Evolving Cyclic Control for a Hexapod Robot Performing Area Coverage

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

Abstract

For a robot to search an entire area, it must follow a path that allows the range of its sensors to cover all parts of the area. This problem is a subset of path planning called area coverage. Most work done in this type of path planning has concentrated on ways of dividing the area up to avoid obstacles while covering the area. This is an important step in the process, but often takes for granted the movement of the robot within clear areas. This is not a problem if the robot has sufficient calibration to ensure the accuracy of calculated turns or if it has accurate enough navigational devices to keep track of its location. However, simple legged robots usually lack both of these attributes. It is difficult to make turns that fit a specified arch and sufficient on board navigational devices are expensive and/or too large to carry. In this paper, we use cyclic genetic algorithms to learn the control cycles required to make an actual hexapod robot perform area coverage.

1. Introduction

Path planning is the formation of the set of moves that the robot will take to transport it from a starting point/configuration to a goal point/configuration. The complexity of the computation of this series of moves is reduced when the environment is known and the robot is accurate enough so that the exact results of control manipulations of its actuators can be predicted. In the optimal situation, the actuators will have a direct and precise effect on the position and orientation of the robot. Such is the case when one is dealing with wheeled robots that have calibrated movement control. Typically these robots can be commanded to rotate *k* degrees and move forward *x* millimeters. This level of control not only helps in dead reckoning because the robot's location is precisely known, but it also helps in making plans that require precise maneuvers such as a 180° turn.

Area coverage is a type of path planning that is concerned with the coverage of an area. This may be done by an insect to search for food or check for enemies. A robot could also be searching for enemies or clearing the area of deadly devices such as mines. The robot's sensors (manipulators) are assumed to have a certain width of effectiveness and the area is described as having defined boundaries and possibly some obstacles. The path planned is supposed to ensure that the area covered by the robot's sensors compared to the total area within the defined boundaries is equal to the desired coverage. In most cases the desired coverage is 100%, but due to the decreasing effectiveness of sensors as the distance increases from the robot, this exact percentage is seldom attained. The desired coverage is therefore often described in some other way such as separation of paths or attainment of fixed blocks of space distributed throughout the area.

Previous research in the area of coverage path planning concentrates primarily on covering a specified area while contending with obstacle avoidance. Zelinsky et al. [8] used an extension to a path planning methodology, which divided the area into cells that were marked with the distance to the goal, to form a cell to cell path through the area. Tests were done on the Yamabico robot, which used precise movement calculations for dead reckoning and ultrasonic sensor sightings of landmarks to re-adjust its position. Choset and Pignon [1] divided the area into obstacle free sub-areas (they called cells) and found an exhaustive path through the adjacency graph representing these cells. Within each cell the back-and-forth boustrophedic motions (Figure 1) were used to assure coverage. Tests in simulation worked well. Tests done on a Nomadic 200 mobile robot base using only dead reckoning resulted in tracks with lines that were not perfectly parallel. Ollis and Stentz [4] used vision to control the lines in their boustrophedic motions to do automated harvesting. The Demeter harvester robot used a GPS system and dead reckoning, in addition to the vision system, to ensure field coverage. Hofner and Schmidt [3] used templates appropriate for the type of robot to determine the best path within areas that Choset would call cells. This allowed for other than boustrophedic motions if the robot lacked the capability to turn that sharply. Robot dead reckoning was supplemented by an extensive use of landmarks sensed by ultrasonic sensors to maintain the desired track. Hert at el. [2] proposed a method for sea floor coverage by an autonomous underwater vehicle, which used an on-line planar algorithm and sensors to explore areas with arbitrary shape. The track over the ground was done with back-and-forth boustrophedic motions controlled by a GPS system.

Common to all of these works is either precise control of the robot or some navigational means of making continual corrections to its movement. These assets are not always available. Legged robots, especially inexpensive ones with minimal sensors and precision of movement, cannot be positioned perfectly with exact headings. What is often taken for granted in these papers, a capability to perform perfect back-and-forth boustrophedic motions (Figure 2), is not easily done by these legged robots. Whereas calibrated wheeled robots can be commanded to turn in a circular arc for *n* msecs or rotate to *k* degrees, simple legged robots can only turn for so many gait cycles with one of a variety of turns. The exact time and rate cannot be specified. In addition, often what is considered a straight gait results in a small drift to one side or the other due to performance differences from one side to the other of the robot. The best straight may actually be what is programmed in as a minimal turn.

An additional factor driving this research is that differing turn rates can have vastly different efficiencies. This is especially true for legged robots, but is also applicable to most wheeled robots. The most efficient path to cover the area given depends greatly on the capabilities of the robot. If it can efficiently rotate or turn sharply, its best strategy may be to do a ladder pattern (boustrophedic with square turns). If tight turns cost in efficiency, it may be better to make large sweeps with some coverage overlap or to buttonhook the ladder turns, keeping the rungs about the same.

In this paper, we use an actual hexapod robot to test this method for learning turn cycles that will produce the tracks required for area coverage. The method involves the following steps. Use a cyclic genetic algorithm (a form of evolutionary computation designed to learn cycles of behavior) to learn a near optimal gait for the hexapod robot. Alter this gait in such a way that several degrees of turn are produced. Run tests on these turn programs to measure the rate of turn in a single complete step (a gait cycle). Use these gait cycles as available tasks that a CGA can use in simulation to find the control sequence required to perform area coverage. And finally, test the control programs on the actual robot. Previous work, done solely in simulation, showed that this method can be used to learn the cycle of turns and straights to do area coverage within cells. These tests produced promising results and helped to confirm the usefulness of CGAs for learning control cycles to do area coverage. Tests presented in this paper show that CGAs can effectively learn control programs for actual hexapod robots to do area coverage.

2. The Robot

The robot used was the ServoBot (Figure 1), which was developed at Indiana University by David Braun for legged robot and colony experimentation. It is a small, inexpensive hexapod robot that has many of the motion characteristics of larger complex robots. The controller used was a basic stamp, which can hold a normal gait, affecters that can produce a turn from this gait, and a sequence of commands that indicate what affecters should be invoked. The control program can be downloaded to the basic stamp through a connecting cable or transmitted via radio waves from an external computer running the learning system. Once the sequence of activations is transmitted, the cable can be disconnected allowing autonomous movement.

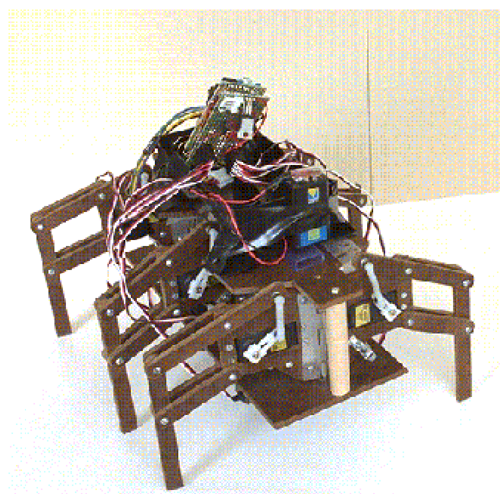

Figure 1: ServoBot

The normal gait is made up of a control sequence, which is a list of activations that the on-board controller will continually repeat. Each activation (a 12 bit number) controls the instantaneous movement of the 12 servo actuators. A bit value of 0 drives the corresponding servo toward its full throw in one direction, a bit value of 1 drives it toward its full throw in the other direction. A repeated sequence of these activations can be evolved by a cyclic genetic algorithm (Section 4) to produce an optimal gait for a specific SevroBot [6] to move straight ahead. The execution of this sequence results in a full cycle of leg movements with the legs returning to their start positions. We refer to this cycle as a gait cycle. The gait cycle for the robot we used was 58 activations (servo pulses).

Differing degrees of turn were provided in the gait cycles through the use of *affecters*. These affecters could interrupt activations to the thrust actuators for either the left or right side of the robot. Since the normal gait consisted of a sequence of 29 servo pulses to move the leg from the full front to full back position, anything less than 29 would result in some dragging of the legs on that side. Affecters from 0 to 15 were possible. A one bit indicator specified if the affecter was right or left. A 4 bit number indicated its strength. The number of pulses on each side could be calculated by multiplying the strength of the affecter by 2 (except that an affecter of 15 yields 29 pulses resulting in no turn). Sixty gait cycles to produce turns and straights were created by assigning an affecter, which resulted in a turn throughout that cycle.

The controller was programmed to make the turns specified in an input sequence by application of the affecters to produce the corresponding gait cycle. The input sequence included the turn direction, turn strength (affecter), and the number of times to repeat that gait cycle. Up to nine changes in gait cycles could be used with up to 63 repetitions of that gait cycle. The effective result was to produce cycles of gait cycles that could be used to define a desired path over the ground.

3. Coverage Area

During area coverage the robot is trying to maximize the area covered in minimal time. For our area coverage problem we wanted the robot to fully search, starting from a specific point, an area of specific width (180 cm). Since the robot was judged by the area covered in a set amount of time, plus we wanted to find the most efficient cycles of behavior required to do it, the area to be searched had no bound on one side. The area width was purposely small to accommodate ease in actual testing and to force more turns during training.

The simulated search was for mines that would be fully contained in the area. In order to detect a mine, the robot had to have the entire width of its body (excluding the legs), at its mid point, within the same 60x60 cm square as the mine. For test purposes, 60x60 blocks with mines were placed to completely fill the area. The robot's task was to find as many mines as possible while ensuring that no mines had been missed. The robot's movement was not restrained in any way by the environment. There was no physical constraint requiring it to stay within the mine area.

Figure 2 shows the simulated search area, which was $180\times\infty$ cm. The mine blocks are shown as squares within the area. The actual test area was a 180×210 cm area of the floor where the 60×60 mine blocks were marked by tape. Tests in this area were done by observing the squares covered as the robot completed the first 50 gait generations of its search path. For the simulation, the robot's track was recorded in a list of calculated positions over the ground. Mine detection was calculated from these positions.

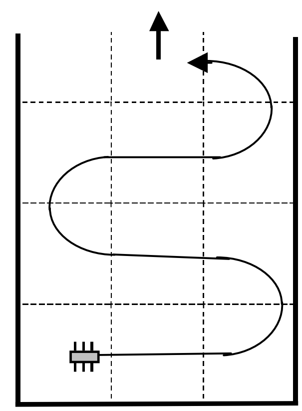

Figure 2: Search area for coverage with a back-and-forth boustropedic pattern of motion.

4. Cyclic Genetic Algorithms

Cyclic genetic algorithms differ from the standard GA in that the chromosome can be thought of as a circle and the genes can represent subtasks that are to be completed in a predetermined segment of time. For the area coverage problem the genes represented a set of gait cycles that were to be sustained for one cycle each. The trained chromosome contained the cycle of these gait cycles that was continually repeated by our robot's controller to produce a path that was to efficiently cover the designated area.

The controller program has a provision for nine changing gait cycles in the search cycle. Each gait cycle takes 5 bits to identify and the repetitions of each gait cycle can be from 0 to 63. The CGA chromosome used directly resembled the required input to the controller. Each chromosome was made up of nine genes and each gene of the chromosome was made up of 2 parts (a 5 bit number to identify the gait cycle and a 6 bit number to specify the repetitions).

Selection probability was determined by the individual's fitness (Figure 3). This fitness was calculated by counting the number of mines detected (mine blocks covered by robot’s path) after it had completed a specified number of gait cycles. Counting, which was not done until the search path was completed, began by rows from the bottom of the area. As soon as a mine block was missed no more rows were counted, although the mines from the partial row were counted. For the fitnesses calculated during training, mines visited more than once were not counted, although they did count for row completions. This was to discourage paths that were wasting time re-searching covered area. Once a fitness was calculated for each individual in the population, pairs were stochastically selected for reproduction.

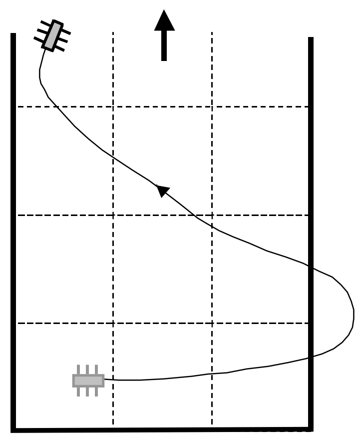

Figure 3: Fitness Calculation. This track would result in a fitness of 4. The first row of mines, plus the second row’s right mine are covered. The second row’s middle mine is not considered covered because the robot’s body at the midsection was not entirely within the block. The third row middle mine would be detected by the robot, but will not count since there were mines in the second row that were not covered. Also note that the robot is free to depart the area with no direct fitness penalty.

The details of the genetic operators used for reproduction are covered in previous work [7]. Two point crossover was performed by randomly picking corresponding spots in the two selected parents. An alternate type of crossover was a gene-by-gene crossover that performs crossover within each of the corresponding genes of the two chromosomes. One of two types of mutation were possible; each gene had a random chance of being replaced by a new random gene or 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 that performed poorly in their current position were eliminated and genes that were good in the execution of their early repetitions and subsequently dropped in the later repetitions were modified by reducing their repetitions. Momentarily Elevated Mutation was initiated whenever it was determined that the CGA had converged prematurely. With the assumption that many of the population’s individuals were the same, each individual in the population went through mutation at a rate more than usual. The idea was to introduce several small changes while retaining important sub-solutions within the population. Although high mutation can be very disruptive to each individual, with enough identical individuals the good building blocks should survive and provide a chance of generating a superior individual due to the introduced perturbations.

5. Training

Training was done to find the best search path for our specific robot. The robot's base gait cycle was learned using a CGA that was optimizing for speed. 15 left and 15 right gait cycles were programmed using the affecters described in section 2.3. Each gait cycle was tested for rate of turn by running the robot for 4 cycles while taking three measurements (Figure 4).

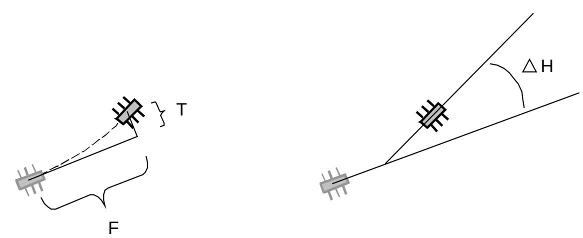

Figure 4: Gait Cycle Turn Measurements. The left diagram shows *F* and *T*. *F* is the distance moved forward (relative to the start position heading). T is the distance moved in the turn direction (perpendicular to the start position heading). The right diagram shows ΔH, which is the change in heading from before to after turn execution.

The test area (Figure 2) was simulated by an *xy* grid where point (0,0) was the lower left corner. The lower right corner of the area was the point (180,0). The lower boundary was at y = 0, the left boundary was at x = 0, the right boundary was at x = 180, and there was no upper boundary. Mines were considered to be in 60×60 square blocks. The first row had centers at (30,30), (90,30), and (150,30). The second row started at (90,30), etc. The robot’s start position was placed at (45,30). This location assured acquisition of the first mine and put it in a good starting place to acquire the first row of mines. Motion was determined by applying each gait cycle from the chromosome one at a time. Using the current *xy* position and heading of the robot, a new position was calculated by applying the forward (*F*) and left/right (*T*) movements stored for that gait cycle as described in the previous section. The new heading was an addition of the current heading and the gait cycle heading change (ΔH). The path was not restricted from going outside of the area and the calculations remained the same if it did. This allowed, if appropriate, for the robot to do its turns out of the area so that it could attempt straight tracks within the area.

An initial population of 64 individuals, made up of chromosomes described in section 4.1, were randomly generated. Each individual, representing a cycle of gait cycles that would form a path, was tested to determine its fitness after 100 to 200 of these gait cycles were executed. The CGA was run for 5000 generations with the best solution (individual chromosome) saved whenever there was an increase in fitness (more mines covered in the allotted time).

6. Results

Five initially random populations were each trained using the CGA. Tests were done on the individuals saved during training to record the progress of the best individual in solving the area coverage problem. The average of the best fitnesses for each recorded generation from the five populations was then calculated. The final solution from each population was tested on the robot to check the correspondence of the simulation to the actual robot. In addition, all intermediate solutions of the best population were tested on the actual robot. The tests on the robot, limited by time and space, were done for 50 gait cycles.

The progression of learning as the CGA evolved solutions in simulation for the five populations had an initial growth was relatively fast [7]. The average blocks covered went from 3 to 23 within 500 generations. In this area, the CGA was evolving the basic back-and-forth boustrophedic pattern. The fitness growth plateaued, however, because the resultant solution for some of the populations was close but still needed some small change. Small adjustments at this point were required to get that perfect amount of turn or alignment, but these small adjustments could also result in changes that caused major drops in fitness. Consequently, the learning was slow until just the right set of mutations was recombined to get a good solution. Of the five starting populations, all but one made this final adjustment within 5000 generations.

Two tests were done on the actual robot. Using five intermediate solutions where increases in fitness took place (at 0, 9, 35, 151, & 217 generations) during the training of the population with the best resultant solution, a comparison was made between the model and the actual robot (fitnesses averaged over five trails). Figure 5 shows this comparison, which was done using 50 gait cycles on both the model and the actual robot. The robot, being slightly off from the model, did better than the model predicted during the low fitness runs. The robot had a little stronger left drift than initially calculated. This, in combination with the non-deterministic nature of actual robots, resulted in a 0-generation path that ran straight out of the area (as the model's path did), but sometimes covered the right mine of the second row on its way. The 9-generation solution also gained by the inaccuracy in the robot, but not to the same extent. For the 35-generation solution, it made no difference. From then on, the inaccuracy caused decreasing performance in the actual robot over the model, although the error still remained within 10%.

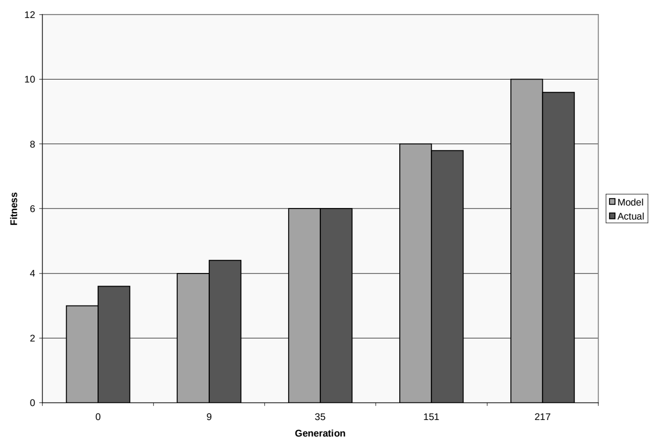

Figure 5: Comparison of the model to the actual robot's fitness (averaged over 5 trials) for five levels of training in the evolution of the population with the best resultant coverage solution.

The best solutions from all five populations were also tested on the actual robot. Figure 6 shows the results of these tests. In most cases the actual robot's performance was not as good as predicted due to the slight inaccuracy in its turn capabilities. In one case (P4), where the training on the model fell short of the optimal after 5000 generations, the inaccuracy helped. The generated path had tighter right turns, which caused it to waste time in gait cycles that had minimal distance coverage, plus resulted in some shift during simulation. Its fitness after 50 generations was 7 using the model. Since the actual robot had more left drift than was unaccounted for, a tighter programmed right turn resulted in a correct right turn, which improved efficiency plus reduced the drift. The P3 and P5 solutions worked well in early tests. Their paths were such that the model could gain a fitness of 9 in 50 gait cycles, but the left mine of the second row was just barely covered. In later tests, the robot started to just barely miss this block, resulting in fitnesses as low as 5. It was assumed that the robot's performance changed slightly due to normal wear. The solutions from populations P1 and P2 worked well and were consistent (within +/- 1 block) on the actual robot.

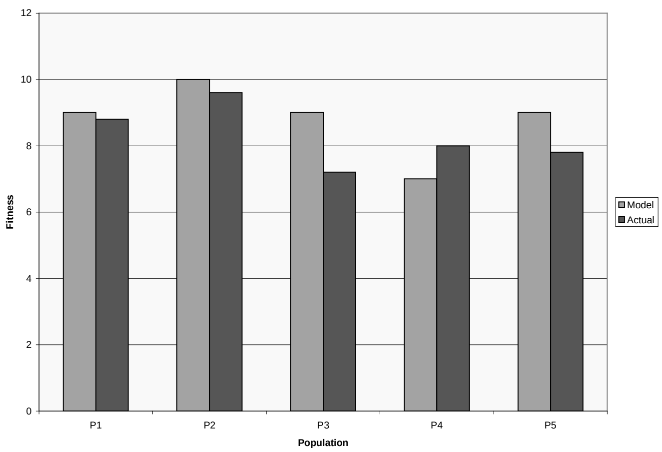

Figure 6: Comparison of the model to the actual robot's fitness (averaged over 5 trials) for the final solution of each population.

7. Conclusions

All five populations resulted in a search pattern that was similar to the back-and-forth boustrophedic pattern. Several factors made it unlikely that the pattern would be a perfect back-and-forth boustrophedic pattern. This pattern requires that a cycle of straight/left turn/straight/right turn be continually repeated. Any deviation from the initial robot heading and *x* position after subsequent cycles will continually cause the pattern to drift out of the area. Having a non-standard array of possible turns makes it difficult to get the right combination to get back to this initial heading and *x* position. While doing so, it is also important that the *y* distance from the start position of one cycle to the next be two blocks in width. Another difficulty is that there are no straight gait cycles; all have some left or right drift. This forces the learning system to either replace the straight leg with a constant minimally arching turn and correcting the angles during the turnarounds or by continual compensations of alternating left and right turns. The lack of symmetry in turning capabilities also causes a problem. Having symmetric turn capabilities would allow the robot to repeat the same pattern as it makes its left and right turns. This would be especially effective if a set of its turns could generate a 180° turnaround. Our legged robot had neither symmetric turns nor a set of turns all in one direction that would equal 180°.

Given all these negative factors, there was much work to be done by the CGA to generate the back-and-forth boustrophedic pattern. All five populations had unique solutions and four of the five covered 30 or more blocks in 200 gait cycles. The remaining solution was still a back-and-forth boustrophedic pattern, but having sharper turns than necessary, tended to lack efficiency resulting in only 22 blocks being covered. We did not succeed in manually programming a full solution for comparison purposes. The number of possible variances made the process very time consuming. The P2 solution, however, with a fitness of 34, looked as if it was almost optimal for 200 gait cycles. Four of the five tests resulted in near optimal solutions. The CGA worked well in producing programs that controlled our actual hexapod robot performing area coverage.

8. References

1. H. Choset and P. Pignon, (1998). "Coverage Path Planning: The Boustrophedon Cellular Decomposition."
2. C. Hofner and G. Schmidt, (1995). "Path planning and guidance techniques for autonomous mobile cleaning robot." Robotics and Autonomous Systems, 14.
3. S. Hert, S. Tiwari, and V. Lumelsky, (1996). "A Terrain-Covering Algorithm for an Autonomous Underwater Vehicle." *Journal of Autonomous Robots*, (Spec. Issue on Underwater Robotics), 3, (pp. 91-119).
4. M. Ollis and A. Stentz, (1997). "Vision-Based Perception for an Autonomous Harvester." *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotic Systems*.
5. G. Parker and G. Rawlins, (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).
6. G. Parker, D. Braun, and I. Cyliax, (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).
7. G. Parker (2001). "Learning Control Cycles for Area Coverage with Cyclic Genetic Algorithms." *Advances in Fuzzy Systems and Evolutionary Computation. Proceedings of the 2nd WSES International Conference on Evolutionary Computation (EC '01)*. (pp. 283-289).
8. A. Zelinsky, R. Jarvis, J. Byrne, and S. Yuta, (1993). "Planning Paths of Complete Coverage of an Unstructured environment by a Mobile Robot." *Proceedings of International conference on Advanced Robotics*. (pp. 533-538).