Learning Navigation for Recharging a Self-Sufficient Colony Robot

Gary Parker (*Member, IEEE*) and Richard Zbeda

Abstract—**It is desirable that colony robots be autonomous and self-sufficient, which requires that they can perform their duties while maintaining enough energy to operate. In previous work, we reported the equipping of legged robots with high capacitance capacitors for power storage and the configuration of one of these robots to make practical use of its power storage in a colony recharging system. Research reported in this paper involves the learning of a control program that allows this robot to navigate to a charging station. The viability of the configuration and the learned control program were verified by observing the actual robot as it operated using the best of the solutions produced in simulation.**

I. Introduction

In order to perform long-term and independent colony tasks, robots and their respective environmental systems must have two properties: autonomy and self-sufficiency (McFarland [1]). Autonomy means that the robots are able to make their own decisions and govern their own behavior. Self-sufficiency denotes the ability of a system to maintain these robots in a viable state for long periods of time, such that the robots can maintain their own power supply. Specifically, the system must include recharging facilities, i.e. rechargeable batteries and a self-recharge device; in addition it must also rely on mechanisms that enable the robots to examine their power supply constantly and to locate and use a charging station. Any effective autonomous and self-sufficient robot must balance these two competing characteristics through the basic cycle outlined by McFarland and Speir [2], namely, work - find fuel - refuel.

The framework of autonomous and self-sufficient robots has been an area of interesting research. Yuta and Hada [3] accomplished a "sport" record by making a robot that ran continuously for a week by recharging its battery every ten minutes. Birk [4] points out the problem of batteries and shows that cell chemistry may constrain consistent robot behavior. Alternating the use of multiple rechargeable battery packs was proposed. Floreanno and Mondada [5] implemented evolved navigation and obstacle avoidance behavior on a Khepera robot that regularly located a charging station before the robot's batteries lost power. Due to long charging times for any appropriate rechargeable batteries, however, no battery hardware was used and the batteries and their recharging were simulated. Sempé, Muñoz, and Drogoul [6] presented and compared various strategies to enable a group of robots to share a charging station, as only one could recharge its batteries at it at a time. Individual robots would wander until their go-and-recharge power threshold was reached, at which point they would navigate towards a power station. Steels [7] built an ecosystem where two rival mobile robots compete to use a shared power supply. No detailed results were presented. Michaud [8] proposed using artificial emotions to organize long term activity for a group of robots, although no experiments involving physical robots were presented.

Common to most of these previous works is that batteries were used as an on-board power supply and that robot behavior was preprogrammed. In our previous work, we presented capacitors as a replacement for batteries used as an onboard power supply [9]. Capacitors are like batteries, except that they expend their charge and recharge much faster. This property enables a more continuous pattern of behavior in robots as opposed to having them work and recharge for long periods of time. In addition to facilitating continuous behavior, capacitors are effective for two other reasons: resistance to old age [4], and size/power. In addition, our work is unique because robot behavior is not preprogrammed, but is learned through a means of computational intelligence.

The work presented in this paper is the first of two segments of research in learning autonomous and self-sufficient robot behavior. The research involves the learning of two separate but related tasks: area coverage as an assigned task while the robot is working, and navigation, for when the robot must find fuel. We employ incremental learning of the two separate behaviors/tasks to learn the overall autonomous and self-sufficient behavior. Learning behavior in incremental steps was proposed by de Garis [10] and is a widely used approach in evolutionary robotics [11]. The navigation task was learned first and made a module available to the GA learning the area coverage task. This incremental approach was necessary, as opposed to learning both tasks at once, because each behavior is complicated in of itself and the pattern of area coverage is highly dependant on the ability of the robot to effectively navigate and find fuel. The main focus of this paper is on learning the navigation behavior module.

The navigational task required that the robot travel directly and efficiently towards the charging station. Numerous studies exist addressing learning navigational behavior. Mondada and Floreano developed Khepera, a miniature mobile robot, to study the evolution of control structures and had the robot perform, among other tasks, navigation and

Manuscript received March 1, 2007.

G.B. Parker is the Director of Computer Science at Connecticut College, New London, CT, USA (e-mail: parker@conncoll.edu)

R.S. Zbeda is a graduate in Computer Science at University of Pennsylvania (e-mail: rzbeda@seas.upenn.edu).

obstacle avoidance [5]. The controller consisted of an artificial neural network. Its weights were evolved using a combination of neural networks and standard genetic algorithms with fitness scaling and "biased mutations" [12]. Tuci, Quinn, and Harvey used a Khepera robot that was placed in an arena with the task to navigate towards/search for a target placed at one end of the arena [13]. No obstacle avoidance was implemented to help during the navigation; if the robot crashed into a wall, the trial was terminated. They used a neural network controller with fixed-connection weights and "leaky integrator" neurons, and a simple genetic algorithm for learning. At the National University of Singapore, controller evolution was studied using an incremental approach on a Khepera robot doing navigation and obstacle avoidance [14]. The goal was to test this incremental approach by first creating a neural controller for the mobile robot to perform straight navigation while avoiding obstacles and then later extend it to a wall following behavior. Ram, Arkin, Boone, and Pearce applied genetic algorithms to the learning of robot navigation behaviors for reactive control systems [15]. The task to be performed was navigation of dynamic environments. Only simulation results were obtained. In our work, we report the use of a cyclic genetic algorithm to learn the navigation task for a hexapod robot. The learning takes place in simulation and the viability of the resulting solution is tested on the actual robot.

II. Cyclic Genetic Algorithm (CGA)

The *cyclic genetic algorithm* (CGA) [16], a variant of Holland's genetic algorithm (GA) [17], was developed for automatic code generation for cyclic control problems. The genes in a CGA chromosome are tasks to be completed by the agent, as opposed to being traits of a solution as in the traditional GA. While still employing the same evolutionary processes, a CGA is distinct in that it contains a list of instructions, which could include loops to facilitate the repetition of a sequence of tasks (as shown in Fig. 1) in its chromosome. The tasks can be single actions or sub-cycles of actions and the entire chromosome can be executed repeatedly for a specified number of times; thus there are multiple levels of looping involved.

CGAs are particularly useful for problems that require the learning of cyclic and repetitive behavior, such as actual or simulated robot movement. When considering the cyclic nature of autonomous and self-sufficient robots, the CGA is particularly ideal. The CGA was used to evolve single-loop programs for robotic control of gait cycles for hexapod robots [16,18]. In these problems, a large portion of the chromosome was composed of a single loop, as the chromosome was structured to complete one repetitive overall task. However, when introducing dynamic sensor input based on changes in the environment and the need for differing overall tasks, a single loop was inadequate. To address this issue, a CGA with conditional branching [19], where a multi-loop control program was evolved such that it would branch to specified loops in response to sensor inputs, was developed.

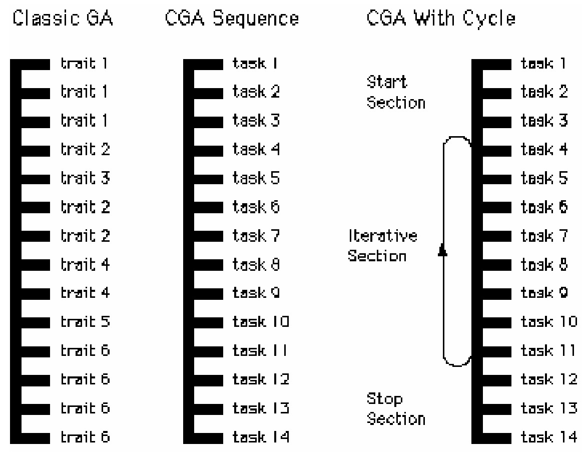

Fig 1. GA and CGA chromosomes.

In further work [20], the gene structure of the CGA chromosome was modified so that the implementation of a system of conditional branching was feasible for a problem with many sensors. The key difference was that the robot used did not just learn what actions had to be executed with branching tests occurring automatically between such actions. Instead, the CGA also had to learn when to execute sensor branching tests, what sensors to test, and what chromosome segment to jump to in the case of a change in detected sensor input. By learning to conduct its own branching in this way, the more significant branch conditionals could be executed, thus reducing the total number of loops/segments needed in the chromosome. As a result, the total number of loops/segments in the chromosome could be set independently from the total number of sensors used, rather than creating a loop/segment for every single sensor input combination. Such a multi-loop CGA with conditional branching is used in this work to learn the robot tasks of area coverage and navigation.

III. Experimental Framework of the Dual-Task Self-Sufficient Problem

The robot used in this research was the ServoBot, which was developed by David Braun at Indiana University for legged robot and colony robotics experimentation. It is a small, inexpensive hexapod robot constructed out of Masonite, which is a type of hard-pressed wood. Each of the six legs has two degrees of freedom as well as two servos which provide forward thrust and vertical movement. The ServoBot was originally constructed to be capable of carrying its own power supply in the form of one 9V battery and 4 AA batteries. The 4 AA batteries power the 12 servos, while the 9V battery powers the onboard BASIC Stamp II microcontroller (which controls the robot).

It was determined in previous research [9] that supplying the power to the colony should not involve non-rechargeable batteries since they only supply power for a limited period of time. A solution using capacitors was developed that included a total capacitance of 150F and a 4.6V (limitation due to the high capacitance capacitors used) level of charge. The higher the capacitance is, the higher the level of power the capacitors can hold at any voltage. When charged/used, the capacitors voltage level goes up/down. However, when being supplied/used at a level of charge beyond 4.6V, the capacitors voltage would increase/decrease very fast and was not as reliable a source of power for the servos.

Given this implementation, the capacitors would expend their charge by powering the servomotors and would recharge by connecting to a power station, which was a flat piece of wood (Fig. 2), with two metal plates connected to a power supply that was laid against one of the walls of the colony space. Future research will deal with supplying power to the power station through solar panels instead of a regulated power source. Recharging was facilitated by the robot's two metallic probes that extend outwards 7 inches in front of it and are connected to the capacitors. When these wires were placed in contact with the metal plates of the charging station, they would recharge the capacitors.

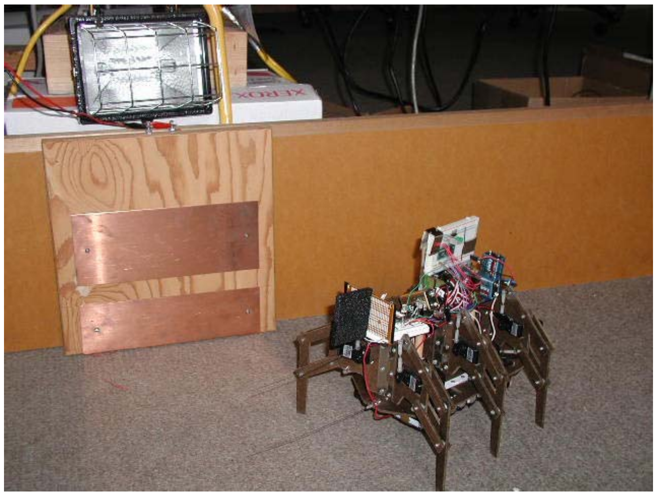
Fig 2. The robot and its metallic probes, power station with its metal plates, and light source. Six pairs of capacitors to power the ServoBot are mounted under the robot.

In order to decrease the charge time, the power supply was set at a voltage level higher that the capacitors maximum accepted voltage and the process was stopped when the maximum accepted voltage was reached. It was possible to significantly decrease the charge time in this way since the graph of Voltage vs. Time for charging a capacitor is an exponential graph. Tests were done to determine the run time and charge time for our specific capacitor configuration 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 [9].

In order for this self-sufficient system to be effective, we devised a control scheme that allowed the ServoBot to effectively make use of it [21]. A voltage sensor (a microcontroller [PIC 12F675] functioning as an Analog-to-Digital voltage converter [ADC]) was placed on the robot so that it could tell when it was time to charge, when it was charging, and when charging was complete. Additionally, the robot needed sensors to be able to autonomously find and travel to the power station to recharge on its own. Light sensors (two CdS [cadmium sulfide] photocells) and a source were installed for this purpose. The information from these two sensors was made available to onboard BASIC Stamp II controllers so that they could guide robot behavior towards self-sufficiency.

IV. Learning the Navigation Task as Part of a Self-Sufficient Framework

This research involved evolving a multi-loop control program using a CGA with conditional branching in simulation. This program was to direct a simulated hexapod colony robot towards effectively completing the dual self-sufficient tasks of area coverage (i.e. work) and navigation (i.e. finding fuel). In this paper, we report the successful completion of a control program for navigation. The simulation was defined in such a way that it realistically and accurately represented the physical colony robot, colony space, and self-sufficient power supply system. In this way, any evolved behavior in simulation could be transferred to the physical system with minimal discrepancies.

A. The Simulation Details

The simulation was modeled after the ServoBot -- the nature of its gait cycles. In this paper, we consider a gait cycle as a full step cycle where the legs have gone through all the positions in a step and have returned to their original position. On the physical robot, the standard gait, or walking style, 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.

In previous work, a repeated sequence of these activations was evolved with a cyclic genetic algorithm to produce an optimal gait (in terms of speed) for a specific ServoBot [18]; the gait generated for our test was a tripod gait. The tripod gait is where legs 0, 3, & 4 (right-front, left-middle, and right-back) 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.

Differing degrees of turn were provided in the gait cycle through the use of affecters. These affecters could interrupt (in differing degrees) activations to the thrust actuators for either the left or right side of the robot. In addition to left and right turn gaits, there was also a four extra preprogrammed gaits: no movement, straight backwards, left rotate, and right rotate. In all, a total of 16 different gait cycles were used.

Each turn gait cycle was measured for rate of turn and displacement. The form on measurement consisted of a list of three numbers: F, T, and ΔH. F was the distance in centimeters that the 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. A diagram of F, T, and ΔH measurement is shown in Fig. 3. Turn rates, defined using F, T, and ΔH; were stored for each gait cycle. Figure 4 shows the 16 gait cycle measurements.

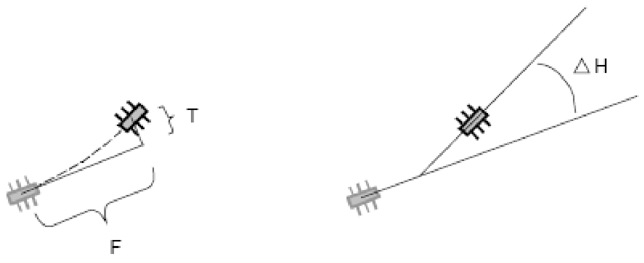

Fig 3. Gait Cycle Turn Measurements. The left diagram shows F and T. F is the distance moved forward (relative to the start position heading). T is the distance moved in the turn direction (perpendicular to the start position heading). The right diagram shows ΔH, which is the change in heading from before to after turn execution.

The robot's gait cycle measurements were used in the simulation to calculate moves by the simulated robot. The robot's position in the simulation area was fully described by its xy coordinates, as well as a number between 0 and 359 showing the direction of its heading. 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. The new heading was an addition of the current heading and the gait cycle heading change (ΔH).

```
(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. For each gait cycle, 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 the right turn gait cycles and gaits 8-13 are the 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 simulated robot would move and operate within the simulation area (500x500 units). A simulated power station was positioned at the coordinate (500,250) and covering the y range of 225-275. This simulated power station represents the size and placement of the actual power station. In order for the simulated robot to use the power station, its simulated wires, which extend out 45 units in front of it, must make contact with the power station while the robot is angled between 45 and 135 degrees (the robot center cannot get within 25 units of a wall, in which case the run would halt). The required angle is a realistic property of charging because the physical robot's probes cannot touch the power stations plated at an extreme angle. Further, the power station is marked by a powerful light source that only illuminates a certain area of the simulated colony space, as specified in Fig. 5. The light emanates from two point coordinates ((500,240) (500,260)) which are near the power station. They can only be seen/sensed by the robot from a point in the illuminated area, and only when either of the two light sources fall under the robot's sensor's vision. The actual distribution of light was measured in the physical colony space and the simulated light distribution is a relatively accurate representation of it.

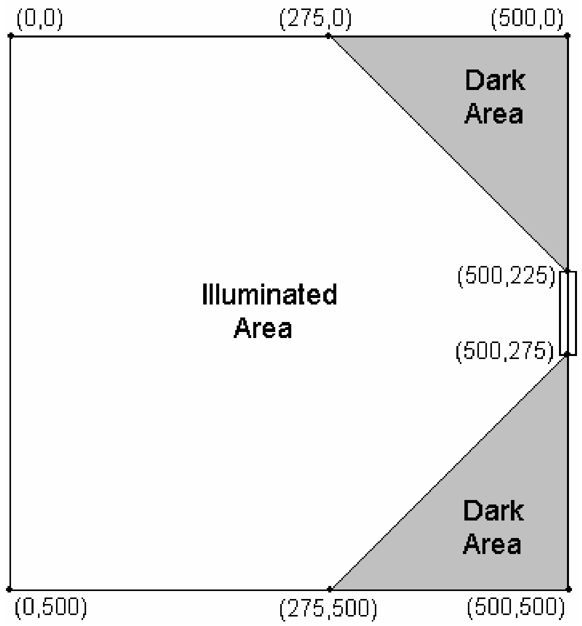

Fig 5. The simulated colony space.

The robot is equipped with three types of sensors – one that would enable it to detect the presence of a wall, one a direct beam of light from a light source, and one that would enable it to detect whether or not it had power below a certain threshold. The simulated robot has both left and right light and obstacle avoidance sensors, as well as one power sensor for a total of five sensors. Each sensor has two possible states: 0 (inactive) and 1 (active). The simulated thresholds used in delineating membership in such states have been specifically set to correspond with values determined through experimentation on the physical robot. In terms of the object avoidance sensors, the activation distance is 95.5 units, meaning that these sensors can sense a wall 95.5 units away. Also, the span of each of the object avoidance sensor's vision is 45 degrees. Since the left and right sensors are situated at the front of the robot, with their spans overlapping by 10 degrees, a total span of 80 degrees is provided. These sensors are important because avoiding walls is necessary to survival - contact with a wall will automatically end the life of the robot. In terms of light sensors, the activation distance is infinite, meaning that once the front of the robot is in line with the light source, a light signal is detected. The span of the light sensor's vision is 80 degrees. Since the two sensors are situated at the front of the robot, with their spans overlapping by 40 degrees, a total span of 120 degree is provided. Lastly, the robot's power sensor senses when the robot's capacitor power level is below the lower power threshold. The lower power threshold signifies the power-level below which the robot is in danger of not having enough power to move.

The quantities of power usage and thresholds were all approximated from observations of the actual robot and its capacitors. Among the quantities measured were the power usage per gait cycle, the empty power threshold, the low power threshold, and the high power threshold. In terms of the power usage per cycle, unless the robot remains motionless, it will expend a constant 0.0209V of charge. The empty power threshold was set at 2.7V and marks the power level below which the robot will not have enough power to execute another gait cycle. The low power threshold was set at 4V. The high power threshold is 5V and marks the amount of energy attained just after completing a charging routine at the charging station. A charging routine is a preprogrammed routine in which the robot connects with the power station (while its power level is below the lower threshold level), recharges up to the high threshold level, executes 4 straight backwards gait cycles (which can be executed only at this time) and then resumes its area coverage task.

V. Learning the Navigation Module

In order to learn the navigation task, a multi-loop CGA with conditional branching was used; a type that learns jumping from one loop to another, which was designed to handle many sensor input combinations [20]. Multiple loops were needed because different overall tasks of repetitive behavior were required for different sensors inputs as there were such behaviors as turning in a circle until light is sensed, and moving towards the light. The correct jumping between such types of sub-tasks needed to be learned. As a result, a chromosome had to be developed that would have a sufficient number of loops to cover each sub-task; in addition, there had to be a sufficient number of instructions in each loop to create enough gait cycle movement and/or conditional branch tests to make the loop function effective. However, while the number of loops and instructions in the loops had to be determined to be large enough for the problem, they could not be so large that the CGA could not converge on a good solution.

Each chromosome consisted of 8 genes, and each gene (Fig. 6) consisted of a 2 bit number followed by four 7 bit numbers. The gene represented a "for" loop with the two bit number specifying how many times the loop should be executed; the possible values being 01 (once), 10 (twice), 11 (three times) and 00 (infinite). The four 7 bit numbers represented the instructions in the loop.

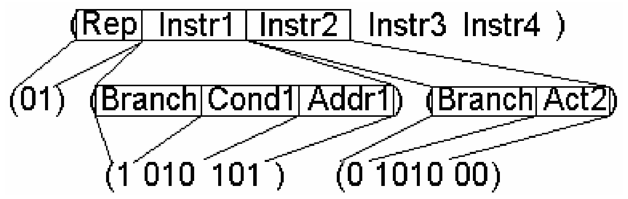

Fig 6. The structure of a gene. This example shows the two different instruction types.

The flow of the fitness evaluation of a chromosome would begin, in general, by completing the list of instructions in the first gene, or "for" loop. As the algorithm finished executing each instruction, it went on to the next instruction in the "for" loop, unless a branch condition instruction in the gene sent the point of execution to another gene. Additionally, as the algorithm finished executing a gene, it went on to the beginning of the next gene. In the case that the algorithm finished executing the last gene, control returned to the beginning of the first gene. The algorithm would continue until the run terminated. In order to avoid infinite branching, after 32 consecutive branches without a gait cycle instruction, the simulation run was set to halt.

Within this flow of execution of instructions, two types of instructions could be executed. The first is an instruction which directs the robot to move a specific gait cycle. In this case, the robot moves according to the displacement and rotation of the gait. The second is a conditional branching instruction, which tests specific sensors (Fig. 7) for their input states. If the sensor tests are positive, the chromosome section that controls the robot changes to a beginning of the gene specified in the instruction. Otherwise, the conditional branch instruction is passed, and the next instruction in the gene is executed.

```
0-no sensing
1-OBJ R
2-OBJ L
3-OBJ R & OBJ L
4-PS
5-LIGHT R
6-LIGHT L
7-LIGHT R & LIGHT L
```

Fig 7. The eight sensor state tests.

For each 7-bit instruction, the first bit represented whether or not the instruction was a conditional branching test or if it represented a gait cycle to be executed. If it was a 0, then the next 4 bits would signify the one of 16 gait cycles to be executed (although the straight backwards gait cycle could only be executed after a charging routine). Otherwise, if it was a 1, then the next 3 bits would signify one of 8 combined sensor input combinations/states that were designed with respect to their relevance towards useful comparisons between sensor types. 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 actual sensor states matched the conditional sensor state combination.

The actual training procedure was as follows. Five populations on 256 randomly generated individuals were created. Evolution was carried out for 1024 generations. For each generation each individual was assigned the same 10 randomly generated starting positions. Then each individual was evaluated starting from the 10 randomly generated starting positions. The fitness of each individual was based on its average performance in the 10 runs. The final average was raised to the 1.5 power to amplify fitness differences between individuals in a population. After each individual had been evaluated and assigned a fitness value, the individual with the best fitness was automatically included in the next generation. The remaining individuals were produced through the application of the three standard genetic operators, namely, selection, crossover and mutation. The populations at periodic generations from 0 up to 1024 were saved during the evolution.

To assign a fitness value to a single individual for a single run, the following procedure was used. The robot would start with a power level set at the low power threshold of 4V and a relatively low maximum of 200 steps in order to induce it to waste little time and travel straight towards the power station. Its goal to would be to connect with the charging station or to get as close to it as possible (without getting within 25 units of the wall) before the run was terminated. The robot got the maximum fitness for reaching the charging station. If the robot did not reach the power station it received a fitness value related to its proximity to the power station so that the closer the robot was, the higher its fitness rating. In this latter case, the proximity distance was square-rooted in order to amplify the reward received at closer distances.

Termination of a run and the assignment of a fitness value could occur for any of four reasons: the power-station was reached, the maximum number of steps (e.g. gait cycles) were taken, the robot got within 25 units of a wall (collision with the wall), and the power level of the robot dropped below the empty power threshold of 2.7V.

VI. Results

For each of the 5 tests (each with a population that was initially randomly generated), population details were stored at the 0, 1, 32, 64, 128, 256, 384, 512, 768, and 1024 generations. Tests were conducted, such that the populations 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 (Fig. 8).

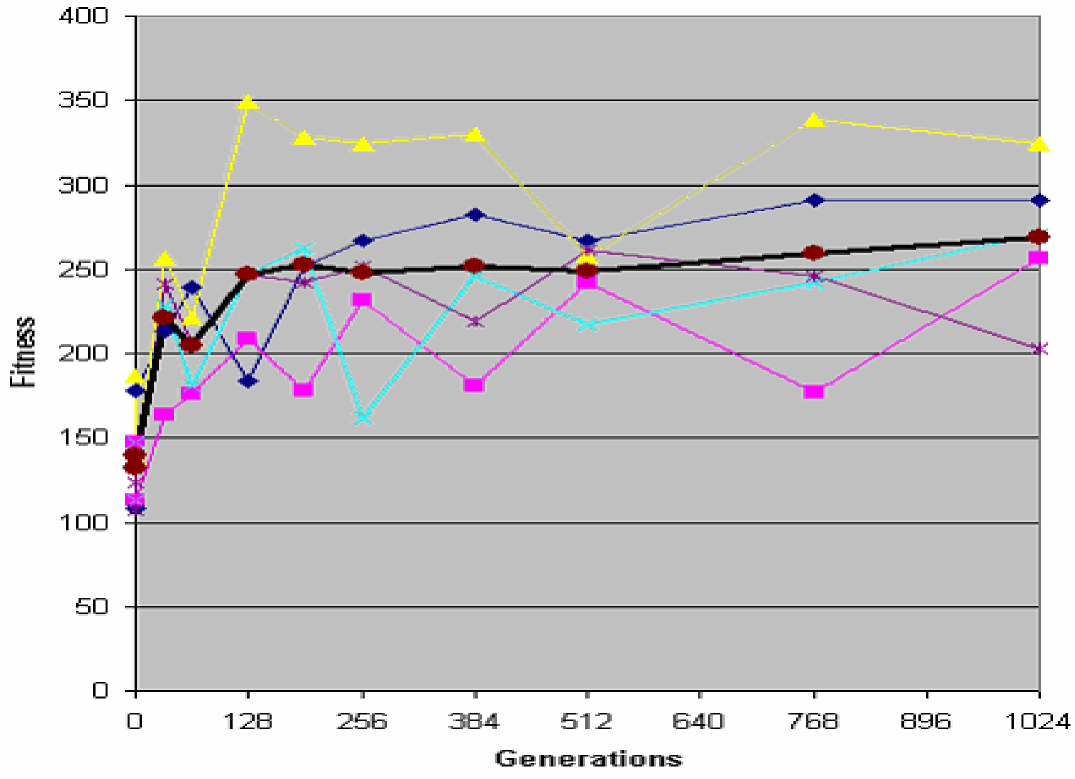

Fig 8. Fitness of the power seeking module. Each line represents the fitness of one of 5 populations; the bold line shows the average of the 5. The best individual at each generation for each population was saved and individuals from selected generations were tested on a series of common staring positions.

Observations of the robots in simulation and in the actual colony space revealed that the CGA had learned reasonable solutions. In each population, the individual with the best fitness would make a few turns if its sensors did not sense any light. If the individual was in one of the dark areas, it would circle until it moved out of the dark area or ran out of steps. If the individual robot sensed light, it would travel directly towards the power station such that it would continuously make minimal left turns until its left light sensor did not sense any light, switch to making minimal right turns until its right light sensor did not sense anything, and carry on in this manner back-and-forth. The resulting repetitive cyclic behavior was carried out until the power station area was reached.

It is evident from the results of all five runs of the navigation module that the population fitness average reached a relatively high point after several hundred generations and improved only modestly after that to a high at 1024 generations. Observations of the simulated robot revealed that its behavioral improvement was commonly characterized by its increasingly immediate and direct routes to the power station – although this improvement was more drastic in early generations than in latter ones. The best solution produced in simulation was downloaded to the actual robot. Observations of its behavior verified that the CGA learned solution was effective on the physical robot operating in the actual colony space.

VII. Conclusions

The use of a multi-loop CGA with conditional branching was shown to be an effective learning method for learning the navigation task and tests on the actual robot confirmed that the navigation control program produced a reasonable track over the ground for the robot to locate the charging station. This successful navigation behavior that was developed could be used later in learning the work segment of the self-sufficient behavior.

While our experiment was specific to our colony setup, we believe that our approach to a self-sufficient system has general relevance and can be replicated on other colony area systems. The use of capacitors and a charging station is readily applicable to small and light weight robots in which short run/charge times are desired.

There is much future work that can be accomplished in developing our self-sufficient approach. Improved capacitors rated at 2.7v and 100F are now available and would greatly improve the ability of the capacitors to hold concentrated charge at higher voltage levels. They could increase the robot's work cycle time of a self-sufficient task. Using solar or wind energy to power the charging station could make the entire system self-sufficient. We also believe that issues with multiple robot interaction in a self-sufficient system should be addressed. Numerous behavioral relationships such as cooperation and competition could be exhibited.

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

[5] 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.

[6] 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.

[7] Steels L. (1994) A case study in the behavior-oriented design of autonomous agents. In: From animals to animats 3. Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior. Clif D., Husbands P., Meyer J.-A., Wilson S.W. (eds). USA, MIT Press, 445-452.

[8] Michaud F., Robichaud E., Audet J. (2001) Using Motives and Artificial Emotions for Prolonged Activity of a Group of Autonomous Robots. To appear in: Proceedings of the AAAI Fall Symposium on Emotions. Cape Code Massachusetts, USA.

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

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

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

[12] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA, 1989.

[13] E. Tuci, M. Quinn, and I. Harvey, "Evolving Fixed-Weight Networks for Learning Robots," Proceedings of Congress on Evolutionary Computation (CEC2002), 2002.

[14] D. Bajaj and M. Ang, "An Incremental Approach in Evolving Robot Behavior," Proceedings of the Sixth International Conference on Control, Automation, Robotics and Vision, 2000.

[15] A. Ram, R. Arkin, G. Boone, and M. Pearce, "Using Genetic Algorithms to Learn Reactive Control Parameters for Autonomous Robotic Navigation," Adaptive Behavior, vol. 2, issue 3, 1994.

[16] 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.

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

[18] 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).

[19] 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.

[20] 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.

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