EVOLVING LEG CYCLES TO PRODUCE HEXAPOD GAITS

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

ABSTRACT

Gait generation for hexapod robots is accomplished by dividing the problem into two parts: leg cycle learning and gait cycle learning. Servo pulses required to generate a single leg cycle are learned by a cyclic genetic algorithm. This learning takes into account the peculiarities of the leg's capabilities plus determines the proper sequence of pulses needed to generate smooth movement by the servos. The best means of combining these leg cycles into a gait cycle is learned by a genetic algorithm. 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 a simulated hexapod robot.

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

INTRODUCTION

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 [2,3]. This method can produce reasonable gaits that can operate on simpler controllers, but they cannot fully exploit the capabilities of the robot. Learning the leg cycles separately from their coordination allows the system to better use each leg as long as the controllers are complex enough to handle the increased details. In our work, we use robots that have servomotors 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 is capable in the time between pulses. A sequence of pulses with small changes in pulse length is required to get rate control.

Some form of evolutionary computation (EC) would work well in learning what signals are needed for the leg cycle since EC is well suited for adapting a solution to the peculiarities of a problem. The difficulty comes in that most forms of evolutionary computation are not naturally equipped to handle the cyclic nature of these leg cycles. Graham Spencer [4] 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 on robot simulations, resulted in gaits that maintained sustained forward movement but could not obtain the optimal tripod gait.

Beer and Gallagher [1] used genetic algorithms to develop neural network controllers for a hexapod robot. To do separate learning, they used a single leg controller network to learn the best leg cycle. The fitness was computed by measuring the forward velocity of the leg while it was on the ground. To solve the gait generation problem, six copies of the evolved leg cycle were used to evolve the connections between the legs.

In previous work [2,3], we used Cyclic Genetic Algorithms (CGAs) to generate the sequence of primitive instructions used by a single BASIC Stamp II (sold by Parallax, Inc.) to produce a gait. Tests showed that CGAs could produce tripod gaits on robot simulations that were transferable to an actual autonomous hexapod robot, but 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 controller.

In this paper, we use 6 additional BASIC Stamp IIs (one for each leg) to take better advantage of the capabilities of the servos. Each leg stamp 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 used by a standard genetic algorithm to produce gaits for the robot. Tests confirm the viability of this method for producing gaits.

SERVOBOT

This learning method was designed to work on the ServoBot robot. 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.

Control for the ServoBot will be provided by 7 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.

CYCLIC GENETIC ALGORITHMS TO PRODUCE LEG CYCLES

Cyclic Genetic Algorithms were developed [2] 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 CGA chromosome can also have two trails, which are provided to allow for pre and post-cycle procedure; they provide a means for completing tasks before and after entering the cycle. For the generation of a sequence of servo pulses that can be repeated to generate a leg cycle, the pre and post-cycles were not used. 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 moving the leg from its current position to a new position. 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 within 20 microsecond 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 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 effect the smoothness of the robot's movement over the ground.

((R1 HP1 VP1) (R2 HP2 VP2) (R3 HP3 VP3) (R4 HP4 VP4) (R5 HP5 VP5) ... (R8 HP8 VP8))

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.

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 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.

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. The standard CGA genetic operators [2,3], including crossover and mutation, were used.

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 result of this training, done for each leg, is shown in Figure 2. 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.

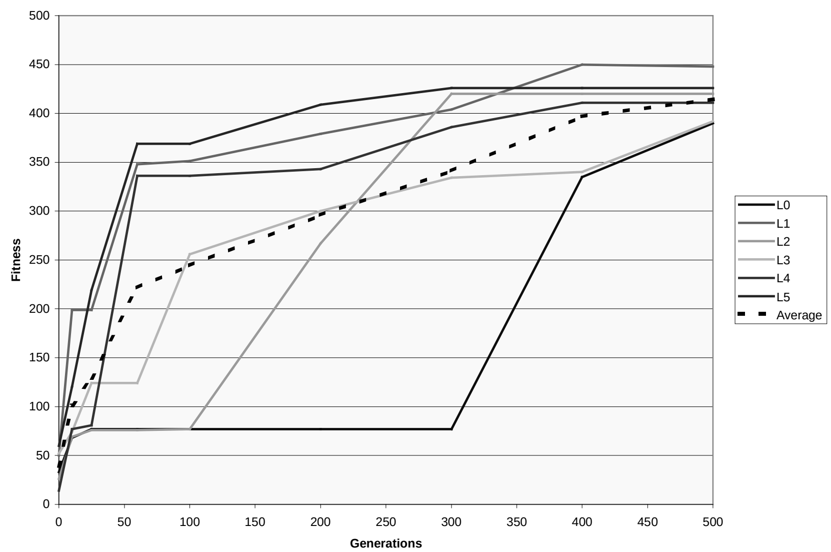

Figure 2. Single leg training for each leg.

GAIT CYCLE GENERATION

Leg cycle training was repeated, in preparation for gait training, 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. The length used was 36 pulses, which was the average maximum optimal length found in the previous test.

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 was made up of 7 parts. One part was a single integer that represented the number of pulses in a gait cycle. The six other parts each designated the leg cycle to be used and its start time for each of the six legs. Start time was the gait cycle count where the leg cycle should begin. Once the gait cycle started, it would continually cycle.

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 began 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 combined to be running simultaneously to determine at each pulse what the result of the 6 leg pulses would be.

Five populations, each having 64 randomly generated chromosomes, were produced to start training, which was done for 500 generations. The GA quickly learned adequate coordination within 100 generations (fitnesses improved from an average of 300 to an average of 1200). After that, the GA worked to improve this solution to find the optimal leg cycle lengths and starting times for each leg. The average fitness after 500 generations was 1340; in all 5 cases, near optimal tripod gaits are produced.

CONCLUSIONS

CGAs are an effective means of generating the cycles of pulses required to produce a leg cycle for a two servo leg. Tests in simulation showed that they improve leg performance significantly during training. These leg cycles can then 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.

ACKNOWLEDGMENTS

A portion of this research was performed at Indiana University's Adaptive Systems Laboratory, supported in part by NSF Graduate Research Traineeship Grant GER93-54898.

REFERENCES

1. Beer, R. D., and Gallagher, J. C. "Evolving Dynamical Neural Networks for Adaptive Behavior." *Adaptive Behavior*, 1 (1992), 91-122.
2. Parker, G. and Rawlins, G. "Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots." *Proceedings of the World Automation Congress (WAC'96), Volume 3, Robotic and Manufacturing Systems*. (1996), 617-622.
3. Parker, G., Braun, D., and Cyliax I. "Evolving Hexapod Gaits Using a Cyclic Genetic Algorithm." *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC'97)*. (1997), 141-144.
4. Spencer, G. "Automatic Generation of Programs for Crawling and Walking." *Advances in Genetic Programming*. K. Kinnear, Jr. (ed.), Cambridge, Ma: MIT Press. (1994), 335-353.