# Evolving Predator Control Programs for an Actual Hexapod Robot Predator

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

Basar Gulcu
Department of Computer Science and Engineering
Sabancı University
Istanbul, Turkey
basargulcu@gmail.com

***Abstract*—In the development of autonomous robots, control program learning systems are important since they allow the robots to adapt to changes in their surroundings. Evolutionary Computation (EC) is a method that is used widely in learning systems. In previous research, we used a Cyclic Genetic Algorithm (CGA), a form of EC, to evolve a simulated predator robot to test the effectiveness of a learning system in the predator/prey problem. The learned control program performed search, chase, and capture behavior using 64 sensor states relative to the nearest obstacle and the target, a simulated prey robot. In this paper, we present the results of a new set of trials, which were tested on the actual robots. The actual robots successfully performed desired behaviors, showing the effectiveness of the CGA learning system.**

***Keywords - robotics, genetic algorithm, evolutionary robotics, cyclic genetic algorithm, autonomous agent learning***

## I. INTRODUCTION

Learning robot control is an important part of developing robots. Learning the control program reduces the development time of the robot as opposed to programming the control program. Learning the control program also allows the robot to adapt to changes in its surrounding area. The predator/prey problem can be used to demonstrate the effectiveness of learning systems which produce control programs for actual robots. In this study, a learned control program was tested on a predator, which is an autonomous hexapod robot tasked to pursue a prey, which is another autonomous hexapod robot.

The predator/prey problem is well suited to demonstrate the effectiveness of robot controller learning systems. The prey tries to avoid the predator by going in the opposite direction. The predator's aim is to capture the prey. In our experiments, we did not have obstacles; except the walls and the robots themselves. The prey considers all obstacles as dangerous. It runs away from the nearest obstacle with prioritizing respectively front, middle, and back. The predator searches for the prey when it is outside of its sensor range. The predator looks for the prey and then chases it until it is captured. The predator can detect any obstacle or prey in front of it. While performing its tasks, the predator will move in response to the nearest obstacle, while ignoring the obstacles that are further away.

The method used for learning a controller was the Cyclic Genetic Algorithm (CGA), which is a form of Evolutionary Computation (EC). EC has been used by various researchers in order to learn control programs for autonomous robots. Yao used EC to learn the connection weights of an artificial neural network [1]. For a legged locomotion controller, Beer and Gallagher did their experiments by only specifying their agent's overall performance [2]. Lund and Miglino used EC to evolve a neural network controller for a Khepera robot which avoided walls and obstacles successfully [3].

Another EC method used for learning control programs is Genetic Programming (GP). Using GP, Busch et al. programmed a system to create new gaits from predefined movements [4]. The produced gaits were performed by a simulated robot. Lazarus and Hu developed a simulated robot, which successfully avoided obstacles with the use of sensors while following the walls [5]. A controller for the Khepera robot was developed by Nordin et al. with the use of GP [6].

The CGA was developed to implement loops in the control programs [7]. Although the CGA has the standard operations as Holland's Genetic Algorithm (GA) has [8], the genes of the chromosome represent tasks instead of traits; each gene is assigned tasks to execute. Parashkevov and Parker integrated Conditional Branching into CGA and experimented on the predator/prey scenario [9, 10]. The sensors were used to create 16 states. Basar and Parker used CGA on the predator/prey problem to learn controller of a simulated predator robot with 64 discrete states [11]. In the research reported in this paper, we used a new set of trial cases to test the system on an actual predator robot. The controller that was evolved by a CGA made the actual robot capable of avoiding the walls while locating, chasing, and capturing the prey.

## II. ACTUAL AND SIMULATED ENVIRONMENTS

The experiments took place in an 8'x8' area called the colony space in the lab (Figure 1). The floor was covered with a low nap carpet and was divided into 1'x1' squares to help measure the distance traveled by robots. The carpet was chosen to decrease the slippage of the legged robots. The colony space was surrounded by one foot high wooden walls.

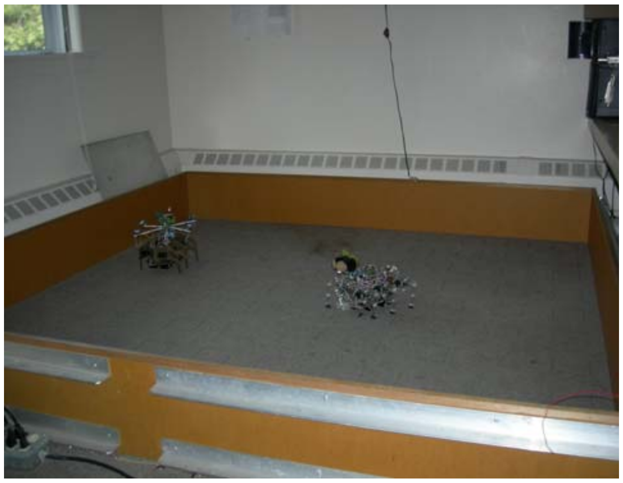


Figure 1. A photograph of the prey and the predator in the colony space.

## A. *The Prey*

The prey was a ServoBot, which is a hexapod designed by David Braun. ServoBots were designed for legged robot experimentation. The prey was made of Masonite (hard-pressed particle wood). It had six legs, three on each side, and each leg was controlled by two servo motors. Each leg had two degrees of freedom, capable of moving horizontally forward/backward and vertically up/down.

The prey had 360 degrees of vision. Six SONAR sensors (from Parallax, Inc) were placed with 60 degrees in between each. Using 0 degrees as the heading of the robot, the SONAR sensors were facing 30, 90, 150, 210, 270 and 330 degrees. Each SONAR sensor was capable of 60 degrees of vision with a range of 150 inches. The whole rack of SONAR sensors were placed on top of the controller chips. To avoid collisions with lower obstacles, the prey had two 10" antennas (tactile sensors) directed at 45 and 315 degrees. A light bulb was placed on top of the prey, to enable the predator to distinguish it from the walls (Figure 2).

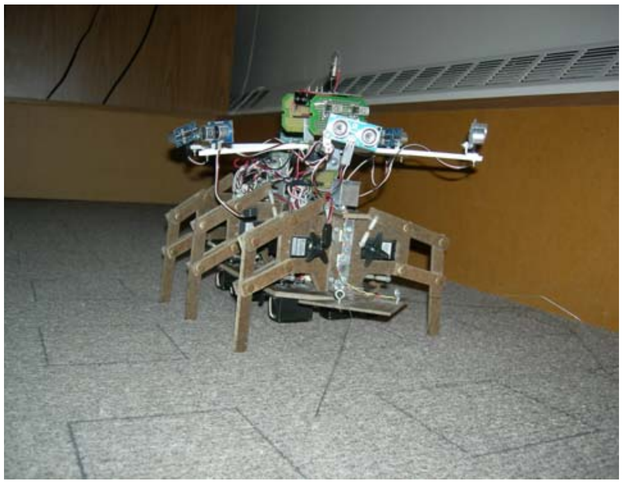


Figure 2. Photo of the prey with 6 range finding SONAR sensors and two antennas (wire touch sensors)

The prey gathered data from the sensors after each step. A step (gait) is a complete leg cycle where a foot gets back to its initial position after performing a sequence of movements. In our experiment, we used front left leg being at forward and down position as the starting point of each gait.

The prey had two controller chips: the locomotion controller and the main controller. Each has 16 usable pins. The locomotion controller was a Basic Stamp 2 (BS2, from Parallax, Inc). The servo motors were all connected to the locomotion controller (using 12 pins). Each servo was controlled by a pulse from the locomotion controller, which told it to rotate clockwise or counter-clockwise. The gait cycles were formed by a combination of movements of the 12 servo motors. Three of the controller pins were used to communicate with the other controller chip, the main controller. The main controller was a Basic Stamp 2p24 (BS2p24, from Parallax, Inc). There was no special reason for using a BS2p24 as a main controller on the prey, except to make the control configuration similar to that of the predator. All of the sensors were connected to the main controller. After evaluating the output of the sensors, the main controller determines which movement needs to be executed, and then commands the locomotion controller to perform the movement (Table 1).

TABLE I. THE TABLE SHOWS THE POSSIBLE MOVEMENTS OF THE PREY (LEFT) AND THE PREDATOR (RIGHT). THE CONTROL PROGRAM RUNNING IN THE MAIN CONTROLLER DIRECTS THE LOCOMOTION CONTROLLER BY SENDING THE 3-BIT CONTROL SIGNAL. THE PREDATOR IS DIFFERENT FROM THE PREY IN THAT 001 IS BACKWARD (DEEMED MORE APPROPRIATE FOR THE PREDATOR) INSTEAD OF WAIT.

| **Binary Message** | **Prey Movement** | **Predator Movement** |
|---|---|---|
| 000 | Forward | Forward |
| 001 | Wait | Backward |
| 010 | Right-Forward | Right-Forward |
| 011 | Left-Forward | Left-Forward |
| 100 | Rotate-Right | Rotate-Right |
| 101 | Rotate-Left | Rotate-Left |
| 110 | Backup-Right | Backup-Right |
| 111 | Backup-Left | Backup-Left |

## B. *The Predator*

The predator was also a ServoBot, although it was made of Plexiglas. The predator's sensor configuration was different than that of the prey. The predator had two antennas (tactile sensors), each of which was 9.5 inches like the prey. The purpose of the antennas was to detect obstacles which were close to the robot at 25 degrees and 335 degrees. Since antennas weren't capable of detecting objects further away two SONAR sensors (from Parallax, Inc) directed at their direction, 30 degrees and 330 degrees. There was also a third SONAR sensor directed at the heading direction of the robot. The

SONAR sensors had 60 degrees of vision with a range of 150 inches. It was mounted 5 inches above the ground level. Unlike the prey, the predator had a limited angle of vision since it had only three SONAR sensors, which were directed in the front. The predator was also equipped with two light sensors, which were facing forward. Their range was variable depending on the ambient light. Our recordings were completed in an area with ambient light, which decreased the range of the light sensors to 30 inches whereas the experiments were held in a completely dark area. The light sensors were covered with tubes that faced forward, which limited them to see only directional light sources. They were positioned to detect light from North-East and North-West (North as the direction of the robot). The light source had to be almost directly in the front in order to invoke both of the light sensors (Figure 3).

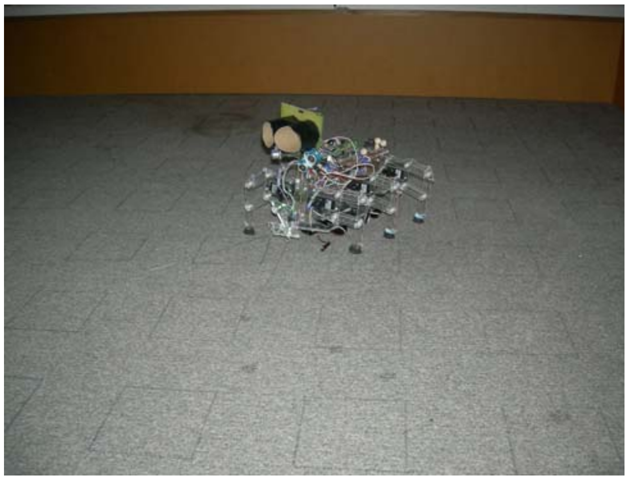


Figure 3. The photo of the predator with 2 light sensors, 3 SONAR sensors, and 2 antennas.

The predator was equipped with two chips: the main controller and the locomotion controller. All of the sensors on the predator were connected to the main controller chip, which was a BS2p24. A BS2p24 was essential for the predator to implement the control program that was learned. Like the prey, the predator also had a locomotion controller, which was a BS2. It worked exactly the same as the prey's locomotion controller. Three pins were used for the main controller to command the locomotion controller to execute eight different movements.

Depending on the output of the sensors, the predator determined which action to execute. In order to allow the learning to operate at a higher level, we developed the processing needed to transform the raw sensor data into 8 categories of the robot's situation relative to both the nearest obstacle and the target (prey). The predator's state at any point in time was the combination of these two factors. Since there were 8 possible situations for the nearest obstacle, and 8 for the target, there were 64 possible combinations (Table 2).

### C. *Simulation*

The learning of the robots took place in a simulated area of 300x300 units, in which each robot was represented with an x and y coordinate and a direction between 0-359. In the simulation area, the point (0, 0) was positioned on the bottom left corner and the direction 0 was to the East of the area. The program was written in Java.

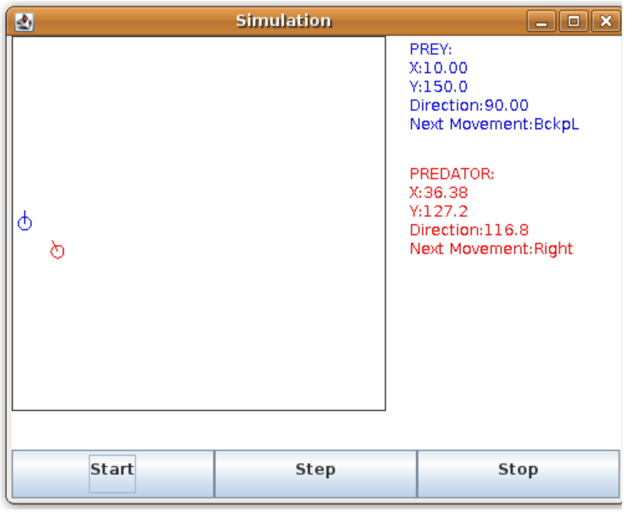
TABLE II. THE TABLE SHOWS THE EIGHT POSSIBLE SENSOR SITUATIONS RELATIVE TO THE NEAREST OBSTACLE AND EIGHT RELATIVE TO THE TARGET. THE COMBINATION OF THESE DEFINES THE STATE OF THE ROBOT.

| **Obstacle** | **Target** |
|---|---|
| no_object | no_target |
| near_right | near_right |
| far_right | far_right |
| near_left | near_left |
| far_left | far_left |
| near_front | near_front |
| middle_front | middle_front |
| far_front | far_front |

A population of predators was created to be tested against a pre-programmed prey robot. During each step of the learning process, the simulation updated the positions of the robots and their sensors with respect to each other and the walls (there were no other obstacles). The learning system took in the desired number of generations for the agents to evolve as a parameter and while operating, the average fitness of the population was output every 10th generation. In the 1st, 100th and the 300th generations, the whole population was printed out with the corresponding fitness values. For observation purposes, each robot was represented by a circle and a line pointing at the direction of the robot. In the simulation, we were able to position the agents as we wanted, and move step by step to see which movement was going to be executed as a result of their decision (Figure 4).

The data used to determine movement by the simulated robots was measured by the performance on the actual robots. The measurements taken were the changes in distance traveled in the direction of the heading, distance traveled perpendicular to the initial heading, and heading after the move. For example, for the prey ServoBot, a command of Forward (Table 1) involved distance change of 5 units along the initial heading, a 0.05 unit perpendicular distance change, and a 4.6 degree change from the initial heading. A command of Right-Forward (Table 1) involved a distance change of 2.38 units along the initial heading, a 1.11 unit perpendicular distance change, and a -39 degree change from the initial heading. These changes were used to calculate the new x and y position of the simulated robot and its new orientation after each move.

The Prey and Predator simulations included the list of values for each movement (measured from executing the command on the actual robot) and the decision function that determined which movement to make.



PREY:
X:10.00
Y:150.0
Direction:90.00
Next Movement:BckpL

PREDATOR:
X:36.38
Y:127.2
Direction:116.8
Next Movement:Right

Figure 4. A random screenshot of the Simulation. Each agent is represented with a circle and a line showing its direction. The prey is the blue one and the predator is the red one.

## III. Genetic Algorithm

Evolutionary Computation (EC) in this study to evolve a control program for the predator to catch the prey. It was used with a computer simulation to run a population of possible solutions to determine their fitness values.

The type of EC used was the Cyclic Genetic Algorithm (CGA). It is capable of learning a cyclic combination of decisions/actions, which are coded in the chromosome [7]. The chromosome can be divided into blocks that contain the movement primitive and the number of times that it should be repeated. These blocks can also represent conditionals that control the process of execution. The CGA provides a method for learning control programs that produce cyclic behavior.

### A. *The Cyclic Genetic Algorithm Applied to the Predator/Prey Problem*

For the predator/prey problem in this study, it was determined that only one action was needed for each of the possible sensor inputs. This action could continue until the sensor situation changed. A CGA with conditional branching [9] that has only one instruction in each loop could be used. In effect, this would be functionally the same as a fully connected finite state machine with control returning to the present node if there are no changes. A population of 256 randomly generated chromosomes was used for this problem.

The predator can only be in 64 (8*8) different states (Table 2). Therefore, a chromosome with 64 genes is sufficient for the CGA to learn a controller for the predator. Each block of the chromosome (gene) represents the action to be taken. When the condition is met for a gene, the movement in that block takes place. An example chromosome is shown in Figure 5. Since each movement is represented with 3 bits and there are 64 states, each chromosome is 192 bits. If the robot does not sense an obstacle or the prey, the action taken (Table 1) is the one corresponding to the 3 bits of the first block. In this case, it would execute movement 001, which is move backwards.

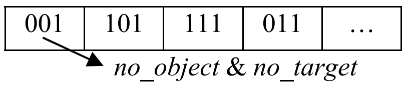


Figure 5. Example chromosome divided into blocks. The total length is 192 bits. Each block holds a movement corresponding to that state (a combination of sensor readings). The first block holds movement for the state of *no_object & no_target*.

With random positioning, the possibility of a certain gene not being visited was high. Therefore, at each generation, 10 different starting positions were randomly selected. Each of the individuals of the population was tested using these starting positions by running for 200 steps (gait cycles) in pursuit of the prey. Each individual was evaluated by the average distance that they approached the prey. The maximum distance that the predator and the prey could be was 424.26 (since the field is a square of 300 units). Therefore Equation 1 was used to calculate positive change.

$$score = maximum\_distance - distance\_between\_agents \qquad (1)$$

The score is calculated after each step. If the score of the individual was greater than 403.05 (95% of the maximum score), it means the predator was close enough to catch the prey. In this case the score was doubled as a reward for the capture. To encourage a rapid capture, a bonus that decreases as the number of steps increases is added to the score (Equation 2).

$$n^{th} step\_score = score + \frac{score}{n^2} \qquad (2)$$

At the 1st step, the bonus is as much as the score. As the run progresses, the bonus quickly becomes negligible. The fitness value was average of these step scores. Since we let each individual to run 200 steps from each starting position, and there were 10 starting positions, they took 2000 steps. Hence the fitness value was the total sum of the step scores divided by 2000.

The roulette wheel method of selection was used. An individual's chance of selection for the next generation was biased by its fitness. The more successful an individual was, the more chance it had to be involved in producing the next generation. Two-point crossover was performed on the two selected individuals. The newly formed chromosome was subjected to a mutation function, which inverts a bit (if the bit is 0, make it 1 and vice versa) by a chance of 0.003%. This process was repeated 256 times and the next generation was formed. The population size was constant.

## IV. Results

Five trials, where the initial population was randomly generated and the CGA ran for 300 generations, were conducted. The average of the individuals at each generation was highly varying due to random generation of the starting positions. In order to standardize the comparison, we randomly generated 100 starting positions before the simulation. Every 10th generation, individuals also run from these starting positions and the data was recorded (Figure 6). The improvement of individuals increase vigorously until 100th generation and the average fitness did not improve significantly after the 170th generation.

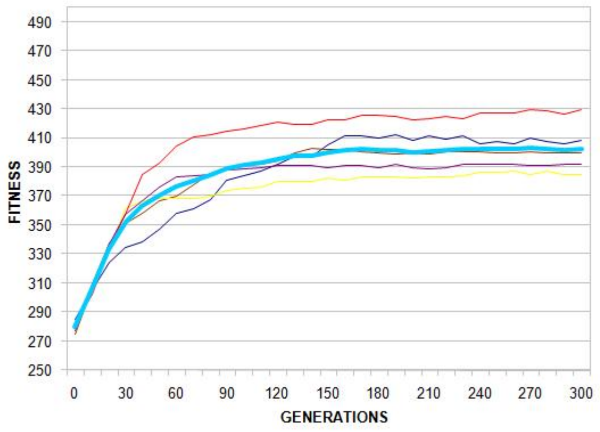


Figure 6. The average fitness for 100 fixed positions over 300 generations in 5 trials are displayed. The bold line is the average of all trials.

From each of the five trials, individuals from 1st, 100th and 300th generations were picked to test the effectiveness of the developed control program on the actual robot. In all of these tests, the predator was placed at south east corner, 10 inches away from the walls, heading to west. The prey was placed in the middle of the west wall, heading to north (Figure 7).

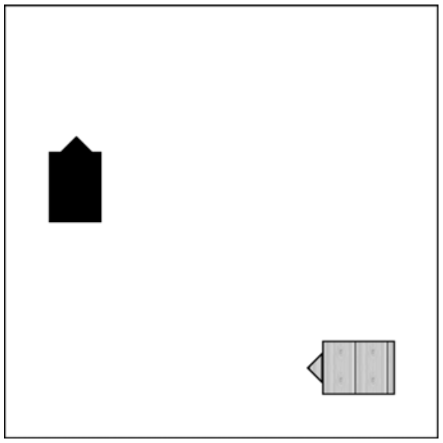


Figure7. Diagram of the actual robot test. The light colored (striped) object is the predator directed to West. The shaded object is the prey directed to North.

In each trial, the robots were allowed to run until the predator caught the prey or for 100 steps, whichever came first. The results of these tests were shown in Figure 8. As can be seen, the actual tests verify the effectiveness of the learning system.

A detailed description of the movements of the robot using control programs from trial 4 follows. In each test, the predator started with a left turn while the prey was located at the west of the colony. Figures 9, 10 and 11 depict the movements of the robots in the 1st, 100th and 300th generations respectively. The black robot is the prey and the light robot is the predator.

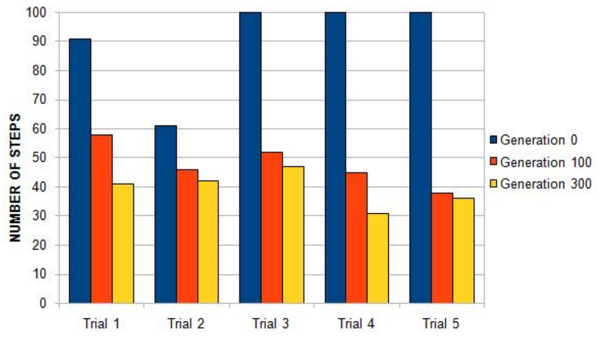


Figure8. Numbers of steps of the randomly selected three individuals from 1st, 100th and 300th generations over 5 trials are displayed. Each individual was allowed to run until capture, but no more than 100 steps.

In the 1st generation, the robot began with a left rotation (Rotate-Left, see Table 1). While it was rotating, it got closer to the South wall and started to perform the Left-Forward movement. The result was that the robot rotated to be perpendicular to the South wall. Facing the South wall, it continued to attempt to go straight and the trial ended as the robot kept pushing the South wall. While the predator was moving as such, the prey moved toward the opposite corner and then turned to head to the center of the colony, which gave it several escape options.

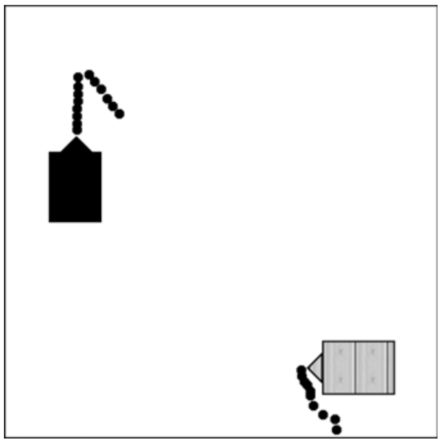


Figure 9. The predator rotated left, performed a Left-Forward and then went straight to the South wall.

At the 100th generation the predator made a couple of Left-Forward movements, bumped to the South wall, but kept performing several more left turns. While turning left, the predator got close to the South-East corner of the area, hence performed a back-up right (the robots' heading changes to left), which gave it an advantage on turning to the left. As a result, it oriented itself in the right direction to sense the prey. When the predator detected the prey, the prey was turning toward the center of the colony space from the North-West corner as in the previous test. It was moving away from the corner, which turned it in a direction to head toward the predator. Since the prey was trying to get away from the closest obstacle, it started to head towards the predator. By the time the distance between the prey and the wall was greater than the distance between the prey and the predator, the predator was moving straight toward the prey. Although the prey tried to run away from the predator with a right rotation, the gap closed before it could make the turn and the trial ended with a capture.

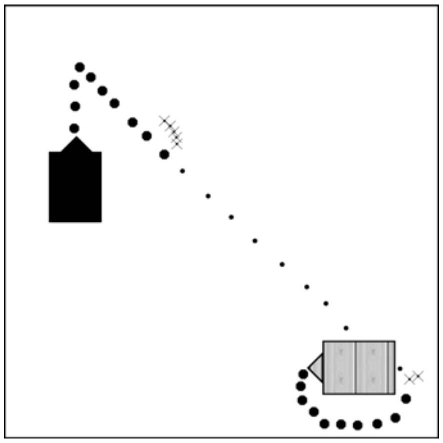


Figure 10. The predator performed a series of Left-Forward, then Backup-Left (where the crosses are) and then went straight.

At the 300th generation, the predator started with a right turn. Since the prey was placed by the west wall, starting with a right turn favored the predator. Once the prey was detected, the predator tracked directly toward it and made the capture shortly after it left the North-West corner. The result was a significant improvement in the capture time.

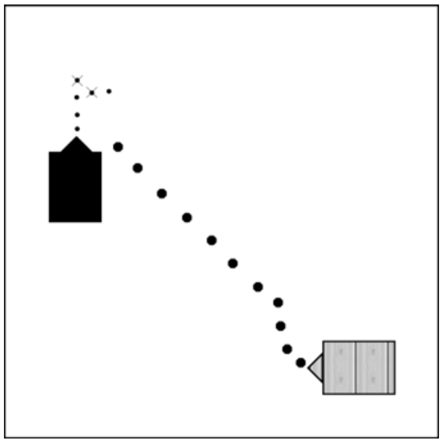


Figure 11. The predator rotated right and went straight.

Since the control program was generated randomly at 1st generation, we didn't expect the individuals from 1st generation to capture the prey. These controllers were not capable of avoiding walls, much less tracking toward the prey. By the 100th generation, the controllers were capable of avoiding walls and tracking toward the prey. The changes from the 100th to the 300th generations were minor as the robot improved on the effectiveness of its wall avoidance and tracking capabilities.

## V. Conclusions

The results show that the CGA can learn an effective control program for a predator in the predator/prey problem. The tests on the actual robots also show that the learning method was effective. The CGA learned the proper actions in response to 64 different possible sensor inputs. In the initial/random population, the predator would not get close to the prey except by chance. After 100 generations of training, the trained predators were able to avoid walls and chase the prey. The tests on actual robots matched with the results of the simulation.

The next step will be to use the CGA learning method to evolve the prey. Since our final predator controller was successful at capturing the prey, we want to determine if a prey controller can be evolved that will allow it to successfully evade the current best predator. If this is the case, we will experiment with competitive co-evolution as both the predator & prey learn concurrently.

## References

[1] Yao, X. Evolving artificial neural networks. *Proc. Computation,* (2001), 283-289. *IEEE*, *87*, 9 (1999), 1423-1447.

[2] Beer, R. D. and Gallagher, J. C. Evolving dramatical neural networks for adaptive behavior. *Adaptive Behavior, 1*, 1 (1992), 91-122.

[3] Lund, H. H. and Miglino, O. From simulated to real robots. *Proc. IEEE Third International Conference on Evolutionary Computation, NJ* (1996).

[4] Busch, J., Ziegler, J., Aue, C., Ross, A., Sawitzki, D. and Banzhaf, W. Automatic generation of control programs for walking robots using genetic programming. *EuroGP 2002, LNCS 2278* (2002), 258-267.

[5] Lazarus, C. and Hu, H. Using genetic programming to evolve robot behaviours. *Proc. Third British Conference on Autonomous Mobile Robotics & Autonomous Systems, Manchester, UK* (2001).

[6] Nordin, P., Banzhaf, W. and Brameier, M. Evolution of a world model for a miniature robot using genetic programming. *Robotics and Autonomous Systems*, *25* (1998), 105-116.

[7] Parker, G. B. Generating arachnid robot gaits with cyclic genetic algorithms. *Genetic Programming 1998: Proc. of the Third Annual Conference,* (July 1998), 576-583.

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

[9] Parker, G. B., Parashkevov I. I., Blumenthal, H. J. and Guildman, T. W. Cyclic genetic algorithms for evolving multi-loop control programs. *Proc. of the World Automation Congress (WAC '04)* (June 2004).

[10] Parker, G. B. and Parashkevov, I. Cyclic genetic algorithm with conditional branching in a predator-prey scenario. *Proc. of the 2005 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2005)* (Waikoloa, Hawaii, October 2005).

[11] Parker, G. B. and Gulcu, B. Evolving predator control programs for a hexapod robot pursuing a prey. *Proc. of the World Automation Congress International Symposium on Intelligent Automation and Control (ISIAC 2008)* (Waikoloa, Hawaii, October 2008.