9-2008

Evolving predator control programs for a hexapod robot pursuing a prey

Gary Parker
Connecticut College, parker@conncoll.edu

Basar Gulcu

Recommended Citation
Parker, G.; Gulcu, B., "Evolving predator control programs for a hexapod robot pursuing a prey," Automation Congress, 2008. WAC 2008. World , vol., no., pp.1,7, Sept. 28 2008-Oct. 2 2008

Evolving predator control programs for a hexapod robot pursuing a prey

Keywords
Cyclic Genetic Algorithms, Predator, Prey, Problem Simulation

EVOLVING PREDATOR CONTROL PROGRAMS FOR A HEXAPOD ROBOT PURSUING A PREY

GARY PARKER, CONNECTICUT COLLEGE, USA, parker@conncoll.edu
BASAR GULCU, CONNECTICUT COLLEGE, USA, bgul@conncoll.edu

ABSTRACT

Control program learning systems for autonomous robots are important to assist in their development and to allow them to adapt to changes in their capabilities and/or the environment. A common method for learning in robotics is Evolutionary Computation (EC) and a good problem to demonstrate the effectiveness of a learning system is the predator/prey problem. In previous research, we used a Cyclic Genetic Algorithm (CGA), a form of EC, to evolve the control program for a predator robot with a simple sensor configuration of 4 binary sensors, which yielded 16 possible sensor states. In this paper, we present the use of a CGA to learn control for a predator robot with a more complicated sensor setup, which yields 64 sensor states. The learning system successfully evolved a control program that produced search, chase, and capture behavior in the simulated predator robot.

KEYWORDS: Cyclic Genetic Algorithms, Predator, Prey, Problem Simulation

1. INTRODUCTION

Robot control is an important issue. Systems that can learn control programs reduce development time and allow the robot to adapt to changes in its capability to move. One interesting problem that can be used to test control learning systems is the predator/prey problem. For this, the robots must evolve to find an optimal/efficient way for the predator to catch the prey. In other words, the robot must develop a strategy to maintain its existence. In this study, a control program was learned for an autonomous hexapod robot, the predator, which allowed it to catch another autonomous robot, the prey. The mechanism for learning was a Cyclic Genetic Algorithm (CGA).

Several researchers have used Evolutionary Computation (EC) for learning control programs for robots. The most common use is for learning connection weights and/or architectures in artificial neural networks [1]. Beer and Gallagher states that one of the significant advantage of this approach is that one needs to specify only a measure of an agent's overall performance [2]. As a result of their experiments, legged locomotion controllers were evolved successfully. An additional example is the evolving of a neural network controller for a Khepera robot in work by Lund and Miglino [3]. The robot was able to avoid walls and obstacles.

Genetic Programming (GP) is another EC method used for learning robot control. A system developed by Busch et al. [4] was able to produce gaits free from the robots' predefined movements. The evolved controller was used in simulated robots. Lazarus and Hu's simulated robot was capable of avoiding obstacles while following walls with the assistance of sensors [5]. Nordin et al. also created a system in which a controller for the Khepera robot was evolved by GP [6].

In order to implement loops in evolved programs, the CGA was developed [7]. Like Holland's Genetic Algorithm (GA), it has the standard operations: selection, crossover, and mutation [8]. In the CGA, however, the genes of the chromosome represent tasks to execute as opposed to traits of a solution. Parashkevov and Parker integrated Conditional Branching into CGA and experimented on the predator/prey problem [9, 10]. The use of sensors was limited to on and off with a total of 16 possible sensor state combinations. In our research, we implement the CGA on the predator/prey problem with an expanded set of possible sensor responses. Since some of the sensors are capable of detecting distance we can create additional discrete states making a total of 64 combinations.

The controller that was evolved by a CGA is capable of avoiding obstacles while searching for and chasing the prey.

2. PREDATOR & PREY PROBLEM

A good test bed for learning in robotics is the predator/prey problem. In this problem, the prey does everything possible to evade capture while avoiding obstacles. The predator's goal is to capture the prey while avoiding obstacles. For our experiments, the prey perceives all obstacles (walls, predator) as hazardous. If there is a nearby obstacle, it moves away from it. Otherwise, it stays where it is. The predator looks for prey. Its task can be split into two parts: search and chase. It can detect any obstacle or target that is in front of it. If one obstacle that it detects is closer than another, the robot will ignore the further obstacle. If the target is detected, the predator is to track toward it.

2.1 The Colony Space

The colony space (Figure 1) that is being used for the experimentation is a 8x8foot area in the lab. Within the colony space, the floor is divided up into 1x1 foot square blocks to measure the movement of each agent. The walls of the colony space are made up of a wooden border that is one foot in height. The floor is covered with low nap carpet, which provides enough friction for minimal slippage in the movement of the legged robots.

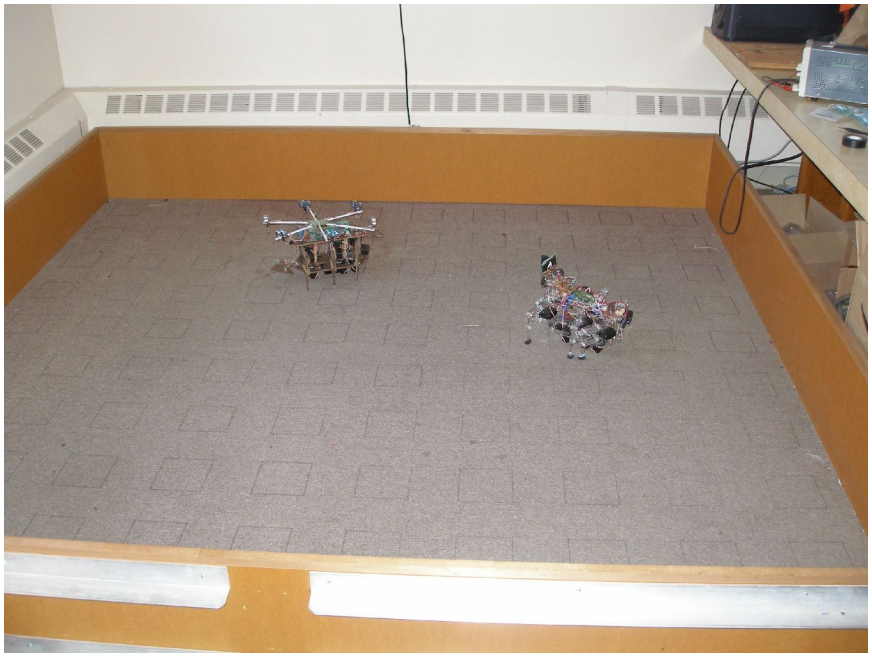

Figure 1. A photograph of the prey and the predator in the colony space.

2.2 The Prey

The prey (Figure 2) is a ServoBot, which is a hexapod robot designed by David Braun for legged robot experimentation. The body and legs of the prey are made of Masonite (hard-pressed particle wood). It has twelve servo motors, two per leg, which provides two degrees of freedom per leg. Each leg can move vertically up/down and horizontally forward/back.

The prey is equipped with six SONAR sensors (from Parallax, Inc.), which are positioned to face 60 degrees apart to provide 360 coverage. Using 0 degrees as the heading of the robot, the first sensor is 30 degrees and the rest of the sensors face in the directions of 90, 150, 210, 270 and 330 degrees. Since each SONAR sensor has approximately 60 degrees of vision; there are no blind spots. The range of each SONAR sensor is approximately 150 inches. The whole rack of SONAR

sensors is placed on top of the robot. In addition to the SONAR sensors, there are 10 inch antennas pointing 45 degrees and 315 degrees. These tactile sensors are mounted lower than the SONAR sensors to provide protection against collisions with low obstacles.

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

Binary Message	**Prey Movement**	**Predator Movement**
000	*Forwards*	*Forwards*
001	*Wait*	*Backwards*
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*

Table 1. Possible movements of the prey (left) and 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 Backwards (deemed more appropriate for the predator) instead of Wait.

Control for the prey is provided by two micro-controllers; the locomotion controller and the main controller. Both of them are BASIC Stamp 2 (BS2). There are 16 usable pins on each controller. All the servo motors (there are 12 servos) are connected to the locomotion controller. Each pin on the controller can output 5V. Twelve of the pins are connected to each of the 12 servo

motors and can rotate them either clockwise or counter-clockwise. The movement continues until the leg reaches its maximum position. Combinations of servo motor rotations with corresponding leg movements can produce gaits for locomotion. Three of the 16 pins are used to communicate with the other controller chip, the main controller.

The main controller is connected to all of the sensors. After gathering information from the sensors, it directs the locomotion controller to execute one of the 8 movements (Table 1).

Two LEDs are placed on top of the prey, to enable the predator to distinguish it from a wall. The predator does not have a light, so the prey cannot distinguish it from any other obstacle.

2.3 The Predator

The predator (Figure 3) is also a ServoBot. However, the body and the legs of the predator (ServoBot) are made of Plexiglas.

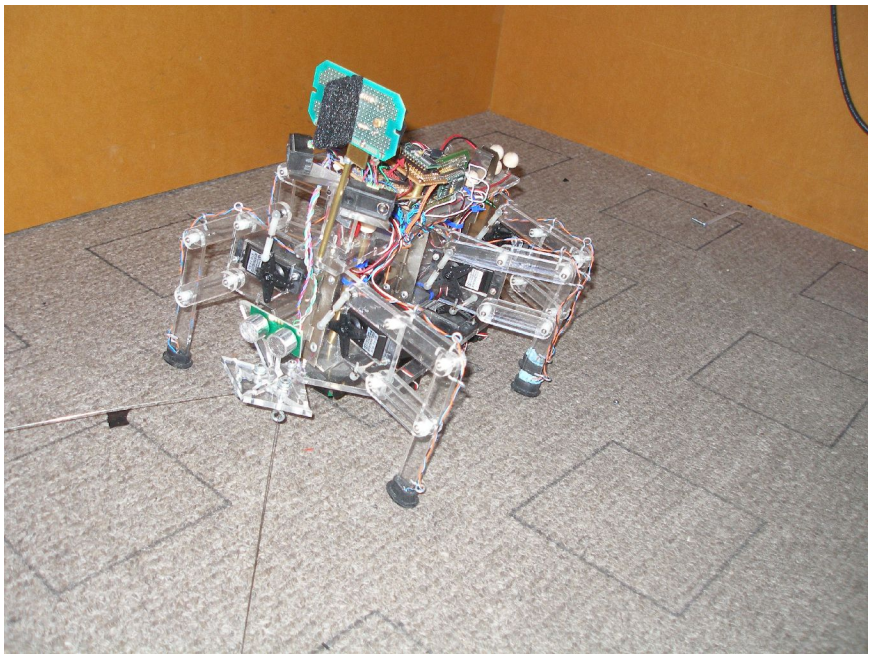

Figure 3. Photo of the predator with 2 light sensors, 2 infra-red sensors, 1 SONAR sensor, and 2 antennas..

The predator is equipped with one SONAR sensor (from Parallax, Inc.) which faces in the direction of 0 degrees. Its range is approximately 150 inches with a 60 degrees of angle of vision. The predator also has 9.5 inch antennas, 50 degrees apart, pointing 25 and -25 degrees. It has two Infra-Red sensors pointing 30 and -30 degrees. Each of these Infra-Red sensors has a range of 50 inches. There are also two light sensors with a separator in between pointing to the front. The range of the light sensors is approximately 60 inches and they have approximately 90 degrees of angle of vision. The separator allows the robot to determine if a light source is coming from the left or right. However, if the light is within 7.5 degrees of the robot's heading both sensors will detect it.

The predator, as with the prey, has two BS2 controller chips; one the main controller and the other the locomotion controller. The main controller gathers data from the sensor outputs, and directs the locomotion controller as to what movement to make.

Depending on the input from sensors, the predator determines which action to take. In order to allow the learning to operate at a higher level, we developed the processing needed to transform the sensor data into 8 categories (Table 2) of the robot's situation relative to both the nearest obstacle and the target (prey). The predator's state at any point in time is the combination of these two

results. Since there are 8 possible situations for the nearest obstacle, and 8 for the target, there are 64 possible combinations.

OBSTACLE	**TARGET**
none	*none*
near_right	*near_right*
far_right	*far_right*
near_left	*near_left*
far_left	*far_left*
near_front	*near_front*
medium_front	*medium_front*
far_ahead	*far_ahead*

Table 2. 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.

The predator's control program uses the state to determine the movement desired to search for or to catch the prey. It directs the locomotion controller to perform the desired movement. The possible movements are shown in the Table 1. The control program sets three of the main controller's pins, which are connected to three of the locomotion controller's pins by direct lines.

3. SIMULATION OF THE ACTUAL PROBLEM

The simulation area is square with each wall 300 units in length. There are no obstacles, except the walls of the area. The robots' positions in the simulation area are fully described by their X and Y coordinates, as well as a number between 0 and 359 that shows the direction of their heading. (0, 0) is positioned as bottom-left corner of the area.

The simulation is written in Java with the Robot Class defined abstractly. Calculations that are required for both the predator and the prey, including sensor output, movement check, movement and the sub functions of these, are done in this class. Predator and Prey are subclasses of Robot.

Every sensor has it angle of visibility, type, and value. If an obstacle is detectable by the sensor, then the distance between the obstacle and the sensor's owner is calculated and the output is updated. In this particular simulation, since there are no free obstacles other than the prey for the predator, and the predator for the prey, most of the sensors are showing the distance to the walls. In order to calculate the distance to the walls, the program must determine which sensor is directed at which wall. Once all the sensors are updated with the distance from the robot to the walls, each sensor's direction is compared with the absolute angle between the prey and the predator. If the values match, then the program calculates the distance between and updates the sensor.

Each movement is defined by its distance change in the direction of its initial heading, its distance change perpendicular to the initial heading and orientation change relative to the initial heading. For example, for the prey ServoBot, a command of Forwards (Table 1) involves 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 (Table 1) involves 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 are used to calculate the new x and y position of the robot and its new orientation.

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

4. CYCLIC GENETIC ALGORITHM TO LEARN PREDATOR CONTROL

Evolutionary Robotics (ER) is the application of Evolutionary Computation (EC), a learning technique based on heredity and survival of the fittest, to generate a control program for the robot. It allows robots to evolve over time and generate the best way to perform their assigned task, which, in this study, is to catch the prey. The form of EC that we used in this research is the Genetic Algorithm (GA). As with other EC techniques, it is a search technique used in ER to find approximate solutions for robot control problems. It is typically used with a computer simulation to run a population of possible solutions to determine their fitnesses. Each individual in the population is a chromosome, which is a set of bits that represents a control program for the robot. The fitness function evaluates each individual and assigns a fitness. The higher score a chromosome receives, the higher possibility of its selection for use in producing the next generation. Two chromosomes are selected and are combined through crossover with the new individual subjected to mutation. This is repeated several times until a new population is formed.

The Cyclic Genetic Algorithm (CGA) is a type of GA. It is capable of learning a cyclic combination of decisions/actions, which are coded in the chromosome. 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.

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 [8] 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.

Since there are 64 (8*8) states that the predator can be in (Table 2), a chromosome defining 64 movement blocks is sufficient for the CGA to learn how to control the predator. Each block of the chromosome represents the action to be taken. When the condition is met for a certain block, the movement in that block takes place. An example chromosome is shown in Figure 4. There are 64 blocks of 3 bits each, making it a 192 bit chromosome. If the robot does not sense an obstacle or the prey, the action taken (Table 1, predator) is the one corresponding to the 3 bits of the first block. In this case, it would execute movement 001, which is move backwards.

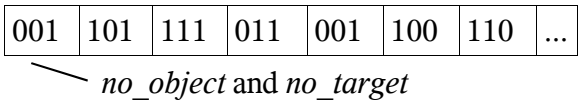

Figure 4. Example chromosome divided into blocks. The total length is 192 bits. Each block holds a movement corresponding to that state (a combination of sensor readings). The first block holds movement for the state of no_object and no_target.

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 100 steps 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 can be is 424.26 (since the field is a square of 300 units). The fitness is calculated by using Eq. (1) and Eq. (2).

$$score = maximum\_distance - distance\_between\_predator\_prey \quad (1)$$

$$f(x) = score + score/(number\_of\_steps\_taken^2) \quad (2)$$

The fitness value is calculated after each step with the highest fitness stored. If the score is greater than 405, it means the predator caught the prey, and as a reward, the score is multiplied by three.

The roulette wheel method of selection is 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. Once the resultant chromosome was formed, it was subjected to a mutation function. The mutation function went through each bit of the string with a 1/300 chance that the bit would be inverted (if the bit is 0, make it 1 and vice versa). This process is repeated for 256 times and the next generation was formed.

5. RESULTS

The CGA ran for 300 generations. Since the individuals at each generation were run on 10 randomly selected locations, the fitness values from generation to generation were highly variable. For example, if the randomly selected positions of the robots were close to each other, the overall fitness of that generation was high. To make a useful comparison between generations, 100 positions were picked at the beginning of each trial. In every 10th generation, all individuals from the population were ran from these fixed positions. Figure 5 shows that the improvement of the whole population increased vigorously in the first 50 generations and the improvement slowed down. One can see that the average did not improve significantly after the 170th generation, but changes can be seen in the individual trials.

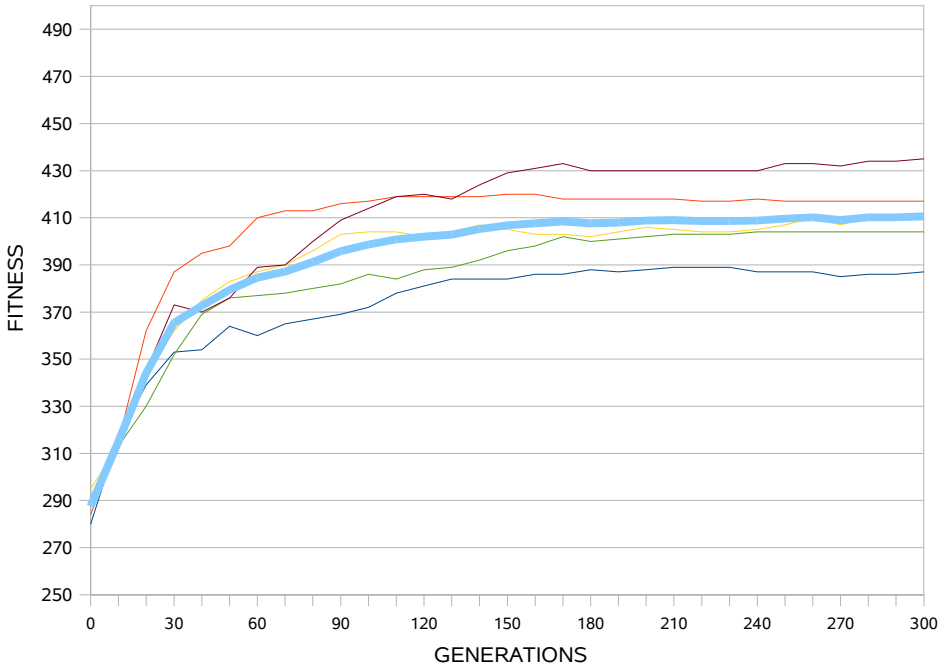

Figure 5. The average fitnesses for 100 fixed positions over generations in 5 trials are displayed. The thick line is the average of all trials.

In addition to plotting the fitness growth of the populations, we observed the behavior of the best individuals in the population of one of the trials. These observations were made of individuals from generation 1, 50, 130 and 270. In the 1st generation, the best individual did not tend to reach the prey at all. It was moving backwards in most of the cases. In the 50th generation, the predator was not successful in locating the prey. However, once the predator detected the prey, it was able to chase it. In the 130th generation, although the predator seemed to act properly in most of the situations and perform locating and chasing properly, there were a few unexpected movements

while chasing the prey. In the 270th generation, the best individual performed its purpose properly. In addition to being able to follow the trail of the prey, it would turn sharp enough at the start to take a shortcut in reaching the prey. The observations of the results showed great improvement throughout the learning process.

6. CONCLUSIONS

The results show that the CGA can learn an effective control program for a predator in the predator/prey problem. 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 50 generations of training, the CGA learned the needed responses to chase the prey. Further evolution resulted in both expected and unexpected results. The expected result was that the predator learned to search for and catch the prey. The unexpected result was that the predator learned to take a shortcut to reach the prey.

The successful process was executed in simulation on robots that closely modeled actual robots. The next step will be to test our results on the actual robots. In further research, the CGA learning method will be used 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. If this is the case, we will experiment with competitive co-evolution as both the predator & prey learn concurrently.

7. REFERENCES

[1] X. Yao, "Evolving artificial neural networks," Proc. Computation, 2001, pp. 283-289. IEEE, Vol. 87, No. 9, 1999, pp.1423-1447.

[2] R. D. Beer and J. C. Gallagher, "Evolving dramatical neural networks for adaptive behavior," *Adaptive Behavior*, Vol. 1, No. 1, 1992, pp. 91-122.

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

[4] 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.

[5] 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.

[6] 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.

[7] G. B. Parker, "Evolving leg cycles to produce hexapod gaits," Proc. World Automation Congress, Vol. 10, Robotic and Manufacturing Systems, 2000, pp. 250-
255.

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

[9] G. B. Parker, I. I. Parashkevov, H. J. Blumenthal, and T. W. Guildman, "Cyclic genetic algorithms for evolving multi-loop control programs," Proc. of the World Automation Congress (WAC 2004), June 2004.

[10] G. B. Parker, and I. Parashkevov, "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), October 2005, Waikoloa, Hawaii.