence of primitives being sent to the robot in each cycle.

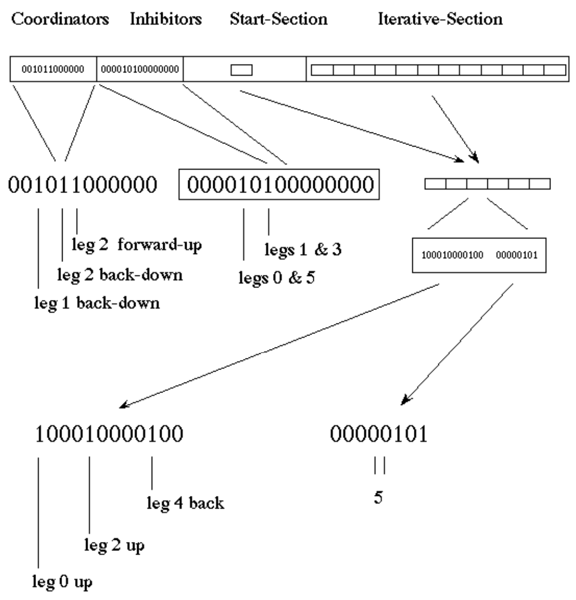

Figure 2: Cyclic Genetic Algorithm Chromosome and Gene Breakdown

The conversion from chromosome to a set of primitive commands went as follows:

1. i_act = apply inhibitors to each activation.
2. ci_act = apply coordinators to each i_act.
3. Write the ci_act from the start-section *moves* number of times.
4. Write a marker separating the start and iterative sections.
5. Write each ci_act from the iterative-section *moves* number of times.

The result was a list of primitive activations separated by a marker that could be put in a file. The robot control program ran through the list of primitives before the marker on startup sending each activation for 100 msec. The list of primitives following the marker was repeated a specified number of iterations. Again each activation was in effect for 100 msec.

Genetic Operators

Pairs of individuals were stochastically selected for reproduction. Their probability for selection was determined by the individual's fitness. It was computed one activation at a time by summing the fitness of individual activations as each is applied to the current state of the simulation. Due to the ServoBot's speed of movement, it was considered viable to maintain balance using dynamic stability. Meaning that the model was considered to be marginally balanced even though there were insufficient legs on the ground to maintain static stability.

Two types of crossover were used. Chromosome crossover was performed at two points between the genes in the iterative section resulting in section swaps between the two cycles. Gene-by-gene crossover allowed corresponding genes to swap encoded information. Crosses could happen between the individual members of the list or within the bits of the specific numbers in the list. This was done in both the start and iterative sections.

Mutation also had two operators, each of which had minimal probability of occurrence. Gene-replace deleted the old gene (*moves* and *activations*) and replaced it with an entirely new one. Gene-mutate changed only one bit in the gene.

An additional genetic operator called the Gene-by-gene evaluator was also used. This operator would randomly pick one or two individuals from the population and increase or decrease all their *moves* by one or evaluate the individual one gene at a time. It removed genes that were significantly worse than preceding genes and reduced the *moves* of genes that had productive activations initially but dropped in effectiveness after some repetitions. It also moved genes with a *moves* number of 0 to the end so that the active genes were always at the start of the iterative section. The result of this operator was to clean up the chromosomes and speed up the elimination of poor *activations*.

RESULTS

Tests were done on pre-trained and random start populations of 64 individuals to determine if they could adapt to reduced capabilities in one or two legs. The one leg experiments, shown in Figure 3, were done using a disabled leg 0 (right front). The "No Leg 0" pair of columns shows the model fitnesses before and after 1000 generations of training. The "Fully Capable" pair of columns shows the model fitnesses before and after 500 generations of training where an optimal tripod gait is produced. Subsequent pairs of columns show the result of reducing the leg capability to the specified percentage. From "Fully Capable" to "0%" the column pairs show the result of an incremental decrease in the capability of leg 0.

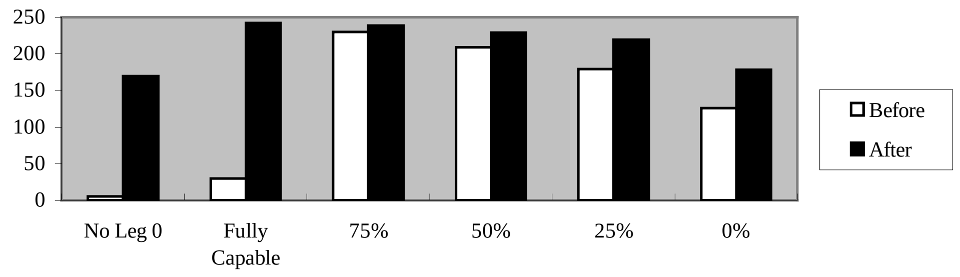

Figure 3: CGA Responses to Leg 0 Disabilities

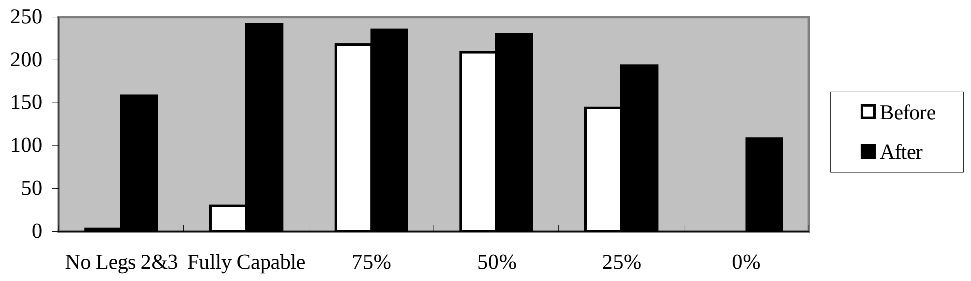

Figure 4: CGA Responses to Leg 2 & 3 Disabilities

The "Before" column shows the model's fitness immediately after the decrease and the "After" column shows it after 500 generations of additional training. In each case, the CGA compensates for the loss and improves the gait. Figure 4 shows the two legs experiments, which were done using disabled legs 2 and 3 (center legs). The generations used were the same as for Figure 3.

Tests on the actual robot confirmed the viability of the produced gaits. Figure 5 shows the distances (in centimeters) traveled by the robot in 30 sec. The Leg0 25% was a surprise as the model predicted that it would be better. The model shortened the stride sufficiently to equal the reduced capability of leg 0. The model could operate efficiently with this reduced stride, but the robot couldn't. Alterations of the model will be required to take this into account. NoLeg23 developed a dynamic gait using alternative outside legs. Legs 0 and 5 repositioned while legs 1 and 4 provided power; this was very successful on the actual robot.

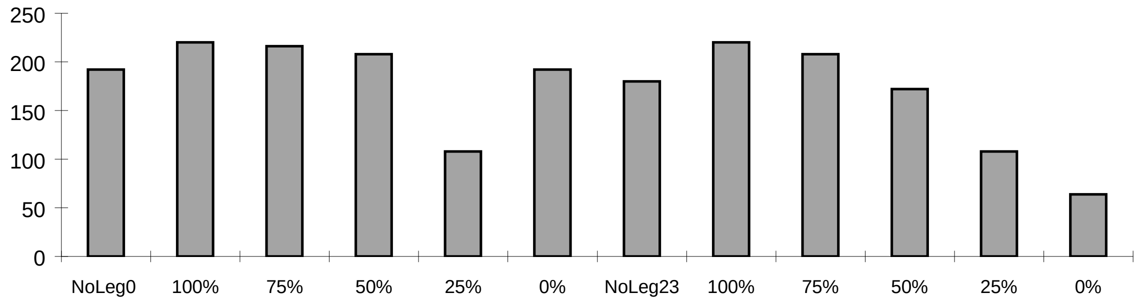

Figure 5: Centimeters Traveled by ServoBot in 30 seconds

These results indicate that the Cyclic Genetic Algorithm is an effective means of producing and maintaining gaits for actual robots. Further research will continue to pursue the viability of CGAs in solving the cyclic control problems of robots.

ACKNOWLEDGMENTS

This research was supported in part by NSF Graduate Research Traineeship Grant GER93-54898.

REFERENCES

1. 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.
2. Beer, R. D., and Gallagher, J. C. (1992). "Evolving Dynamical Neural Networks for Adaptive Behavior." Adaptive Behavior, 1 (pp. 91-122). Cambridge: MIT Press.
3. 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.
4. 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)
5. 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).
6. Holland, J. H. (1975). Adaption in Natural and Artificial Systems. Ann Arbor, Mi: The University of Michigan Press.