Learning Control Cycles for Area Coverage with Cyclic Genetic Algorithms

GARY B. PARKER
Computer Science
Connecticut College
New London, CT 06320
U.S.A
parker@conncoll.edu http://cs.conncoll.edu/parker

Abstract: - Area coverage is a type of path planning that is concerned with finding a pattern of movement that will result in coverage of all parts of an area. Most area coverage planning algorithms assume that the robot can maintain a track over the ground that will result in full coverage in obstacle free areas. This is easily done if the robot can be precisely controlled or has sufficient sensor capability to know the relationship of its current track to its previous one, but simple legged robots usually lack both of these attributes. A means of learning the optimal cycle of turns and straights to produce a full coverage track could greatly improve efficiency. In addition, a system of learning could compensate for the lack of calibration in robot turning systems. In this paper, we introduce a method for learning turn cycles that will produce the tracks required for area coverage. The learning is done using a cyclic genetic algorithm (a form of evolutionary computation designed to learn cycles of behavior). Tests of the simulated robot's dead reckoning capabilities with the learned cycles show that using CGAs is an effective means of learning for area coverage.

Key-Words: - genetic algorithms, path planning, area coverage, hexapod, robot

1 Introduction

Path planning is an important aspect of robot navigation. It 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. Area coverage is a type of path planning that is concerned with the coverage of an area. Some applications are mine sweeping, search and rescue, haul inspection, painting, and vacuuming. 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. 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. Ollis and Stentz [4] used vision to control the lines in their boustrophedic motions to do automated harvesting. Hofner and Schmidt [3] used templates appropriate for the type of robot to determine the best path within areas that Choset would call cells. 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 online planar algorithm and sensors to explore areas with arbitrary shape.

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 1), is not easily done by these legged robots. The exact time and rate of turn 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. Efficiency is also a factor. 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.

A means of learning the optimal cycle of turns and straights could greatly improve the efficiency of area coverage within cells. In addition, a system of learning could compensate for the lack of calibration in robot turning systems. In this paper, we introduce a method for learning turn cycles that will produce the tracks required for area coverage. The learning is done using a cyclic genetic algorithm (a form of evolutionary computation designed to learn cycles of behavior). Previous work has shown that CGAs can effectively learn gait cycles for legged robots [5,6]. The turn cycles learned for the coverage problem are composed of these gait cycles. Tests of the simulated robot's dead reckoning capabilities with the learned cycles show that using CGAs is an effective means of learning for area coverage.

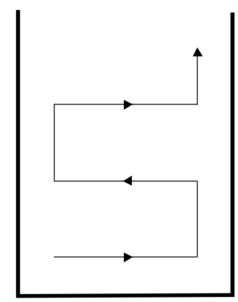

Fig. 1: Back-and-Forth Boustrophedic Motion

2 The Robot Simulation

2.1 ServoBot

The simulation was modeled after the ServoBot, a small, inexpensive hexapod robot with a BASIC Stamp II controller. The controller is capable of holding a normal gait, affecters that can produce a turn from this gait, and a sequence of commands that indicate what affecters should be invoked.

2.2 Gait Cycles

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 controls the instantaneous movement of the 12 servo actuators. The activation can be thought of as 6 pairs of actuations. Each pair is for a single leg with the first bit of the pair being that leg's vertical activation and the second being that leg's horizontal activation. The legs are numbered 0 to 5 with 0,2,4 being on the right from front to back and 1,3,5 being the left legs from front to back. A signal of 1 moves the leg back if it is a horizontal servo and up if it is a vertical servo. A signal of 0 moves it in the opposite direction. 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. This set of input activations is held active by the controller for one servomotor pulse (approximately 25 msec).

A repeated sequence of these activations can be evolved by a cyclic genetic algorithm (Section 3) to produce an optimal gait for a specific SevroBot [7]; the gait generated for our test was a tripod gait (Figure 2). The tripod gait is where legs 0, 3, & 4 alternate with legs 1, 2, & 5 in providing the thrust for forward movement. While one set of legs is providing thrust, the other set is repositioning for its next thrust. In the case of our robot, the entire cycle lasted for 58 activations with each set providing 29 activations of thrust.

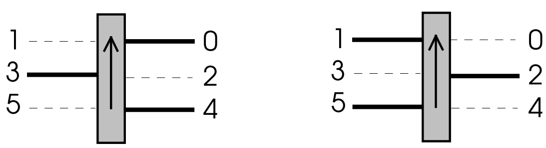

Fig. 2: Tripod Gait. The solid lines show legs that are on the ground and producing thrust. The dashed lines show legs that are repositioning for the next thrust by initially lifting and then lowering again while the leg is moving forward. Legs 0, 3, & 4 alternate with legs 1, 2, & 5 in providing thrust.

2.3 Production of Turns in Gait Cycles

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 pulses of thrust 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. For example: a right side affecter of 7 would allow only 14 (2 x 7) thrusts on the right side while keeping 29 on the left. The result would be that the left side would move further than the right resulting in a right turn. Affecters from 0 to 15 were possible. A one bit indicator specified if the affecter was right or left. A 4 bit number indicated the strength of the effect. 0 meant that side would get no thrust producing a maximum turn. 15 will not effect the normal gait so the result should be a straight track.

Each gait cycle, made up of 58 activations, was assigned an affecter, which resulted in a turn throughout that cycle. For consistency, each gait cycle started with legs 0, 3, & 4 full forward and legs 1, 2, & 5 full back; all the legs were on the ground. As the gait cycle started legs 0, 3, & 4 would provide the thrust as legs 1, 2, & 5 would start to lift and move forward to reposition for their thrust after 29 activations. A single gait cycle was defined as being complete when the legs returned to their starting positions (in this case, after 58 activations).

2.4 Cycles of Gait Cycles

The controller can be programmed to make the turns specified in an input sequence by application of the affecters to produce the corresponding gait cycle. The input sequence includes the turn direction, turn strength (affecter), and the number of times to repeat that gait cycle. Up to nine changes in gait cycles can 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. A cycle of sub-cycles results in a single cyclic behavior.

3 Cyclic Genetic Algorithms

Cyclic genetic algorithms were developed to allow for the representation of a cycle of actions in the chromosome [5,6]. They differ from the standard GA in that the chromosome can be thought of as a circle with up to two tails (Figure 3) and the genes can represent subtasks that are to be completed in a predetermined segment of time. The tails of the CGA chromosome allow for transitional procedures before and/or after the cycle, if required. In our area coverage experiments, we used only the cyclic portion since the start position was known and the search tactic was to be applicable for any duration. The CGA genes can be one of several possibilities. They can be as simple as primitive subtasks (activations) or they can be as complicated as cyclic sub-chromosomes that can be trained separately by a CGA. 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.

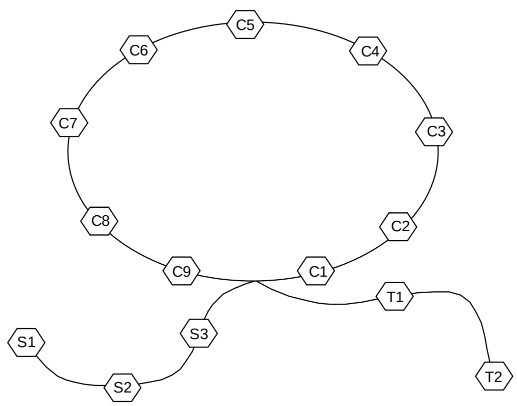

Fig. 3: CGA Chromosome with three genes in the start section (before the cycle), nine genes in the cyclic section, and two genes in the tail section (after the cycle).

A number of variations are possible in the character of CGA chromosomes, including fixed and variable lengths, constraint enforcement sub-segments, and so forth. For the area coverage problem, a fixed length chromosome with no constraints was used.

3.1 Area Coverage Chromosome

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 resembles the required input to the controller. Each chromosome is made up of nine genes and each gene of the chromosome is made up of 2 parts (a 5 bit number and a 6 bit number). The scheme representation of the chromosome is shown in Figure 4. The first number in each pair represents the gait cycle while the second represents the number of times to repeat that gait cycle.

((GS_1 R_1) (GS_2 R_2) (GS_3 R_3) (GS_4 R_4) (GS_5 R_5) (GS_6 R_6) (GS_7 R_7) (GS_8 R_8) (GS_9 R_9))

Fig. 4: Area Coverage Chromosome

3.2 Genetic Operators

Selection probability was determined by the individual's fitness. 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.

Crossover was accomplished by randomly picking corresponding spots in the two selected parents. Since the area coverage chromosomes represented cycles, which could be considered a circle, crossover was performed at two points. The effect was to swap sections within the circle. An alternate type of crossover was a gene-by-gene crossover that performs 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.

Two types of mutation were possible (randomly selected) after each recombination. In one, each gene had a random chance of being replaced by a new random gene. In the other, 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 were evaluated move-by-move by comparing the previous move fitness to the present. Fitnesses for this operator were not computed in the same way as for CGA selection. The area covered (assuming a 20 cm sensor coverage width) within the search area, not the number of mines detected, was calculated for each single gait cycle. This differing fitness calculation was necessary because it gave an immediate fitness for each gait cycle. Since search patterns would sometimes take several gait cycles before each mine was detected (even if the search was being done in the most efficient manner), the usefulness of a single cycle would be difficult to predict using the normal CGA (mine blocks covered) fitness calculation. Judging from this area covered fitness, 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. Genes that had zero repetitions were moved out so that only active genes were at the start of the cyclic section.

Momentarily Elevated Mutation was initiated whenever it was determined that the CGA had converged prematurely. This determination was made by comparing all of the population's individual fitnesses to the fitness of its best individual. If over half of the individuals in the population had a fitness equal to the best; it was assumed that the CGA was stuck. At this point, each individual in the population went through mutation at a rate more than usual. CGA training was then continued. With the assumption that many of the population's individuals were the same, 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 somewhere within the population. Continued evolution can use these to rebuild the original individual, but with the added chance of generating a superior individual due to the introduced perturbations.

4 Method

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). The area width was purposely small to force more turns during training. The area to be searched had no bound on one side since the control program 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 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.

4.1 Simulated Robot Performance

Training was done to find the best search path for a 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. The effect of each of these gait cycles was tested on the actual robot and the results used in a simulation with a CGA to generate area coverage search paths.

Each gait cycle was tested for rate of turn by running the robot for 4 cycles while taking three measurements. *F* was the distance in centimeters that it moved forward. The *F* axis was defined as the heading of the robot before movement. *T* was the distance traveled left or right. The *T* axis was defined as a perpendicular to the *F* axis. Left movement resulted in a negative *T*, right in a positive *T*. ΔH was a measurement (in degrees) of the change in heading from the start heading *F* axis to the heading after execution of the gait cycles. Left was negative, right was positive. After making these measurements, each was divided by 4 to attain the average turn rates. The sharpest turns, affecters less than 3, resulted in turns of greater than 90° after 4 gait cycles, so 3 cycles were used in these cases. Turn rates, defined using *F*, *T*, and ΔH; were stored for each gait cycle.

4.2 Simulated Environment

The test area (Figure 1) 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.

5 Tests

An initial population of 64 individuals, made up of chromosomes described in section 3.1, were randomly generated. Each individual, representing a cycle of gait cycles that would form a path, was tested to determine its fitness after 100 of these gait cycles were executed. 100 gait cycles were enough to ensure that some turning was required to cover optimal mines. It was not, however, enough to force the formation of a cycle that could provide continuing full coverage. Due to this limitation, the required number of gait cycles was randomly (with a 1 out of 2 probability) increased to 200. This allowed for faster fitness computations while using 100 gait cycles, yet put selection pressure on the population to evolve individuals capable of performing well at 200 gait cycles. 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). 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.

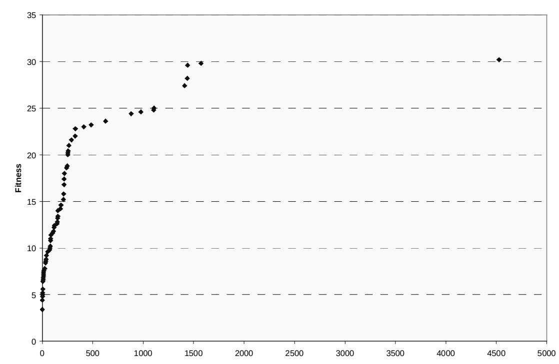

Fig. 5: Average Fitness of Five Populations throughout Training

Figure 5 shows the progression of learning as the CGA evolved solutions to the area coverage problem. As each of the five initial populations evolved, the learning system held the best solution for use by the robot's controller. The average of the fitnesses (mine blocks covered) of these five solutions (one for each population) was computed at each generation. The points on the graph indicate the new average fitness whenever this value changed. As can be observed, the initial growth was relatively fast. The area of the graph where the number of generations is less than 500 has several points, which indicates several fitness increases, and has a steep slope on the learning curve, which indicates quickly increasing fitness. In this area, the CGA was evolving the basic back-and-forth boustrophedic pattern.

The fitness growth plateaus, however, because the resultant solution for some of the populations is close but still needs some small change. It could possibly have tighter turns than required or be slightly out of alignment. These problems can cause lost efficiency (less ground covered in tight turns) and/or the search can eventually drift out of the area. Small adjustments at this point are required to get that perfect amount of turn or alignment, but these small adjustments can also result in changes that cause major drops in fitness. The result is for the individuals in the population to tend toward the local maximum. The probability of a single mutation improving fitness at this point is very low. Mutated individuals, being much less fit, die off almost immediately. It is during this time that momentarily elevated mutation (described in section 3.2) helps breakup the individuals in the population in such a way that the CGA can rebuild them. With all the individuals mutated, they all drop in fitness, putting them on even ground for further evolution. The learning is slow, until just the right set of mutations is recombined to get a good solution. Of the five starting populations, all but one made this final adjustment.

6 Conclusions

The CGA is an effective way of generating the search path required for area coverage. This search path, due to the unbounded nature of our problem, consisted of a single pattern that could be continually repeated. This single pattern (part of the overall search pattern cycle) was made up of several sub-cycles that were generated by the ordering of leg activations by the CGA. The final search pattern was a cycle of sub-cycles generated in a hierarchical fashion by a CGA. This suggests that other problems requiring cyclic behavior, even if recursive, can be handled by hierarchically applying CGAs.

Acknowledgments

Some portion of this research was performed at the Indiana University Adaptive Systems Laboratory and supported in part by NSF Graduate Research Traineeship Grant GER93-54898.

References:

[1] H. Choset and P. Pignon, Coverage Path Planning: The Boustrophedon Cellular Decomposition, *Proceedings of the International Conference on Field and Service Robotics*, 1997.

[2] C. Hofner and G. Schmidt, Path planning and guidance techniques for autonomous mobile cleaning robot, *Robotics and Autonomous Systems*, 14, 1995.

[3] S. Hert, S. Tiwari, and V. Lumelsky, A Terrain-Covering Algorithm for an Autonomous Underwater Vehicle, *Journal of Autonomous Robots*, (Spec. Issue on Underwater Robotics), 3, 1996.

[4] M. Ollis and A. Stentz, Vision-Based Perception for an Autonomous Harvester, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotic Systems*, 1997.

[5] G. Parker, Generating Arachnid Robot Gaits with Cyclic Genetic Algorithms, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 1998.

[6] G. Parker and G. Rawlins, Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots, *Proceedings of the World Automation Congress (WAC'96), Volume 3, Robotic and Manufacturing Systems*, 1996, pp. 617-622.

[7] G. Parker, D. Braun, and I. Cyliax, Evolving Hexapod Gaits Using a Cyclic Genetic Algorithm, *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC'97)*, 1997, pp. 141-144.

[8] A. Zelinsky, R. Jarvis, J. Byrne, and S. Yuta, Planning Paths of Complete Coverage of an Unstructured environment by a Mobile Robot, *Proceedings of International conference on Advanced Robotics*, 1993, pp. 533-538.