10-2007

Learning Navigation for Recharging a Self-Sufficient Colony Robot

Gary Parker
Connecticut College, parker@conncoll.edu

Richard Zbeda

Recommended Citation
Parker, G.; Zbeda, R., "Learning navigation for recharging a self-sufficient colony robot," Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on , vol., no., pp.734,740, 7-10 Oct. 2007 doi: 10.1109/ICSMC.2007.4413695

Learning Navigation for Recharging a Self-Sufficient Colony Robot

Keywords
genetic algorithms, intelligent robots, learning systems, mobile robots, multi-robot systems, navigation path planning

DOI:10.1109/ICSMC.2007.4413695

Learning Area Coverage for a Self-Sufficient Colony Robot

Gary B. Parker, *Member, IEEE,* and Richard Zbeda

Abstract— **It is advantageous for colony robots to be autonomous and self-sufficient. This requires them to perform their duties while maintaining enough energy to operate. Previously, we reported the equipping of power storage for legged robots with high capacitance capacitors, the configuration of one of these robots to effectively use its power storage in a colony recharging system, and the learning of a control program that enabled the robot to navigate to a charging station in simulation. In this work, we report the learning of a control program that allowed the simulated robot to perform area coverage in a self-sufficient framework that made available the best pre-learned navigation behavior module.**

I. Introduction

For robots and their respective systems to perform long-term and independent colony tasks, they must have two properties: autonomy and self-sufficiency [1]. Autonomy means that the robots independently govern their own behavior and make their own decisions. Self-sufficiency implies a system's ability to maintain such robots in an operational state for long periods of time by directing them to maintain their own power supply. In particular, the system must include rechargeable batteries and a self-recharge mechanism; additionally, it must also rely on procedures that enable the robots to constantly examine their power supply and to travel to and use a charging station. To be effective, autonomous and self-sufficient robots must balance these two competing requirements through the basic work - find fuel - refuel cycle [2].

The increasingly recognized field of autonomous and self-sufficient robotics has been an area of interesting research. Yuta and Hada [3] achieved a "sport" record by creating a robot that recharged its battery every ten minutes and operated continuously for a week. Sempé, Muñoz, and Drogoul [4] devised and contrasted various robot group strategies for sharing a charging station where only one robot could recharge its batteries at it at a time. Individual robots would begin wandering and would then navigate towards a power station when their go-and-recharge power threshold was reached. Birk [5] reports a problem of using batteries for a robot by showing that cell chemistry may impede consistent behavior. It was proposed to alternate the use of multiple rechargeable battery packs. Kouzoubov and Austin [6] constructed an inexpensive and effective recharging station and docking algorithm for recharging mobile robots. The system utilized a particle filter to extract the recharging station location from laser range sensor data. Silverman, Jung, Nies and Sukhatme [7] effectively developed a recharging control architecture for a mobile robot to perform door monitoring and obstacle avoidance tasks for long periods of time.

Common to the majority of these previous works is that the robots' on-board power supply consisted of batteries and that robot behavior was preprogrammed. The work reported in this paper is distinctive, firstly, because it builds on a previous work in which we proposed the usage of capacitors in place of batteries as an onboard power supply for a legged colony robot [8]. Capacitors expend their charge and recharge much faster than batteries do. This property helps to obviate the requirement for long work and recharge cycles and allows a more continuous pattern of behavior in self-sufficient robots. In addition, capacitors are effective due to resistance to old age [5], and size/power. Secondly, our work is unique because robot behavior is not preprogrammed, but is learned through a means of evolutionary computation. Although one robot is used in this work, a colony area is employed with the goal of learning self-sufficient behavior for a colony of robots.

In past research, evolutionary computation used to learn the weights of an artificial neural network, genetic programming, and cyclic genetic algorithms have been used to learn control programs for robots. The use of a genetic algorithm to learn the connection weights and/or architectures for artificial neural networks has been a common method of learning robot control [9]. Floreanno and Mondada [10] evolved neural networks to control self-sufficient behavior on the Khepera robot. The robot's task was to perform navigation, obstacle avoidance and to locate a charging station before the robot's simulated batteries lost power. Beer and Gallagher [11] showed that genetic algorithms can be implemented to evolve effective neural networks for chemo-taxis and legged locomotion controllers. Lund and Orazio [12] evolved a neural network control program for a Khepera robot to avoid walls and obstacles in an enclosed space.

Manuscript received October 27, 2008.

G. B. Parker is with Connecticut College, New London, CT, 06320 USA (phone: 860-439-5208; e-mail: parker@conncoll.edu).

R. Zbeda is a graduate student at University of Pennsylvania, Philadelphia, PA, 19104 USA (e-mail: rzbeda@seas.upenn.edu).

Genetic programming (GP) is another common method for learning robot control. Busch et al. [13] used GP to evolve robot control programs to produce gaits for simulated robots that were independently appropriate for each specific robot's morphology. Nordin et al [14] evolved wall-following behavior for a Khepera robot that was successful both in simulation and on the actual robot. Lazarus and Hu [15] used GP to involve sensor input in the development of controllers for simulated robots that performed obstacle avoidance and wall-following tasks.

This paper reports the second of two segments of research in learning autonomous and self-sufficient robot behavior. The research involves the learning of the distinct but related tasks of area coverage when the robot is working, and navigation, for when the robot is finding fuel. Incremental learning of the two separate behaviors is used as method to learning the overall autonomous and self-sufficient behavior. de Garis [16] proposed learning behavior in incremental steps and the method is now a well-established approach in evolutionary robotics [17]. In a previous work, the navigation behavior was learned [18]; the best solution was then made into a module available to the GA learning the area coverage task in this work. While both tasks could have been learned at once, this incremental approach was chosen because the pattern of area coverage heavily relies on the robot's effectiveness in navigating and finding fuel.

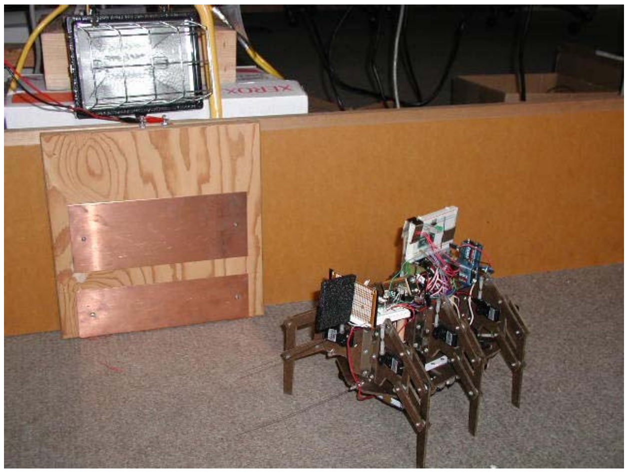


Fig. 1. The robot, its metallic probes, the power station, and the light source. Six pairs of capacitors are attached underneath to power the ServoBot.

Previous research in the area of coverage path planning has involved preprogramming robot behavior to cover an area and avoid obstacles. Several such works have been documented in [19, 20]. In recent work, Parker created learning methods for adaptation to specific capabilities in inexpensive legged robots without precision of movement performing area coverage. Parker [19] used a cyclic genetic algorithm to learn turn cycles for a hexapod robot that produced area coverage patterns. In addition, Parker [20] tested the simulated behavior on a physical hexapod robot. The learning method effectively produced patterns of motion for both the simulated and actual robot.

In this work, we report the use of a cyclic genetic algorithm to learn the area coverage task for the hexapod robot equipped with capacitors, such that the area coverage task is a part of the self-sufficient framework. Specifically, the robot learns the best area coverage pattern subject to a fuel/power constraint while our pre-learned navigation behavior module is made available to the robot. The learning of such self-sufficient area coverage takes place in simulation and the actual robot tests confirmed the viability of the resulting solution.

II. The Robot

The ServoBot is a small, inexpensive hexapod robot developed by David Braun at Indiana University for legged robot and colony robotics experimentation. It is constructed out of Masonite wood and each of its six legs has two degrees of freedom, vertical and horizontal. It has two servos for each leg to provide the forward thrust and vertical movement. Originally, the ServoBot was constructed to carry one 9V battery and 4 AA batteries as a power supply. The 9V battery powers the onboard BASIC Stamp II microcontroller (which controls the robot) and the 4 AA batteries power the 12 servos.

It was determined in previous research [8] that non-rechargeable batteries should not be used to power the colony since they prohibit self-sufficiency. Our solution was to use 6 pairs of capacitors at a total capacitance of 6 * (50F * 50F)/(50F + 50F) = 150F (F stands for farads) and a 4.6V (limitation due to the high capacitance capacitors used) level of charge. The capacitors expend their charge by powering the servomotors and would recharge by connecting to a power station (see Figure 1) via the robot's metallic probes. Tests were done to determine the run time and charge time when the capacitors were charged at 4.6V. It was determined that for the task of walking for approximately 3 minutes the ServoBot had a charge time of 2min 20sec and had a run time of 2min 50sec [8] .

In previous work, we devised a control system in which sensor information was made available to onboard BASIC Stamp II controllers to guide robot behavior towards self-sufficiency [22]. A voltage sensor (a microcontroller [PIC 12F675] functioning as an Analog-to-Digital voltage converter [ADC]) was used to monitor power status. Additionally, light sensors (two CdS [cadmium sulfide] photocells) and a source were installed for the robot to autonomously find and travel to the power station.

III. Learning Area Coverage

This research involved the learning of a control program to direct a simulated hexapod colony robot towards completing the dual self-sufficient tasks of area coverage (i.e. work) and navigation (i.e. finding fuel). The learning method used was a multi-loop cyclic genetic algorithm with conditional branching with the training done on a simulation of the actual robot. In this work, the successful completion of a control program for area coverage is reported. Because the simulation realistically and accurately represented the physical colony robot and power supply system, the evolved behavior could be effectively transferred to the physical system.

A. Cyclic Genetic Algorithm (CGA)

The *cyclic genetic algorithm* (CGA) [23], a variant of Holland's genetic algorithm (GA) [24], was developed to automatically generate code for cyclic control problems. Instead of representing traits of a solution (as in a GA), the genes in a CGA chromosome represent tasks to be completed. The CGA chromosome contains a list of instructions that may include loops to facilitate the execution of a sequence of tasks (see Figure 2). Multiple levels of looping such as loops of single actions, loops covering loops of single actions, and loops of the entire chromosome can be achieved.

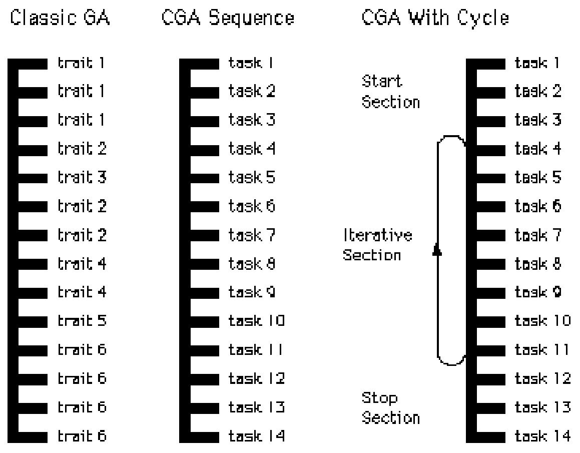


Fig. 2. GA and CGA chromosomes.

CGAs are useful for learning actual and/or simulated self-sufficient behavior since the learning of cyclic and repetitive behavior is involved. Initially, the CGA was used to evolve single-loop programs that directed hexapod robots to execute gait cycles [23,25]. A large portion of the chromosome was composed of a single loop to address one repetitive overall task. With the introduction of dynamic sensor input and subsequent need for different sub-tasks, however, a single loop was inadequate. To address this issue, a CGA with conditional branching [26] was developed to allow branching to specified loops in response to sensor inputs.

To accommodate the use of a particularly large number of sensors, the CGA with conditional branching method was further modified [27]. The key distinction was that the CGA also had to learn what sensors to test, when to execute sensor branching tests, and what chromosome segment to jump to. In this way, the more significant branch conditionals could be executed, thus reducing the total number of loops/segments needed in the chromosome. Such a multi-loop CGA with conditional branching is used in this work to learn the robot tasks of area coverage and navigation.

B. The Simulation

The simulation was modeled after the ServoBot's gait cycles, or, walking styles. A gait cycle is defined as a single step where the robot's legs begin moving and eventually return to their original position after following a full step cycle of positions. On the physical ServoBot, the standard gait consists of a list of activations that the on-board controller uses to continuously direct the instantaneous movement of the 12 servo actuators.

In previous work, a tripod gait was evolved with a cyclic genetic algorithm to provide optimal speed for a specific ServoBot [20]. 32 degrees of turn were provided in the gait cycle through the use of affecters which could interrupt activations to the thrust actuators for either the left or right side of the robot. In this work we used 12 of the left or right turns (gait cycles) developed in the previous work. In addition, we created four preprogrammed gaits (no movement, straight backwards, left rotate, and right rotate) for the ServoBot for a total of 16 different gait cycles.

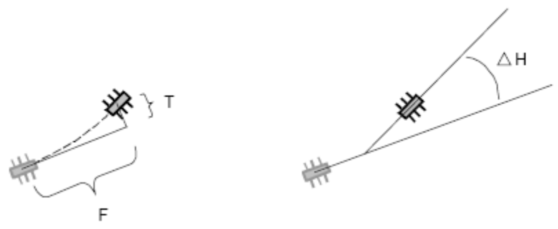


Fig. 3. Gait Cycle Turn Measurements. The left diagram shows F and T. F is the distance moved forward. T is the distance moved in the turn direction. The right diagram shows ΔH, the change in heading [20].

In both the previous work [20] and for the ServoBot used in this research, each turn gait cycle was measured for rate of turn and displacement resulting in a list of three numbers: F, T, and ΔH. F was the distance in centimeters that the robot moved forward. T was the distance traveled perpendicular to the F axis, where left was negative and right was positive. ΔH was a measurement (in degrees) of the change in heading where left was negative and right was positive. A diagram of F, T, and ΔH measurement is shown in Figure 3. Turn rates, defined using F, T, and ΔH; were stored for each gait cycle. Figure 4 shows the 16 gait cycle measurements.

```
(0 (11.6 -0.8 -1.3))               ( 8  (11.4 0.8 2.7))
(1 (11.9 -1.5 -5.0))               ( 9  (10.1 2.3 6.2))
(2 (10.4 -2.8 -10.3))              (10  (8.4 3.5 11.0))
(3 (8.4 -4.3 -15.8))               (11  (5.3 4.0 16.7))
(4  (6.3 -4.8 -20.3))              (12  (3.8 4.3 20.2))
(5  (5.7 -3.7 -24.7))              (13  (3.7 4.0 24.3))
(6  (0.0001 -0.0001 -27.0))  (14  (0.0001 0.0001 27.0))
(7  (0 0 0))                       (15  (-12.4 0.5 -1.8))
```

Fig. 4. The robot capabilities for each of the 16 gait cycles. The first number indicates the gait cycle type, and the remaining list of three numbers represents F, T, and ΔH respectively. Gaits 0-5 are right turn gait cycles, gaits 8-13 are left turn gait cycles, gait 6 is the right rotate gait cycle, gait 14 is the left rotate cycle, gait 7 is the no movement gait cycle and gait 15 is the straight backwards gait cycle.

The robot's gait cycle measurements were used to calculate moves by the simulated robot in the simulation. The robot's position in the simulation area consisted of its xy coordinates and a number between 0 and 359 representing its heading. Motion was determined by applying a gait cycle from the chromosome. The new xy position and heading of the robot were calculated by applying the forward (F), left/right (T), and gait cycle heading change (ΔH) gait cycle values to the current state.

The simulation area (500x500 units) defines the space in which the simulated robot moves and models an 8x8 foot colony area in the lab. The area includes a simulated power station that models the actual power station. The simulated robot's charging mechanisms are also modeled after the actual robot. The robot must direct its metal probes to make contact with the power station at a feasible angle to and distance from the wall. The physical light source/sensor system is also modeled in the simulation. The light emanates from two point coordinates and illuminates the simulated colony space (see Figure 8). The simulated robot can only see the light when its heading and placement allow it to see the light source.

The actual robot sensors were closely modeled in simulation. The robot is equipped with left and right object detection sensors, left and right light sensors and, and a power sensor. Each sensor has two possible states: 0 (inactive) and 1 (active). The object detection sensors each have an activation distance and a 45 degree sensor span while the left and right sensors spans overlap by 10 degrees. The simulated light sensors each have an infinite activation distance and a span of 80 degrees while left and right sensors overlap by 40 degrees. The robot's power sensor senses when the robot's capacitor power level is below the lower power threshold.

Physical power usage quantities and thresholds were modeled in the simulation, specifically, the power usage per gait cycle, the empty power threshold, the low power threshold, and the high power threshold. The power usage per cycle is a constant 0.0209V of charge unless the robot remains motionless. The low power threshold was set at 4V. The empty power threshold marks the power level below which the robot will not have enough power to execute another gait cycle and was set at 2.7V. The high power threshold marks the amount of energy held after completing a charging routine at the charging station and was set at 5V. A charging routine is a preprogrammed routine in which the robot connects with the power station (while in navigation mode), recharges up to the high power threshold, executes 4 straight backwards gait cycles, and then starts/continues its area coverage task.

C. *Navigation Task*

We used a multi-loop CGA with conditional branching to learn the navigation task [18]. The chromosome structure, fitness evaluation and training procedure were highly analogous to the methods used in this work. The resulting behavior of effective navigation to the power station to start a recharging routine was made a behavior module for this work. The simulated robot used it as directed by the learned control program to produce effective area coverage patterns.

D. *Area Coverage using the Navigation Module*

We incrementally built on the previously learned navigation behavior in developing the area coverage behavior. The robot must either be in area coverage (work) mode, or navigation (refueling) mode. The simulated robot needed to learn behavior that would enable it to do area coverage starting from the recharging area, and a conditional branching test to tell it to switch into navigational mode when low on energy.

A multi-loop CGA with conditional branching was used to learn the complete self-sufficient task. It was one that is designed to handle many sensor inputs by learning to jump from one loop to another [27]. Multiple loops were needed for this CGA because different overall tasks such as avoiding walls, backing and turning away from walls after contact, making wide sweeps to across new colony area, and turning to position for additional wide sweeps across new area were required. The correct process control jumping between such types of sub-tasks needed to be learned. In addition, learning jumping was needed to direct the individual to switch into navigation mode.

The structure of the area coverage chromosome part was identical to the structure of the navigation chromosome part. Each chromosome consisted of 8 genes where each gene (see Figure 5) consisted of a 2 bit number followed by four 7 bit numbers. Each gene represented a "for" loop with the 2 bit number specifying how many times the loop should be executed such as 01 (once), 10 (twice), 11 (three times) and 00 (infinite). The four 7 bit numbers represented the instructions in the loop.

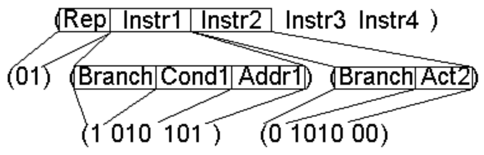
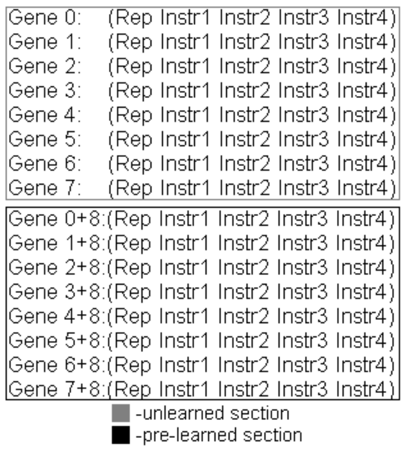


Fig. 5. The structure of a gene. Two different instruction types are shown.

This chromosome structure was combined with the pre-learned navigation chromosome to form a 16 gene chromosome. Specifically, the 16 gene chromosome consisted of an 8 gene pre-learned section and an 8 gene unlearned section (see Figure 6). The same pre-learned chromosome was used as part of each 16 gene chromosome of each population created and was fixed (i.e. it did not change/evolve). The 8 gene pre-learned chromosome was the chromosome with the best fitness in the 1024th generation out of five populations in the navigation simulation.



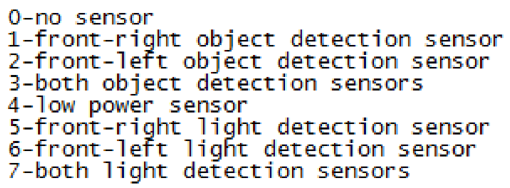
Gene 0:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 1:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 2:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 3:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 4:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 5:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 6:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 7:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 0+8:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 1+8:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 2+8:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 3+8:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 4+8:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 5+8:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 6+8:	(Rep Instr1 Instr2 Instr3 Instr4)
Gene 7+8:	(Rep Instr1 Instr2 Instr3 Instr4)

▨ -unlearned section
■ -pre-learned section

Fig. 6. The initial complete chromosome structure with an unlearned (Genes 0 through 7) and a pre-learned (Genes 8 through 15) section. Control would alternate between the two and was maintained in one section at a time.

Two types of instructions could be executed: a specific gait cycle instruction or a conditional branching instruction. A specific gait cycle instruction directs the robot to move according to the displacement and rotation of the gait. A conditional branching instruction tests specific sensors for their input states (see Figure 7). If tests are positive, control switches to the beginning of the gene specified in the instruction. Otherwise, the next instruction in the gene is executed.

For any 7-bit instruction, the first bit represented the type of instruction. If it was a 0, then the next 4 bits would signify the one of 16 gait cycles to be executed. If it was a 1, then the next 3 bits would signify one of 8 combined sensor input combinations/states that were considered. The 8 sensor combinations are listed in Figure 7. The 3 bits after the conditional signified the address of one of the 8 segments to which a jump would be executed if the test was positive. When executing the pre-learned section, an extra most-significant bit was added to the branching addresses so that control would always reside in genes 8 through 15.



```
0-no sensor
1-front-right object detection sensor
2-front-left object detection sensor
3-both object detection sensors
4-low power sensor
5-front-right light detection sensor
6-front-left light detection sensor
7-both light detection sensors
```

Figure 7. The eight sensor state tests. A conditional branching test was positive if the sensor/s considered formed a subset of all activated sensors at the current step. The low power sensor is activated when the robot power level falls below the low power threshold of 4.0V.

The flow of the fitness evaluation of a chromosome was nearly identical to that used in learning the navigation task. It would begin by completing the list of instructions in the first gene of the section (i.e. unlearned or pre-learned) it is in. Instructions would be executed in the “for” loop until it finished, in which case the next gene started, or until a branch condition instruction sent the point of execution to another gene. If the algorithm finished executing the last gene then control would return to the beginning of the first gene. To avoid infinite branching the run would halt after 32 consecutive branches without a gait cycle instruction.

A fitness value assigned to an individual for a single run was based on the number of squares it reached. The robot is given a total of 600 steps to cover the colony area. The colony area was sectioned off into 100 squares that were each 50x50 units, and the contact width on each square was the center 38x38 units of the square. The robot’s x and y coordinates (at the robot’s center) had to be within the contact width of a square for the square to have been reached. Each new square reached was added to a list of squares, and after the run completed the sum total of the squares represented the robot’s fitness value. Additionally, there was a zone of 25 units off each wall that if reached would terminate the run. This was meant to discourage the robot from getting too close to the walls, but also made it difficult for the robot to reach the outer layer of squares. In training, the outer layer of squares was accessible but for the final fitness evaluation tests this outer layer was not counted. Thus robot could get a maximum fitness of 100 squares in training and 64 squared in post-evolution tests. The 64 colony area squares are shown in Figure 8.

The fitness assignment procedure was as follows. The robot would start with a power level set at the low power threshold of 4V in order to induce it to learn to conduct a conditional branching test testing its low power sensors when low on power in a timely manner. When such a test was positive, the robot would switch into the first gene of the navigation chromosome section and travel straight towards the power station. If and after undergoing a charging routine, the robot would automatically switch back to the first gene of the of the area coverage chromosome section. When in area coverage mode, the robot had the incentive to learn an effective means of covering the area through a series of straights and turns, such that new squares were consistently being reached. In addition, because the periodic conditional branching test of the low power sensor allowed the robot to effectively reach the power station to refuel when low on energy and do more area coverage, the robot would benefit from learning to periodically conduct such tests.

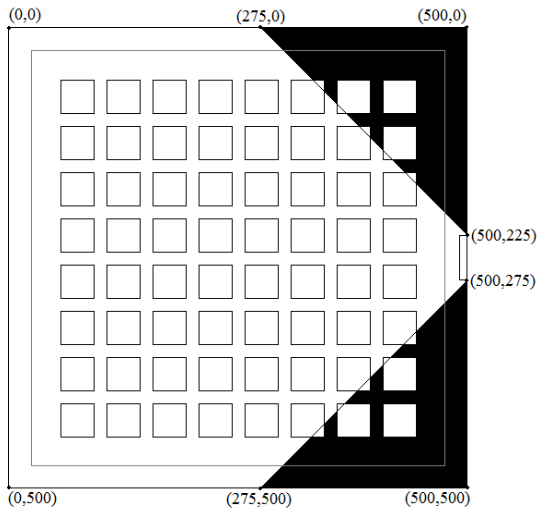


Fig. 8. The simulated colony area with the squares to be covered, the light sources and light distribution in the area, the kill zone, and the power station. The figure is drawn to scale.

Termination of a run and the assignment of a fitness value could occur for four reasons: the all the squares in the area were reached resulting in a maximal fitness, the maximum number of steps were taken, the robot got within 25 units of a wall, or the power level of the robot dropped below the empty power threshold of 2.7V resulting in consecutive no-movement gaits.

The actual training procedure was as follows. Five populations of 256 individuals were created where the first 8 genes were randomly generated and the second 8 genes were the unchanging pre-learned navigational chromosome. Evolution was carried out for 512 generations in which each individual was assigned the same 10 randomly generated starting positions for each generation. The fitness of each individual was based on its average fitness value in the 10 runs. This average was raised to the 1.5 power to magnify differences in fitness values between individuals in a population. The individual with the best fitness was automatically included in the next generation. The area coverage section of the remaining individuals was produced through the application of the three standard genetic operators, namely, selection, crossover and mutation. The populations at several generations from 0 up to 512 were saved during the evolution.

In evolving the area coverage portion of the chromosome two chromosomes for crossover and mutation were chosen using roulette wheel selection. For each pair, one of two different types of crossovers occurred, each chosen with a 1-in-2 probability. For the first type of crossover, a random index between 8 and 15 was chosen. The resulting chromosome was made up of the [8, index] genes of the first chromosome and the (index, 15] genes of the second chromosome. The second type of crossover used gene-by-gene crossover, where each corresponding gene of the parent chromosomes was combined using one-point crossover.

After crossover, the resultant chromosome was subject to one of two types of mutation. The first type was used when the first type of crossover was used. This involved going through each gene in the chromosome, such that each instruction was randomly regenerated with a 1-in-600 chance. The 2 bit "for loop" number was left unchanged. The second type of mutation was bit-by-bit mutation, and was used when gene-by-gene crossover was used. In this type of mutation, each gene of the chromosome was subject to a bit-by-bit mutation, such that each bit in the gene was flipped with a 1-in-200 probability.

IV. RESULTS

There were 5 tests (each with a population where individuals' unlearned chromosome sections were initially randomly generated) to check the viability of the learning system. During the evolution of each of the populations, the entire population (along with the individual with the highest fitness in the population) was stored at the 0, 1, 32, 64, 128, 256, 384, and 512 generations. Tests on each population's best individual at each of these generations were evaluated at 100 randomly generated start positions near the power station. The best individual fitness (averaged over the 100 trials) from each population at each stored generation was recorded and is shown in Figure 9.

For each of the 5 tests (each with a population where individuals' unlearned chromosome sections were initially randomly generated), a population was stored at the 0, 1, 32, 64, 128, 256, 384, and 512 generations. Tests on each population at each of these generations were evaluated at 100 randomly generated start positions. The best individual fitness (averaged over the 100 trials) from each population at each stored generation was recorded. Figure 9 shows this fitness divided by 64 (max possible fitness) to determine a fitness percentage.

Observations of the robots in simulation and in the actual colony space revealed that the CGA had learned reasonable solutions after 512 generations. In each population, the individual with the best fitness would first back up from the charging station and then rotate in one direction to change its heading in preparation for forward movement away from the charging station. Afterwards, it would make minimal turns and sweep the area with a wide arc pattern. Occasionally, it would get close to a wall in which case the turns became more extreme in order to avoid getting too close, and after which, the robot resumed making minimal turns. After reaching a wall or point far from the charging station, the robot would switch back to navigation mode. Variation in each area coverage sweep was generated due to the fact that the robot nearly always starts in a different spot when doing area coverage as the charging routine was approached from a different angle each time. It is also notable that in each area coverage sweep, the robot would avoid being in the dark area when about to switch into navigation mode. This would be necessary to allow the robot to use its light sensors to successfully reach the charging station.

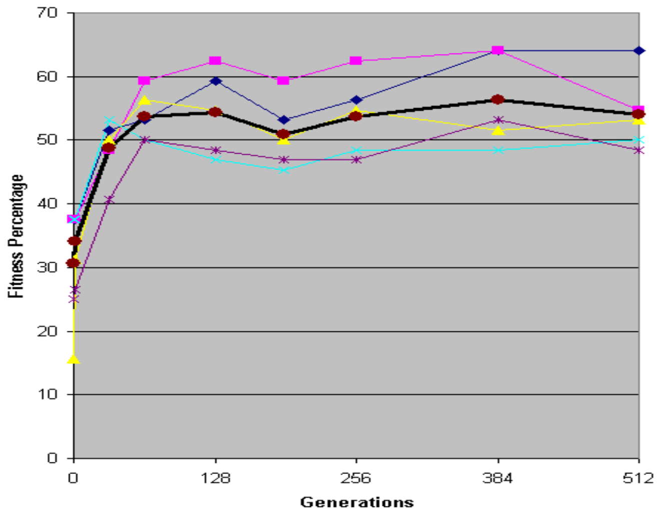


Fig. 9. Fitness of the area coverage behavior. Percentage fitness is the fitness divided by the max possible fitness. Each line represents the fitness of one of 5 populations; the bold line shows the average of the 5. The best individual for each population at each generation was saved and individuals from selected generations were tested from a series of common staring positions.

The described behavior is shown in Figure 10. The best individual from the 512 generation for each of the 5 populations was run from one specific set of xy-coordinates (x=415, y=252, h=98). These coordinates were chosen because they signify a plausible robot position after undergoing a charging routine. Each test shows how the robot would make wide overlapping sweeps of the colony area. It is notable that from Figure 10 that there is significant differences in the general strategy of making wide overlapping sweeps of the area. In tests 1 and 3, the robot makes minimal left turns and stays within the center of the area. In test 2, the robot also makes minimal left turns, but stays more in the upper half of the colony area. In test 4, the robot makes relatively sharp right turns. In test 5, the robot alternates between making minimal right turns in covering the upper half of the area, and making solely straight movements in covering the lower half of the area.

It is evident from the results of all five runs of the complete task performance that the population fitness average gained fitness quickly up through 64 generations and remained in a narrow range of change after that (see Figure 9). This is probably because when any relatively successful strategy of covering area in this manner is learned, it is very specialized and detailed with respect to the environmental settings, and any evolutionary attempts to significantly improve/change it may come at heavy costs in the form of lowered task performance and may not be worthwhile. The best solution does not achieve a standardized method of area coverage such as back-and-forth boustrophedic motions since this pattern would probably not be feasible with the power consumption constraints. The best solution produced in simulation was downloaded to the actual robot and initial observations of its behavior indicated that the CGA learned solution can be effectively used on the physical robot operating in the actual colony space.

V. CONCLUSIONS

We have shown that using a multi-loop CGA with conditional branching is an effective learning method for learning the self-sufficient area coverage task. Tests on the actual robot confirmed that the self-sufficient area coverage control program produced a reasonable area coverage and navigation track over the ground in the colony space.

We believe that our approach to a self-sufficient system has general significance even though this experiment was specific to our colony setup. Using capacitors presents a viable power supply alternative especially for small and light weight robots for which short run/charge times are desired.

Our self-sufficient approach requires much additional development to increase its effectiveness. It would greatly enhance the ability of our robots to hold concentrated charge at higher voltage levels by using improved capacitors rated at 2.7v and 100F that are now available. Such a change would increase the robot's work cycle time of a self-sufficient task, which would allow it to develop more elaborate search patterns. We could make the entire system self-sufficient by using solar or wind energy to power the charging station. Multiple robot interaction in a self-sufficient system should be addressed to exhibit behavioral relationships such as cooperation and competition.

REFERENCES

[1] McFarland D. (1995) Autonomy and Self-Sufficiency in Robots. The Artificial Life Route To Artificial Intelligence. Building Embodied, Situated Agents. Steels L.(ed). Lawrence Erlbaum Ass. Pub. USA, 187-213.

[2] McFarland D., E. Spier. (1997) Basic Cycles, Utility and Opportunism in Self-sufficient Robots. Robotics and Autonomous System (20), 179-190.

[3] Yuta S., Hada Y. (2000) First Stage Experiments of Long Term Activity of Autonomous Mobile Robot: Result of Repetitive Base Docking over a Week. In: Proceedings of ISER'00, 7th International Symposium on Experimental Robotics, 235-244.

[4] Sempé F., Muñoz A., Drogoul A. "Autonomous Robots Sharing a Charging Station with no Communication: a Case Study." Proceedings of the 6th International Symposium on Distributed Autonomous Robotic Systems (DARS'02). June 2002.

[5] Birk A. (1997) Autonomous Recharging of Mobile Robots. In: Proceedings of the 30th International Symposium on Automative Technology and Automation. Isata Press.

[6] K. Kouzoubov, D. Austin. "Autonomous Recharging for Mobile Robotics." Proceedings of the Australasian Conference on Robotics and Automation. Auckland, Australia. November 2002.

[7] M. Silverman, B. Jung, D. Nies, G. Sukhatme. "Staying Alive Longer: Autonomous Robot Recharging Put to the Test." Center for Robotics and Embedded Systems (CRES) Technical Report CRES-03-015. University of Southern California, 2003.

[8] G. Parker, R. Georgescu, and K. Northcutt, "Continuous Power Supply for a Robot Colony." Proceedings of the World Automation Congress (WAC 2004). June 2004.

[9] X. Yao, "Evolving artificial neural networks," Proc. IEEE, Vol. 87, No. 9, 1999, pp.1423-1447.

[10] D. Floreano and F. Mondada, "Evolution of Homing Navigation in a Real Mobile Robot," IEEE Transactions on Systems, Man and Cybernetics, Vol. 26, No. 3, 1996, pp 396-407.

[11] R. D. Beer and J. C. Gallagher, "Evolving Dynamical Neural Networks For Adaptive Behavior," *Adaptive Behavior,* Vol. 1, No. 1, 1992, pp. 91-122.

[12] H. H. Lund and O. Miglino, "From Simulated to Real Robots," *Proc. IEEE Third International Conference on Evolutionary Computation*, NJ, 1996.

[13] J. Busch, J. Ziegler, C. Aue, A. Ross, D. Sawitzki, and W. Banzhaf, "Automatic Generation of Control Programs for Walking Robots Using Genetic Programming," *EuroGP 2002*, LNCS 2278, 2002, pp. 258-267.

[14] P. Nordin, W. Banzhaf, and M. Brameier, "Evolution of a World Model for a Miniature Robot using Genetic Programming," *Robotics and Autonomous Systems*, Vol. 25, 1998, pp. 105-116.

[15] C. Lazarus and H. Hu, "Using Genetic Programming to Evolve Robot Behaviours," *Proc. Third British Conference on Autonomous Mobile Robotics & Autonomous Systems*, Manchester, UK 2001.

[16] H. de Garis, "Genetic Programming: GenNets, Artificial Nervous Systems, Artificial Embryos", Ph.D. thesis, Université libre de Bruxelles, Belgium, (1991).

[17] Petrovic, P. (1999) Overview of Incremental Evolution Approaches to Evolutionary Robotics, Proceedings to Norwegian Conference on Computer Science, p. 151-162.

[18] G. Parker and R. Zbeda, "Learning Navigation for Recharging a Self-Sufficient Colony Robot," Proceedings of the 2007 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2007). October 2007.

[19] G. B. Parker, "Learning Control Cycles for Area coverage with Cyclic Genetic Algorithms," Proc. Second WSES International Conference on Evolutionary Computation (EC '01). February 2001 (pp. 283-289).

[20] G. Parker, "Evolving Cyclic Control for a Hexapod Robot Performing Area Coverage." Proceedings of 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2001). August 2001 (pp. 561-566).

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

[22] G. Parker and R. Zbeda, "Controlled Use of a Robot Colony Power Supply." Proceedings of the 2005 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2005). October 2005.

[23] G. B. Parker and G. J. E. Rawlins "Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots," Proc. World Automation Congress, Vol. 3, Robotic and Manufacturing Systems, 1996, pp. 617-622.

[24] J. H. Holland, Adaptation in Natural and Artificial Systems, Ann Arbor, MI, The University of Michigan Press, 1975.

[25] 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). July 1997 (pp. 141-144).

[26] G. B. Parker, I. I. Parashkevov, H. J. Blumenthal, and T. W. Guildman, "Cyclic Genetic Algorithms for Evolving Multi-Loop Control Programs," Proceedings of the World Automation Congress (WAC 2004). June 2004.

[27] G. Parker and R. Georgescu, "Using Cyclic Genetic Algorithms to Evolve Multi-Loop Control Programs." Proceedings of the 2005 IEEE International Conference on Mechatronics and Automation (ICMA 2005). July 2005.

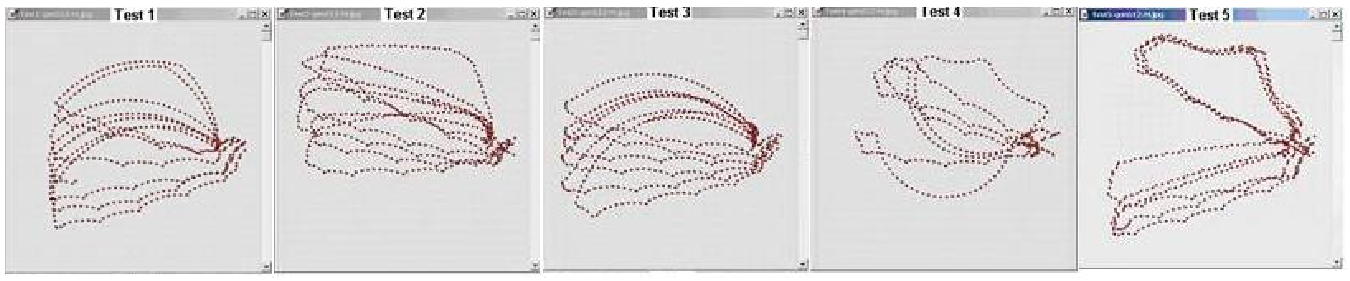


Fig. 10. The best individual from the 512 generation for each of the 5 populations was run from xy-coordinates (x=415, y=252, h=98). In each test the robot makes wide overlapping sweeps of the colony area, but in a different general strategy.