4

# Evolving Controllers for Miniature Robots

Michael Botros

Department of Computer and Electrical Engineering,
Faculty of Engineering, McMaster University,
1280 Main St. West, Hamilton, Ontario , Canada L8S 4K1,
botrosmw@mcmaster.ca

Using traditional path planning and artificial intelligence techniques has restricted the use of mobile robots to limited tasks in previously known environments, yet potential applications include dynamic and unstructured environments. One of the very promising methods of designing controllers for autonomous and mobile robots is using Evolutionary Computations, a class of algorithms which mimics the natural evolution process.

In this chapter we present a series of experiments in evolutionary robotics that used the miniature mobile robot Khepera. Khepera robot is widely used in evolutionary experiments due to its small size and light weight which simplify the setup of the environments needed for the experiments. The controllers evolved by the presented experiments include classical and spiking neural networks controllers, fuzzy logic controllers and computer program obtained by Genetic Programming. The tasks performed by the robots through the experiments reflect learning many basic as well as high level behaviors. These behaviors include: navigating in dynamic environment with static or dynamic obstacles, seeking and following the light sources present in the environment, returning home for recharging the battery, and collecting trash objects from the environment. The chapter also presents an experiment in co-evolution in which a predator-prey behavior is learned by two robots. The chapter ends with an experiment that evolves spiking neural networks, a new artificial neural networks model that accurately models the biological neuron activation. This experiment presents the use of evolution to obtain a spiking neural network that enables the robot to navigate depending only on vision information.

## 4.1 Introduction

Khepera is a miniature mobile robot that is widely used in laboratories and universities in conducting experiments aiming at developing new control algo-

rithms for autonomous robots. It was developed by the Swiss Federal Institute of Technology and manufactured by K-team [1] [2]. Khepera robot is cylindrical in shape with a diameter of 55 mm and a height of 30 mm. Its weight is about 70 gm. Its small size and weight made it ideal robotic platform for experiments of control algorithms that could be carried out in small environments such as a desktop.

The robot is supported by two wheels; each wheel is controlled by a DC motor that can rotate in both directions. The variation of the velocities of the two wheels, magnitude and direction, will result in wide variety of resulting trajectories. For example if the two wheels rotate with equal speeds and in same direction, the robot will move in straight line, but if the two velocities are equal in magnitude but different in direction the robot will rotate around its axis.

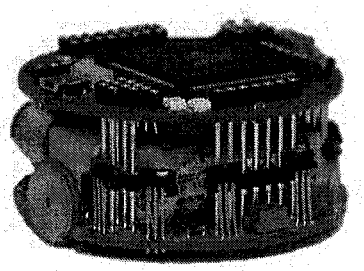

**Fig. 4.1.** Miniature mobile robot Khepera (with permission of K-team).

The robot is equipped with eight infrared sensors. Six of the sensors are distributed on the front side of the robot while the other two are placed on its back. The exact position of the sensors is shown in figure (4.2). The same sensor hardware can act as both ambient light intensity sensor and proximity sensor.

Each of the eight sensors consists of emitter and receiver parts so that these sensors can function as proximity sensors or ambient light sensors. To function as proximity sensors, it emits light and receive the reflected light intensity. The measured value is the difference between the received light intensity and the ambient light. This reading has range [0, 1023] and it gives a rough estimate how far the obstacles are. The higher reflected light intensity the closer obstacles are. It should be noted that we cannot find a direct mapping between the sensor reading and the distance from the obstacle, as this reading depends on factors other than the distance to the obstacle such as the color of the obstacle.

To function as ambient light sensors, sensors use only receiver part of the device to measure the ambient light intensity and return a value that falls in

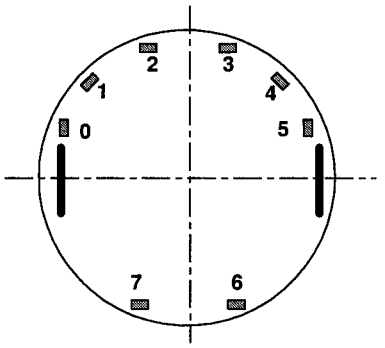


**Fig. 4.2.** The position of the eight sensors on the robot (with permission of K-team)

the range of [0, 1023]. Again, these measurements depend very strongly on many factors such as the distance to the light source and its direction.

An interesting feature of the Khepera robot is its autonomy, which includes autonomy of power and control algorithm. For the purpose of power autonomy, the robot is equipped with rechargeable batteries that can last for about 45 minutes. For experiments that may require much longer time, the robot can be connected to a host computer by a lightweight cable to provide it with the needed electrical power. This is an important feature that allowed long control experiments (such as developing evolutionary algorithms) to be carried out without repetitive recharging.

On the other hand, for the control autonomy, the robot's CPU board is equipped with MC68331 microcontroller with 512K bytes of ROM (system memory) and 256K bytes of RAM (user memory). This RAM memory can accommodate reasonable length program codes to provide control autonomy. The robot can be programmed using Cross-C compiler and the program will be uploaded to the robot through serial port communication with a host computer. Also the robot can be remotely controlled by a host computer where the control commands are sent to the robot through the serial link connection mentioned above. This mode of operation has an advantage of using computational power of the host computer.

## 4.2 Evolutionary Computations and Robotics

The term Evolutionary Computation is used to describe a set of algorithms that use the idea of evolution in solving complex computational problems such as our problem of designing a robot controller. It includes algorithms such as Genetic Algorithms GA, Genetic Programming GP and Evolutionary Strategies. They operate on a population or a group of individuals each representing a proposed solution of the problem. Then they apply a set of biologically in-

spired operators such as mutation and crossover to obtain a better generation which is more suited to the problem to be solved.

So what can Evolutionary Computation offer to robotics? First thing it offers to robotics is an optimization tool. Optimization is a frequent type of problems solved by Genetic Algorithms due to the embedded competition between individuals. In applying the Genetic Algorithm for optimization, the individuals are usually points in the space to be searched for optimum point and the fitness is the function to be optimized. The reproduction aims at generating new points from existing ones until the optimum point is found. Genetic Algorithm offers useful properties for the optimization problem:

- It is applicable to continuous, discrete and mixed optimization problems and it requires no information about the continuity or the differentiability of the function to be optimized. It also can be used for problems of optimization with constraints. The constraints on the parameters to be optimized can be easily translated to constraints on the genetic operators to produce individuals inside the search domain defined by the constraints.
- Genetic Algorithms are suitable for many practical problems that require multi-objective functions. Multi-objective optimization can be accomplished by designing fitness function that is a weighted sum of required objectives. Another solution is using Co-evolution where multiple populations are used instead of single population. Each population is bred to optimize certain objective while individuals are exchanged between them (migration).

For example, one of the possible methods for evolving a neural network controller is to let the evolutionary algorithm choose the optimal weights of the neural network, so the problem of evolving this controller to perform obstacle avoidance behavior can be viewed as a problem of optimizing the different weights. Also this problem is multi-objective optimization because we want the neural network to achieve different goals such as avoiding the obstacles while keeping a reasonable velocity and keeping a straight path.

Second thing evolutionary computations can offer to robotics is providing a method of learning rules necessary for the robot to achieve some task. In this case the controller is mainly a set of rules and we want to choose the optimal set of rules that serve this task. Programming the rules by hand or testing different combinations of them is a tedious task. An example of using the genetic algorithm to learn robots rules is a system built at the Naval Research Laboratories and is called SAMUEL [3]. It used the method described above to learn Nomad robot navigation and obstacle avoidance. Rules are not the only form of controllers that can be designed by evolutionary computations [4] [5]. In the next sections see how evolutionary computations can be used to design controllers such as neural networks and fuzzy logic controllers.

## 4.3 Evolving Neural Network Controllers

Many researchers have found neural network and interesting solution for the problem of the building behaviors for the Khepera robot. The ability to learn and the ability to deal with noisy sensors were apparent advantages in favor of the neural network.

Different approaches exist for designing neural network controllers. One approach is to use neural networks learning algorithms to train the synaptic weights. Example of this work can be found in [6]. Another Approach is to use the Genetic Algorithm as a search or optimization tool to find the best neural network controller through the evolution process. The leading work of evolving neural network controller for a real Khepera robot was done by Floreano and Mondada [7]. They evolved a simple feed forward neural network that consisted of input and output layers with no hidden layers. The neural network controller enabled the robot to navigate in the arena while avoiding obstacles .

We can use the genetic algorithm in different ways to evolve neural networks. It can be used to search for the optimal synaptic weights, or to search for the optimal network architecture along with the synaptic weights. Also, it can be used to evolve the learning parameters needed to train the neural network. Examples of these methods are presented in the following subsections. For example, using the genetic algorithm to search for the suitable synaptic weights given a predefined architecture is presented in experiments 1 and 3 whose goals are to evolve obstacle avoidance and home seeking behaviors respectively. On the other hand, evolving the network architecture is the method used in experiment 2 to develop a light seeking behavior. Finally, evolving Hebbian learning rules and the rate of learning is an example of evolving the learning parameters of the neural network and it is one of the methods used in experiment 5 to co-evolve predator-prey behavior in two robots.

### 4.3.1 Experiment 1: Evolving Obstacle Avoidance Behavior

The goal of this experiment [8] is to evolve a neural network controller for obstacle avoidance navigation in environments with static or dynamic obstacles. The proposed neural network is a feed forward neural network with input layer consisting of 8 neurons, hidden layers of 2 neurons and output layer of 2 other neurons. The inputs of the neural network are the eight proximity sensors that are arranged on the robot as shown in figure (4.2). The input range of each sensor is [0, 1023]. The values of the inputs were scaled to the range [0, 1] before being applied to the neural network. The Outputs of the neural network controller are applied to the motors of left and right wheels. The activation function of the neurons is the sigmoid function which is limited between [-1, 1], so the output of the neural network had to be properly scaled before being applied to the motors.

The fitness function used rewarded the individual which moves with a suitable forward speed and penalize the individual which rotates around itself or comes close to the obstacle. It has the following formula:

$$\text{fitness} = C_1(V_L + V_R - |V_L - V_R|) - C_2 \sum_{i=1}^{8} S_i \tag{4.1}$$

where $V_L, V_R$ are the velocities of left motor, right motor respectively, $S_i$ is the proximity sensor number $i$, and $C_1, C_2$ are suitable positive scaling factors. The term $V_L + V_R$ will maximize the forward speed while term $|V_L - V_R|$ will minimize the rotation of the robot which occurs due the difference between the velocities of left and right wheels. Also, the robot will learn to keep a suitable distance separating it from the obstacles in order to decrease the magnitude of the sum of the sensors. The constants $C_1$, $C_2$ set the relative importance of each component of the fitness function, for example increasing $C_2$ will emphasize the importance of avoiding obstacles relative to keeping a straight path.

The fitness of the individuals is evaluated as follows: each individual was allowed to perform a 400 time step, in each step it reads the proximity sensors, calculate the output speeds using its own neural network and apply these speeds to the motors then it measures the new proximity sensor values and calculate its fitness function according to the above formula. Individual fitness is the sum of its fitness function over the 400 time steps. The above algorithm lasted for 120 generations.

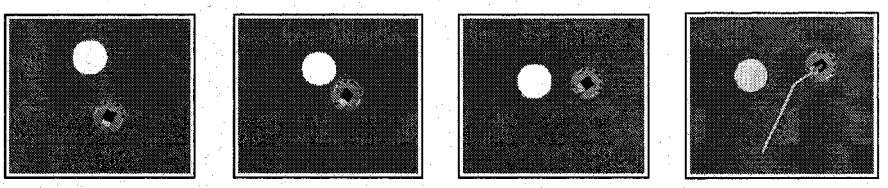

**Fig. 4.3.** Trajectory of the robot in an environment with moving obstacle.

The result of the experiment showed successful emergence of the desired behavior. After 80 generations, the robot was able to move in straight trajectories and it learned to keep a suitable distance between its path and the obstacles or walls. This is clear in the left section of figure (4.4) which shows the behavior of the robot in an environment with large centered obstacle. While moving parallel to the wall, the robot moves in a straight path and maintains certain distance between its path and the wall. Fig. (4.3) shows the behavior of the best fit individual when a round object of the same size of the robot is approaching its path. The slides taken from the motion of the robot shows its turning and avoiding collision with the moving object. Fig. (4.4) shows the behavior of the robot in an environment with obstacles having

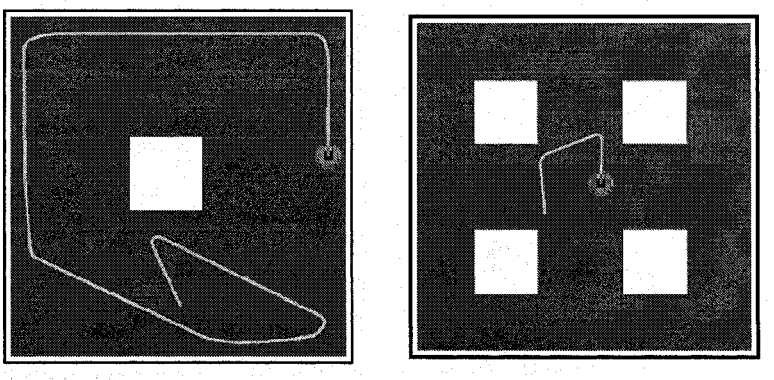

**Fig. 4.4.** Trajectories of the robot in environments with large obstacles with sharp corners .

sharp corners which is difficult to detect if the robot is heading towards the corner. We can see that the robot turns before being close to the corner and this behavior is repeated twice. It should be also noted in this environment that distance between the two obstacles is about twice the diameter of the robot.

### 4.3.2 Experiment 2: Evolving Light Seeking Behavior

This experiment was performed by Hülse et al. [9]. The goal of the experiment is to evolve a neural network controller that enables the robot to seek the light source available in its arena. The proposed neural network had 16 input neurons and 2 output neurons. The input neurons corresponds to the 8 proximity sensors and the 8 ambient light sensors while the two output neurons correspond to the two motor speeds.

The evolutionary algorithm used in this experiment allowed the evolution of the structure of the neural network along with the synaptic weights values. It can evolve the number of the hidden neurons necessary to connect the input and output layers along with their recurrent connections.

The evolution experiment was carried in a simulated environment while the best fit individual was tested in both real and simulated environments. The results of the experiments showed the emergence of light seeking behavior in the early generations. The best fit individual was tested in two simulated environments and in a physical environment. The first simulated environment contained one light source. The robot was able to move towards the light source from different starting positions. The second simulated environment contained more than one light source. The robot moved towards the nearest light source. The best fit controller was then moved to a real robot and tested in a physical environment. In similar conditions to the simulated environment, the robot was able to move to the light source. The environment was slightly modified to test the controller ability to adapt to changes in the physical

environment. When the light source was moved the robot was still able to move towards and follow the light source, which shows consistency with the behavior in the simulated environment. Next, the light source was removed from the environment, and then the robot started to move in curved or semi circular trajectories compared to straight trajectories in the presence of the light source. To test how the behavior is affected by the proximity sensors, the proximity sensors were removed, in this case the robot was still able to move to the light source when it existed in the environment, however in its absence, the robot rotated around its axis. These results show good match between the behavior in simulated and real environments, they also showed that the evolved behavior was invariant when the light source was moved but was affected by removing the connections from the proximity sensors when there was no light source in the environment [9].

We notice in this experiment that the genetic algorithm allowed the evolution of the network architecture along with the best synaptic weights . This method enables the genetic algorithm to search for the best neural network controller in the space of the network architectures. In general, this method would lead to better quality solution than the case of predefined network architecture. On the other hand, this method requires a variable length chromosome that encodes the neural network. Also the chromosome is expected to be longer than the one that encodes only the synaptic weight which would result in longer evolution time.

### 4.3.3 Experiment 3: Evolving Recharging and Home Seeking Behavior

This experiment was performed by Floreano and Mondada [10]. Although the experiment evolved an interesting home seeking behavior, the actual goal of the experiment was to show that behaviors can be evolved without being explicitly included in the fitness function. In this experiment the fitness function didn't include a pleasure part to reward the robots when returning to home (or the recharging area). However, without recharging, the robot will not be able to live longer and achieve a high fitness which was allowed to be calculated over a period longer than the battery life time.

The experiment was conducted in a rectangular environment where one of the corners was illuminated with a tower carrying a number of lamps. This corner was considered the robot's home or recharging area. In this corner, a circular sector of the ground is painted in black such that the robot can detect it using an extra ambient light sensor placed under the robot. This sensor is active in the entire environment except the recharging area.

Using the robot actual battery which lasts for 40-45 minutes will cause the experiment to last for a very long time. Instead, the robot was equipped with a simulated battery that discharges linearly with time in a maximum of 20 seconds. The reading of the battery time can be considered a virtual battery sensor whose value falls between [0, 1], with 1 indicating that the

battery is fully charged. For the robot to detect the light source associated with its recharging area, two sensors acted as ambient light sensors beside their function as proximity sensors. The two sensors are the ones labeled 2 and 6 in figure (4.2).

The neural network controller used was 3 layers neural network with recurrent connections in the hidden layer The input layer has 8 neurons for proximity sensors, 2 neurons for ambient light sensors and 2 other neurons for floor brightness and simulated battery sensor. The output layer consisted of 2 neurons that correspond to the motor speeds.

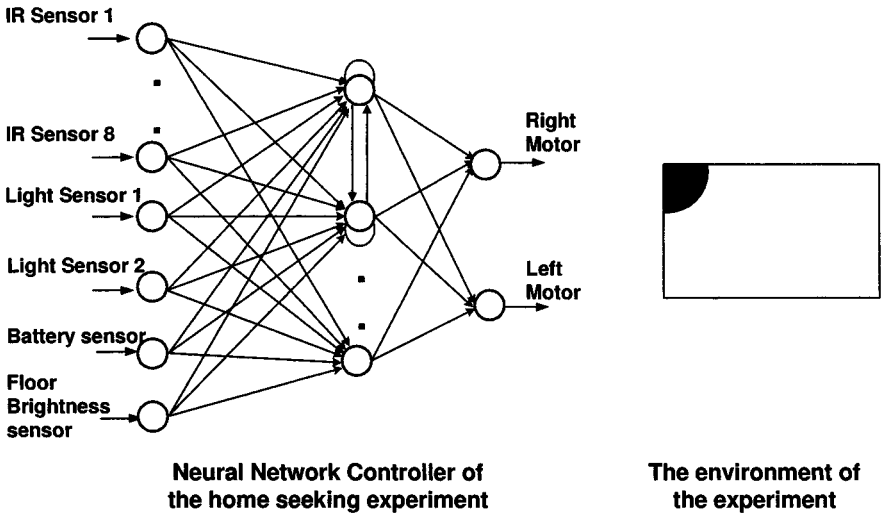


**Fig. 4.5.** The neural network controller of the home seeking experiment (left). A figure of the environment(right).

The fitness function used in the experiment rewarded the individuals that move with large speed and avoid the walls. The fitness function formula is given by [10]:

$$\text{fitness} = v(1 - i) \tag{4.2}$$

where $v$ is normalized average speed of the two motors $0 \leq v \leq 1$, and $i$ is normalized value of the maximum proximity sensor $0 \leq i \leq 1$. The fitness function is calculated and summed over maximum number of 150 time step while the battery life lasts for 20 seconds or 50 time steps. Also the fitness function is not summed when the robot is in the recharging area. The robot should learn to return to the recharging area before its battery life comes to an end. Furthermore, it should not stay there for long since no fitness is gained there. This behavior is not stated explicitly in the fitness function but implicitly implied by the conditions of the experiment.

The genetic algorithm lasted for 240 generations. The results of the experiment showed that in the last generations the behavior of the robot was as expected. It returned to home for recharging without spending much time there after recharging. The behavior of the best fit individuals was as follows: When it was placed in the charging area, it quickly moved away and returned only before the the battery life ends by 5 time steps. Outside the recharging area, it moved with maximum speed avoiding the walls whenever they are encountered. Testing the best fit individuals from different initial positions showed that it was able to return for recharging for many times for most of the initial positions.

Also the results of the experiments showed that we can find a direct relation between the activation level of one of hidden neurons and certain behaviors. Observing the activation level of this hidden node over the robot life showed that it had a low activation level when the robot navigated outside the recharging area but gradually increased during the journey to the back for charging in the last period of the battery life. The activation level reached its maximum when the robot is in the charging area. This fact supports the assertion that this hidden neuron played a role in the behavior responsible for planning the journey back to home before the battery life ends [10].

### 4.3.4 Experiment 4: Evolving Trash Collection Behavior

This experiment was performed by Nolfi [11]. The goal of the experiment is to teach the Khepera robot how to clear the arena from trash objects by grasping and placing them near the walls of the arena. This complex task requires skills such as recognizing the trash object and the walls, grasping and releasing the object, and obstacle avoidance. To accomplish this task the Khepera robot is provided with a gripper module that is added on the top of the robot (see figure 4.6). The gripper can perform two main actions: picking and releasing the object. The robot can detect the presence of an object in the gripper by using a light barrier sensor placed in the gripper.

One approach to teach the robot this complex task is to split it into a set of simpler tasks or behaviors and design a module that control each behavior then designing a coordination method that decides which of these modules will take control of the robot based on the current situation. Each behavior could be designed by hand, evolved or learned by other learning methods. An example of this approach is found in [12] where all the modules are programmed by hand except the grasping behavior which was learned using reinforcement learning. However, in the experiment that we will present the goal was to evolve the entire behavior and to test the hypothesis that different modules of the evolved neural network correspond to certain basic behaviors.

The experiment evolved five different neural network architectures among them two with modular structure. All the architectures had 7 input neurons and 4 output neurons. The input neurons correspond to the 6 proximity sensors on the front side of the robot and the barrier light sensor present in the

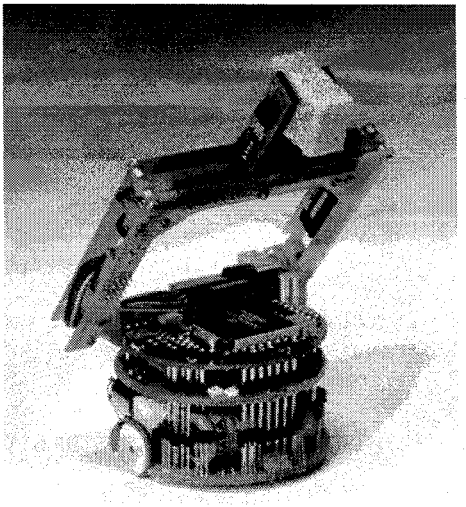

**Fig. 4.6.** Khepera robot with the additional gripper module (with permission of K-team).

gripper. The output neurons are the 2 motor speeds and the 2 actions of the gripper. The five neural network architectures had the following structures:

1. The first neural network is a feed forward neural network with no hidden layer.
2. The second neural network is also a feed forward neural network but with a hidden layer of 4 neurons.
3. The third neural network has recurrent connections between two extra input and output nodes.
4. The fourth neural network has a modular structure. It has two modules each with its own set of the four output neurons. Each module takes control in different predefined situations. The first module takes control when the robot is looking for the trash object and grasping it. Its goal is recognizing the trash object. The second module takes control when the robot is holding the trash object and heading towards the wall. Its goal is recognizing the wall and avoiding obstacles while holding the trash object.
5. The fifth neural network has modular structure too. It consists of two modules. Each module has its own four output neurons in addition to four selector neurons. The selector neurons compete with each other to decide which module will take the control. For example, if at a certain time the activation level of the selector neuron of the left motor is higher in the first module, then output of neuron corresponding to the left motor in the first module will be sent to the left motor.

The environment used in the evolution process was an arena with walls of height 3 cm and it contained 5 trash objects which are cylindrical in shape.

The genetic algorithm used population of 100 individuals for each of the five architectures and it lasted for 1000 generations. The fitness function essentially rewarded individuals for the number of the trash objects successfully placed outside the arena with less rewards for objects that the robot was only successful to pick. Each individual was tested for 15 epochs and its fitness valuation was the sum of its fitness function in each epoch.

The experiment described above was repeated 10 times for every architecture. The 10 best individuals of each architecture were given the same task of clearing the arena from 5 trash objects. The results showed that the fifth neural network excelled the others where 7 of its best 10 individuals were able to successfully complete the task. Only one or two individuals were able to complete the task for the other architectures.

Considering the hypothesis that modular architecture may contain modules that correspond to certain behavior, it was found that the best individual of the fifth architecture use both modules for controlling the left motor and uses only one module for rest of the outputs. This fact showed that relation between modules and basic behaviors could not be proven in this experiment [11]. However, in the experiment of home seeking and battery recharging certain hidden neuron was shown to be responsible for detecting low battery and returning home for recharging.

### 4.3.5 Experiment 5: Co-evolving Predator-Prey Behavior

By co-evolution we mean evolving two competing populations simultaneously such that the fitness evaluation of one is at the expense of the other. The co-evolution adds more competition stress to the evolution process which is, by nature, characterized by the competition for survival among individuals of the same generation. We are now going to present an interesting experiment in co-evolution whose goal was evolving a predatory-prey behavior in two khepera robots. The predator robot is required to chase the prey robot and contact it.

The experiment was performed by Floreano and Nolfi [13] [14]. In the experiment, the predator robot is equipped with a vision module ( see figure 4.7 ) to recognize the prey robot which was provided with a black perturbation that can be easily detected on the white walls of the environment. To provide fair competition, the maximum speed of the prey robot is allowed to be twice that of the predator robot.

The environment was a square one of dimension 47 cm. That size was chosen such that prey will always be within the detection range of the vision module of the predator which can detect objects in range of 5 to 50 cm. The evolution experiment was carried in a simulated environment of the same details of the actual one. This will help to decrease the time of the evolution and to avoid the hardware problems resulting from the twisting of the power cables of the two robots.

The K213 vision module of the khepera robot is an additional module that is connected to the top of the robot. It is cable of providing a linear image of 64 pixels that cover a vision angle of 36 degrees. Furthermore, the module has a microcontroller that can process the image data and instead of sending the 64 bytes of the image to the robot it can detect the least eight pixels in intensity and pass them to the robot.

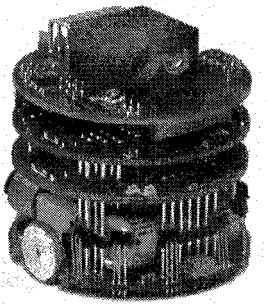

**Fig. 4.7.** Khepera robot with the extra K213 vision module (with permission of K-team).

In the simulated computer environment, the experiment designers divided the vision range to 5 sections each representing a simulated photosensor. These simulated photosensors act as input for the neural network controller of the predator robot. A simulated photosensor is considered active if a pixel of minimal intensity is within its range, possibly because of the presence of the prey robot in this section.

The controllers of the two robots are shown in figure (4.8). Each controller is recurrent neural network. The predator neural network has extra 5 input neurons corresponding to the five photosensors. On the other hand, the two outputs of the prey neural network are multiplied by a factor of two before being applied to the motors of the robot.

The genetic algorithm used two competitive populations each of 100 individuals and the experiment lasted for 100 generations. As we mentioned earlier, the fitness evaluation of each robot is at the expense of the other. The predator robot is awarded for decreasing the time needed to contact the prey. Its fitness is a normalized version of that time and falls in the range of [0, 1]. The prey robot fitness function is just (1 − predator fitness). The fitness function of each individual, predator or prey, is evaluated through testing it against the best individuals of the last 10 generations of the opposite type.

The experiment used direct encoding to encode the synaptic weights of the neural network. Each weight is encoded in 5 bits. The first bit is always used to encode the sign while the other four bits differed according to the instance of the experiment. We will to summarize each of the three instances of the experiments along with its results [13] [14].

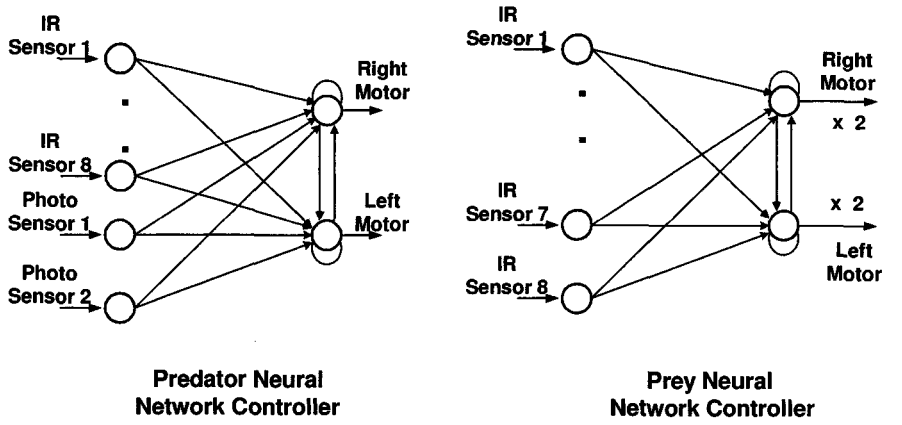


**Fig. 4.8.** The neural network controller of the predator and prey robots.

- First instance of the experiment: In this instance the four bits simply encoded the value of the synaptic weight which falls in the range of [0, 1]. The results of this instance of the experiment showed that there was no population superior to the other all the time span of the evolution. In the first generations the predator was able to chase the prey and contact it. After 70 generations, the prey was cable of turning away when the predator approached it. After 90 generations, the predator learned better attacking methods for chasing the prey.
- Second instance of the experiment: Only two bits were used to encode the value of the weight and the other two bits are used to encode four different level of uniform noise that would be added to the weights. The results of this instance of the experiment showed that the noise level in the synaptic weights of the prey was higher than those of the predator which suggested that the prey made use of this noise to evolve an unpredictable and changing trajectory to confuse the predator robot.
- Third instance of the experiment: The four bits are used to encode the learning parameters of the synaptic weights rather than the value of the weights. Two bits encoded the Hebbian rules and the other two bits encoded the learning rate. The value of the weight is randomly generated between [0, 1] and continuously updated according to the rules. The results of this instance of the experiment showed that the average fitness of the predator is higher than that of the prey. In terms of the apparent behavior, it developed better chasing techniques than that of the first instance of the experiment. In terms of the synaptic weights, the experiment results showed that the synaptic weights were adjusted by the Hebbian learning and the resulting motor speed steered the robot towards the prey, a property which require fine tuning of the weight values if the encoding method of the first instance was used.

The results of this experiment are interesting and reflect how the behavior of the robot was dependent on the types of the parameters of the controller encoded in the gene despite the fact that the controller had the same architecture in the three instances of the experiment. We would expect also that different behaviors could have obtained by allowing the evolution of the architecture of the neural network along with the weights.

## 4.4 Evolving Fuzzy Logic Controllers

Fuzzy Logic is a mathematical tool that can manipulate human vague concepts and linguistic variables. Zadeh in [15] proposed a method to treat human knowledge based on the Theory of Approximate Reasoning. He proposed that systems with ill defined or with uncertain model can be treated by fuzzy logic. These principles were then used to build a controller for the first time in [16].

In this section, we will briefly present how the fuzzy controller can be applied to the problem of mobile robot navigation and obstacle avoidance. The fuzzy controller usually consists of three parts:

### The Fuzzifier

The first step in any fuzzy control application is to specify the fuzzy sets and the corresponding membership functions for each of input or output variables. This process is known as fuzzification. If we apply this to the Khepera input proximity sensor values, we will find that each sensor has a reading value in the range [0,1023]. One of the proposed methods for fuzzification could be: " Near","Medium", and "Far". Also membership function can have other shapes such as the triangular shape or bell shaped. See figure (4.9).

In our example of the Khepera proximity sensor, the reading 300 may have a membership in the fuzzy set "Near" that is equal to 0.75 while the membership in the sets "Medium" and "Far" are equal to 0.25 and 0 respectively. It is clear here that the crisp value 300 has been assigned a membership value for every fuzzy set defined over the range [0, 1023]. Also the output variables (left motor speed and right motor speed) can be fuzzified in the same sense. The fuzzy sets could be "Positive Large", "Positive" , "Zero", "Negative", "Negative Large".

### The Fuzzy Rules

This is the main part of the controller where human knowledge can be represented in the form of if-then rules. The rule usually takes the following form:

*If* (antecedent part) *then* (consequent part)

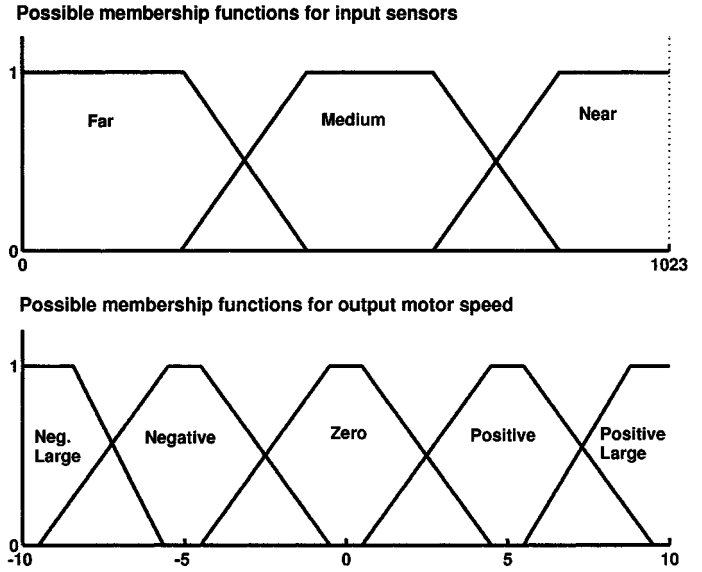


**Fig. 4.9.** Possible membership functions for input sensor and output motor speed.

Where the antecedent part checks the input variables and the consequent part sets one or more of the output variables. For our case of Khepera robot navigation, one of the rules can be:

*If* (left proximity sensor is "Near") *then*
(left speed is "Positive Large" ) and (right speed is "Postive")

This rule tells the robot to turn to right (by moving the left wheel faster than the right wheel) if obstacle is found on the left of the robot. If the left proximity sensor is near with membership value 0.75, then this rule will have firing value equals to 0.75. A group of fuzzy rules resembling the previous one are needed for the safe navigation of the robot.

### The Defuzzifier

The outputs (left and right speeds in our case) need to be crisp values, this will be the role of the defuzzifier to convert them form fuzzy sets to crisp value. This is done through the fusion of different rules based on their firing values.

Since the performance of the fuzzy logic controllers depends on the parameters of the membership functions and the rules used, then we need to search for the best membership functions and the optimal set of rules. This leads us to thinking of genetic algorithm to evolve the best fuzzy logic controller parameters instead of designing it based on the human experience.

### 4.4.1 Experiment 6: Evolving Corridor Following Behavior

This experiment was performed by Lee and Cho [17]. The goal of the experiment was to evolve a fuzzy logic controller that can enable the robot to avoid the obstacles and follow the corridors of the environment. The fuzzy logic controller had 8 inputs corresponding to the 8 proximity sensors of the robots and 2 output neurons that correspond to the motor speeds. The role of the genetic algorithm in designing the controller was to evolve the best membership functions of the inputs and the outputs along with the necessary rules.

The experiment designers chose to divide the input sensory range [0, 1023] into four triangular membership functions. The same number and type of the membership functions were used for the outputs. The parameters of these functions, such as their starting and ending point on the input or output range, were binary encoded in the chromosome . Also the chromosome included information about a set of 10 possible rules.

To encourage the robot to explore the arena and follow the corridors without colliding with their walls, the fitness function had a positive part that is function of the total distance moved and the number of the check points in the arena that the robot passed through. It also has negative part that is function of the number of collisions.

The results of the experiment showed that the best fit individual was able to develop basic behaviors of avoiding collision and following walls. The performance of this evolved fuzzy logic controller was tested in two other simulated environments in which it was observed that the robot developed three distinct sub-behaviors which are: passing corridors, wall following and obstacle avoiding. The corridor passing behavior is active when the robot is moving in a narrow path with obstacles on both sides. The wall following behavior become active when the obstacles or walls are sensed on one side of the robot while the obstacle avoidance behavior become active when obstacles are sensed in front of the robot. A relation could be found between each sub behavior and a subset of the fuzzy rules that support this sub behavior. The robot switched from one sub behavior to the other depending on the current situation till its target was reached [17].

## 4.5 Evolving Controlling Programs

Genetic programming GP applies the evolution model to computer programs. The individuals here are computer programs that represent potential solution to required problem. Usually these problems are too complex or time consuming to be programmed by hand. An example of this type of problems is writing a program to control a mobile robot to navigate and avoid obstacles in a new environment.

Now the question that may arise is how to represent computer programs as individuals and how to design genetic operators, such as crossover and mutation, that is applicable to computer programs. Answers of these questions are in Koza's suggestion [18] of representing programs as trees that is composed of nodes and branches. The nodes are the operators that can take any value from certain function set such as {multiplication, addition..}. The branches are the operands which can be constants, input values or results of another node. Fig. (4.10) shows an example of a tree that represents a simple program.

This tree representation provided a method for performing crossover between two individuals. This is preformed by exchanging parts of the two trees representing the two individuals. To perform mutation operator we need to make sure that the resulting individuals represents a valid computer program. For example the mutation operator can take place by changing the operator in the node by another operator from the function set or by mutating the constants in the operands.

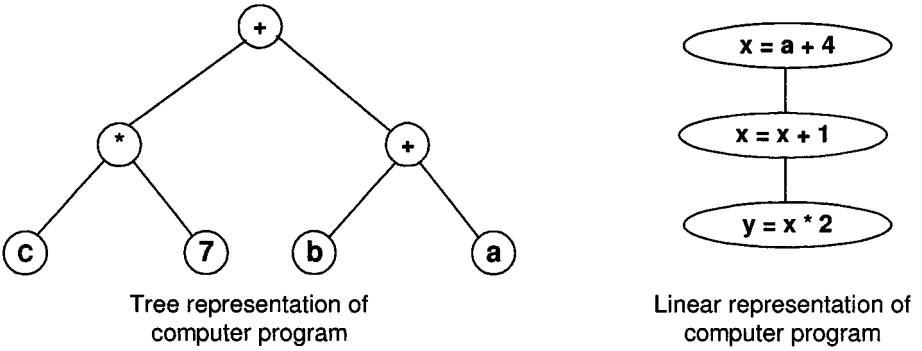


**Fig. 4.10.** Tree representation of computer programs versus linear representation

Having this brief overview of the Genetic Programming GP, we are now ready to present the following experiment in evolving obstacle avoidance controller program using Genetic Programming.

### 4.5.1 Experiment 7: Evolving Obstacle Avoidance behavior using Genetic Programming

This experiment was performed by Nordin and Banzhaf [19]. The goal of the experiment was to evolve a controller program for obstacle avoidance navigation using genetic programming. The experiment was carried on a real khepera robot in two different environments. The first environment was a rectangular arena of size 30 x 40 cm with regular walls while the other is larger in size with obstacles in its center and characterized by irregular walls. In both cases, the khepera robot was controlled by a computer workstation through a serial cable.

Motivated by applying genetic programming on real robots and obtaining a reasonable behavior in a short time, the experiment designers made two choices. First choice was not to use the tree structure we discussed above. Instead, the individual programs were represented as a linear sequence of operations along with their operands.An example of this representation is showed in figure (4.10). Second choice was to represent these instructions in the low level binary format of the controlling workstation (Sun 4). Using this representation, the crossover operators will be carried by exchanging two segments of instructions between two individual programs. The mutation operator was restricted to produce only valid machine instructions.

The population size of the experiment was small and consisted of 50 individuals and tournament selection is used when individuals are needed to be selected for crossover or mutation. The tournament works as follows: First we select $n$ individuals from the population size $N$ and each of the $n$ individuals is tested and its fitness is evaluated, then we choose the best fit individual out of them for crossover and mutation.

The results of the experiment showed successful evolution of the obstacle avoidance behavior in both of the environments. In the first environment, it took the robot 20 minutes to evolve a reasonable obstacle avoidance behavior. In the second environment, it took the robot some longer time compared to learn the same behavior. This may be because of the complexity of the second environment [19].

The results of this experiment showed how the choice of some parameters of the genetic algorithm such as the encoding and selection methods, in addition to the machine format of the programs, helped in evolving the required behavior in small amount of time. We could see that a reasonable behavior emerged in less than an hour in both environments.

## 4.6 Evolving Spiking Neural Network Controllers

In this section, we are going to introduce a new model of the biological neurons that models the dynamical nature of neurons communication. This new model is what we call spiking neurons. We will also present an evolution experiment that evolved spiking neural network for controlling a robot based on vision information only.

To explain the spiking neuron model, we will need first to have a look at the actual way of communication between biological neurons. Biological neurons communicate by sending a large number of short pulses each second. These short pulses are known as spikes. The classical model of neurons considers only the rate of these spikes. The current activation level in the classical model corresponds to the current rate of spikes normalized by its maximum value. On the other hand, the spiking neuron provides more complex model of neuron activation function that depends on the timing between spikes.

One widely used model of spiking neuron is the "Integrate and Fire" model. In this model, the activation of the neuron is described by its membrane potential. Each spike received contributes to the membrane potential according to two factors: the weight of its synaptic connection and the time elapsed since its firing. When the accumulated effect of these spikes cause the membrane potential to go above certain threshold, the neuron fires a spike. After firing the spike, the neuron becomes unable to fire another spike instantaneously. It needs a refractory period $\eta$ before it sends another spike. This refractory time depends on a certain time constant $\tau_m$ of the membrane.

At any time t, the effect of a spike on the neuron potential is a function of the time difference between the current time t and the firing time of the spike $t_{firing}$. This function $\epsilon(t - t_{firing})$ can be modeled by a pulse shaped function as shown in figure (4.11). In the figure, the period $\Delta$ of zero effect corresponds to the time required by spike to reach the neuron. One of the suggested expressions for $\epsilon(t - t_{firing})$ is given by [20], [21]:

$$\epsilon(s) = \begin{cases} \exp(-\frac{s-\Delta}{\tau_m})(1 - \exp(-\frac{s-\Delta}{\tau_s})) & s \geq \Delta \\ 0 & s < \Delta \end{cases} \tag{4.3}$$

where $s = t - t_{firing}$ represents the time elapsed since the firing of the spike, $\tau_s$ is the synapse time constant. Also we can model the refractory period $\eta(s)$ by a negative decaying exponential where the potential of the neuron is set after emitting the spike to a very low negative voltage to prevent emitting another spike immediately. One of the suggested expressions is given by [20], [21]:

$$\eta(s) = -\exp(-\frac{s}{\tau_m}) \tag{4.4}$$

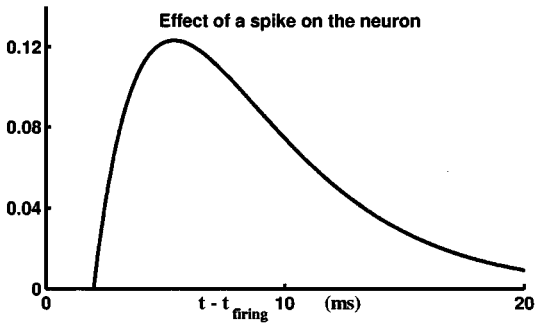


**Fig. 4.11.** The effect of a spike on the neuron $\epsilon(s)$

Now, we can write the the mathematical model of the spiking neuron the gives the potential of neuron $i$ as result of addition of to quantities. The first is due to the effect of received spikes and can be written as the sum

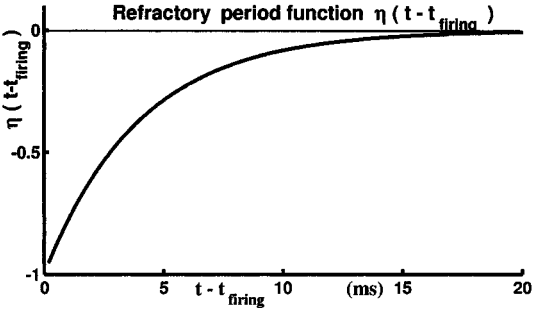


**Fig. 4.12.** Refractory period function $\eta(s)$

of the incoming spikes $\epsilon_j(s_j)$ from other neurons, labeled by index $j$, with each spike effect multiplied by the weight of its synaptic connection $w_j^t$. The second quantity is due to the spikes emitted by neuron $i$ itself and can be written as sum of all refractory functions resulting form the emitted spikes. A mathematical formula of what we have just described can be given by [22]:

$$v_i(t) = \sum_j w_j^t \sum_{All\ recieved\ spikes} \epsilon_j(s_j) + \sum_{All\ emitted\ spikes} (s_i) \qquad (4.5)$$

The above equation describes the model of the activation of the neuron, represented by its membrane voltage, which takes into the consideration the timing of the emitted and received spikes in contributing to the membrane potential. A question might arise here asking why we would be interested in more complex model for neural network to employ is robot controllers. The answer is that model should be better at detecting the time varying relation between the sensors and motors due to its dynamic nature [22]. In the rest of this section, we will see how to employ that new model in controlling Khepera robot and mapping the vision information into motor speeds to develop obstacle avoidance navigation that depends only on the vision information.

### 4.6.1 Experiment 8: Evolving Vision Based Navigation

This experiment was performed by Floreano and Mattiussi [22]. In the experiment, the robot was placed in a rectangular arena whose walls are covered with vertical white and black strips with variable width. The Khepera robot is provided with K213 vision module similar to the one described in the co-evolution experiment in section 3.5. The goal of the evolved controller is to use the information available from the vision module to enable the robot to navigate without colliding with the walls.

The vision module provides a linear image consisting of 64 pixels that cover an angle of 36 degrees. Only 16 equally spaced photoreceptors are used as inputs to the spiking neural network. The values of photoreceptors readings are filtered to obtain information about the contrast, scaled to the range of

[0, 1] and then sent to the spiking neural network. There are extra 2 input neuron in the network whose input is the difference between the actual and the desired motor speeds. Again this difference is scaled to the rage of [0, 1] before being sent to the spiking neural network. The network contained four output neurons, two for each motor speed. The two neurons set the forward and backward speed for each motor. The actual speed sent to the motor is their algebraic sum. In addition to the 18 input neurons and the 4 output neurons the network contained 10 neurons that are connected to the input and output neurons.

The input vision photoreceptors and the output motor speed are interfaced to the spiking neural network as follows. The 16 scaled inputs of photoreceptors are used to set the probability to emit a spike by the corresponding input neurons. Also, the firing rates of the 4 output neurons are mapped to the motor speeds. This explains the reason of using two neurons for each motor speed since that firing rate of the output neurons can not take negative values. The cycle of reading the photoreceptors and updating the motor speed goes in the following order. Every 100 ms, the input photoreceptors are read, filtered, scaled and used to set the probability of emitting a spike by the input neurons. During the 100 ms cycle, the activation level of each neuron, except input neurons, is updated every 1 ms according to the model of equation (4.5) and the neurons are allowed to emit spikes if their activation level exceeds the threshold. At the end of the 100 ms cycle, the spiking rate of the output neurons, calculated over the last 20 ms period of the cycle, is used to update the motor speeds.

The genetic algorithm is used to obtain the best synaptic weights connecting the spiking neurons. The population consisted of 60 individuals and the experiment lasted for 30 generations. Each individual is tested in 400 cycle, in which its fitness is the sum of its motor speeds if they are both positive and zero otherwise. This fitness function will reward the individuals that move forward while offering no reward to individuals that rotate (due to difference in the sign of the speeds) or move backward (when both speeds are negative). The fitness evaluation of the individual is the average of its fitness over the 400 cycles.

The results of the experiment showed that the best individual was able to move in curved trajectories of large radii but without colliding with the surrounding walls. The experiment was repeated using a classical neural network with sigmoid activation function and with same architecture. However, the fitness of its individuals didn't increase with time and its individual neural network controllers were not able to map the vision information into motor speed that secure a safe navigation without colliding with the surrounding walls [22].

## 4.7 Comment on different approaches of evolutionary robotics

We presented different approaches for evolving controllers such as neural networks, fuzzy logic and spiking neural networks. Each approach has appealing advantages as one form of controller for mobile and autonomous robots. It may also include some difficulties or limitations when being evolved. We try in this section to shed some light on the attractive features of these different approaches and some issues that need to be considered when combined with evolutionary computations.

As a general approach, fuzzy logic provides a tool for dealing with systems with uncertain models which suits the dynamic and possibly unknown environments encountered by mobile robots. It has the advantages of implementing human knowledge. It simulates the human method of reasoning by using linguistic variables and knowledge that is represented by its rule base. For example, the human experience in walking or navigation while avoiding possible obstacles can be moved to the robot brain through using a fuzzy controller whose rules are based on this experience.

Another useful feature of fuzzy logic that is interesting in the field of robotics is its ability to combine different rules outputs in the defuzzification process. This ability can be further used in behavior coordination. In this approach different controllers are designed independently, possibly by fuzzy logic, neural networks or even designed by human programmers. Every controller implements a certain behavior or task. A simple example is two controllers for obstacle avoidance and goal seeking. Our problem in behavior coordination is to combine results from different behaviors in one command to send to the effectors or motors. The fuzzy approach for this problem works by providing a number of rules that assigns weights for fusing the different outputs from the controllers based on the current situation. In our example, a typical rule will favor the output of obstacle avoidance behavior when a near obstacle is detected. This method provides a way of combining the outputs of many behaviors each control cycle unlike behavior arbitration methods that choose one active behavior each time based on fixed or dynamic priorities. As we mentioned, these rules can be based on human experience. Further more, genetic algorithm can be employed to evolve the best set of rules for behavior coordination. In fact, this was the approach used by Tunstel et al. in [23] to evolve fuzzy behavior arbitration for planetary microrovers.

On the other hand, fuzzy logic approach lacks a standard method for creating the rules based on the human experience. Also the time taken in computations especially in the defuzzification process may affect the real time performance of the controller and the the robot if not performed using dedicated processors [24]. Another issue that needs to be considered when designing fuzzy logic controller for a robot is the design of the membership functions. In some experiments, redesigning the membership functions led to avoiding oscillations in the robot behavior [25].

Evolutionary computation appears to be a good solution to the problem of automatic design of the fuzzy logic controller. However there are some issues that the controller designer should consider when evolving the fuzzy logic controller. One of these issues is deciding what to evolve, whether it is the membership function parameters, the rules or both of them. Evolving both rules and membership functions has the advantage of decreasing chances of errors due to miss choices made in the early stages of the design, however the evolution process will search in a larger space for the best set of rules and best parameters for the membership functions. It should be noted that even by evolving the rules and the membership parameters, this can not eliminate the designer choice of the type of membership function (triangular or trapezoidal ...etc). Evolving the fuzzy behavior coordination module mentioned earlier is an example of evolving the fuzzy rules while the experiment in section four of this chapter is an example of evolving the fuzzy rules along with the membership functions.

Another issue to be considered in evolving fuzzy logic controller is the number of rules. The number of rules can affect the speed and performance of the robot and the choice of the genetic algorithm as well. Small number of rules will decrease the computations in the fuzzy logic controllers but on the other hand this small number may not cover all the possible situations or sensors combinations encountered by the robot. Evolving controller with fixed number of rules or fixed maximum number of rules will lead to using fixed length chromosome. The other approach of using population of individuals with different number of rules requires variable length chromosomes and possible modification of the genetic operator. Messy genetic algorithm [27] can be a potential evolutionary algorithm for evolving the fuzzy logic controller with variable number of rules. It has a modified version of the traditional crossover genetic operator called cut and slice operator that can deal with the variation of the genetic material length. In fact, it was used by Hoffman and Pfister in [26] to evolve the rules for fuzzy logic controllers to enable a mobile robot to reach its target while avoiding the obstacles.

Another approach of evolutionary robotics that we presented is evolving neural networks. Artificial neural networks offer many characteristics that make them suitable for the problem of controlling autonomous robots. First, the noise present in the sensor readings, whether they are sonar sensors or infrared sensors, makes the neural networks suitable controllers due to their known tolerance to noise. Moreover, if one of the sensors was not functioning, the output of the neural network could still be acceptable [7]. Second, the neural networks are able to learn and they could be trained. The weights and the thresholds and other parameters of the neural network could be adjusted to produce different behaviors even for the same network architecture. Also, neural networks can select the sensors that are suitable for a given behavior by adjusting the weight corresponding to each sensor or input.

As in the case of fuzzy logic, the genetic algorithm can offer an automatic way for designing neural network controller by evolving the synaptic

weights or the network architecture or both of them. Neural networks have many existing learning algorithms, but the genetic algorithms offers potential advantage of the parallel search by using a population of individuals. An issue to be considered in evolving neural networks that may affect the genetic algorithm is the size of the parameter to be evolved. Large networks with large number of synaptic weights may require a long chromosome. In this case the real encoding of these parameters could be considered instead of the binary encoding.

Compared to fuzzy logic, the learning of the neural network which is stored as synaptic weights can not be acquired by human reasoning [24]. For example, in the experiment of trash collection no direct relation was found between modules of the neural network and the certain behavior of the robot, sometimes by observing the activation level of some neurons and certain behaviors of the robot we could find a correlation as in the experiment of home seeking but this is not the general case. On the other hand, the knowledge represented by the rules of the fuzzy logic controller can be acquired by human reasoning. For example, we could read on of the evolved rules in the experiment of evolving fuzzy logic controller and understand what it implies. Another point is that we can not easily implement high level behavior using neural networks as we can do using fuzzy logic. Although many relatively complex behaviors have be evolved using neural networks, such as trash collection, implementing a high level reasoning and selection or coordination between behaviors would require a method that mimics human reasoning.

We have also presented in this chapter a relatively new approach in evolutionary robotics which is evolving behaviors using spiking neural networks. The dynamic model of the spiking neural network suits the time changing relation between the sensors and the motors [22]. On the other hand, the complexity of the model and the need of the interface between the sensors of the robot and input of the spiking neural network have limited the experiments of evolutionary robotics that use it compared to other widely used approaches as artificial neural networks or fuzzy logic. Analog Very Large Scale Integrated Circuits (VLSI) can implement spiking neural networks using circuits with very small area and power consumption, which is an advantage over other approaches. In [28], an analog VLSI circuit that implemented spiking neural networks was used for controlling a robotic leg.

To summarize, each approach of evolutionary robotics is characterized by some potential advantages that makes it a suitable solution for the problem of controlling mobile robots. Also each approach has some limitations or difficulties when being evolved. Choosing which approach is a trade off between the advantages and the limitations.

## 4.8 Summary

In the previous sections we have seen how the evolutionary computations algorithms were successfully used to evolve many types of controllers for Khepera robot. It was used to evolve neural network synaptic weights in the obstacle avoidance behavior of experiment 1 and the battery recharging behavior of experiment 3. We have also seen how it can evolve the architecture of the neural network along with the synaptic weights as in the experiment of evolving light seeking behavior. Alternatively, it can evolve the learning rules and learning rate necessary for training the neural network synaptic weights. Other types of controllers were successfully evolved too, such as fuzzy logic controllers and computer programs.

Many other experiments are conducted using evolutionary computations on different robotic platforms recently. In fact, evolutionary computation is a very promising approach for designing controllers for mobile robots.

**Acknowledgement**

The author would like to thank S. Mercorious and S. Kirolos for their support all over the past years.

## References

1. K-Team, "Khepera User Manual," Lasuanne, Switzerland, 1999.
2. F. Mondada, F. Franz and I. Paolo, "Mobile Robot Miniaturisation: A Tool for Investigation in Control Algorithm," Proceedings of the Third International Symposium on Experimental Robotics, Kyoto, Japan, 1993.
3. A. Schultz and J. Grefenstette, "Using a Genetic Algorithm to Learn Behaviors for Autonomous Vehicles," Naval Research Laboratory, Washington, Dc, 1992.
4. J. Meyer, P. Husbands and I. Harvey, "Evolutionary Robotics: a Survey of Applications and Problems," In Evolutionary Robotics : First European Workshop, Evorobot'98, P. Husbands and J. Meyer (editors), Springer Verlag 1998.
5. I. Harvey, P. Husbands, D. Cliff, A. Thompson, N. Jakobi, "Evolutionary Robotics: the Sussex Approach," In Robotics and Autonomous Systems, Vol. 20, pp. 205-224, 1997.
6. A. Loffler, J. Klahold and U. Ruckert, "The Mini-Robot Khepera as a Foraging Animate: Synthesis and Analysis of Behavior," In Proceedings of the Fifth International Heinz Nixdorf Symposium: Autonomous Minirobots for Research and Edutainment (AMiRE), Vol. 97, pp. 93-130, 2001.
7. D. Floreano and F. Mondada, "Automatic Creation of An Autonomous Agent: Genetic Evolution of a Neural Network Driven Robot," From Animals to Animats:3, Proceedings of the Conference on Simulation of Adaptive Behavior, edited by D. Cliff, P. Husbands and S. Wilson, MIT Press, 1994.
8. M. Botros "Evovlving Neural Network Based Controllers for Autonomous Robots Using Genetic Algorithms," Master Thesis, Cairo University, Egypt, 2003.

9. M. Hulse, B. Lara, F. Pasemann and U. Steinmetz, "Evolving Neural Behaviour Control for Autonomous Robots," Max-Planck Institute for Mathematics in the Sciences, Leipzig, Germany, 2001.
10. D. Floreano and F. Mondada, "Evolution of Homing Navigation in a Real Mobile Robot," IEEE Transactions on Systems, Man, and Cybernetics (B), Vol. 2, pp. 396-407, 1996.
11. S. Nolfi, "Using Emergent Modularity to Develop Control Systems for Mobile Robots," Journal of Adaptive Behavior, Vol. 5, pp. 343-363, 1997.
12. C. Scheier and R. Pfeifer, "Classification as Sensory-Motor Coordination," Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life, edited by F. Moran, A.Moreno, J. Merelo and P. Chacon, Springer Verlag, 1995.
13. D. Floreano and S. Nolfi, "God Save the Red Queen! Competetion in Co-evolutionary Robotics," Genetic Programming 1997: Proceedings of the Second Annual Conference, Stanford University, edited by J. Koza, K. Deb, M. Dorigo, D. Fogel, M. Garzon, H. Iba, and R. Riolo, pp. 398-406, 1997.
14. D. Floreano and S. Nolfi, "Adaptive Behavior in Competing Co-Evolving Species," Fourth European Conference on Artificial Life, MIT press, Cambridge MA, editted by P. Husbands and I. Harvey, pp. 378-387, 1997.
15. Zadeh, L., "Outline of a New Approach to the Analysis of Complex Systems and Decision Process," IEEE Transaction Systems, Man and Cybernetics, Vol. 3, pp 28-40, 1973.
16. E. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with Fuzzy Logic Controller," Journal of Man-Machine Studies, Vol. 7, pp. 1-7, 1975.
17. S. Lee and S. Cho, "Emergent Behaviors of a Fuzzy Sensory-Motor Controller Evolved by Genetic Algorithm," IEEE Transaction Systems Man and Cybernetics (B), Vol. 31, No. 6, pp. 919-929, 2001.
18. J. Koza, "Genetic Programming," MIT Press, Cambridge MA, 1992.
19. P. Nordin and W. Banzhaf, "Genetic Programming Controlling a Miniature Robot," Working Notes for the AAAI Symposium on Genetic Programming, MIT, Cambridge MA, 1995.
20. W. Gerstner and W. Kistler, "Spiking Neuron Models," Cambridge University Press, 2002.
21. W. Gerstner, J. van Hemmen, and J. Cowan, "What Matters in Neuronal Locking?," Neural Computation, Vol. 8, pp. 1653-1676, 1996.
22. D. Floreano and C. Mattiussi, "Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots," Evolutionary Robotics. From Intelligent Robotics to Artificial Life, Springer Verlag, Tokyo, 2001.
23. E. Tunstel, H. Danny, and M. Jamshidi, " Behavior Hierarchy for Autonomous Mobile Robots: Fuzzy-behavior modulation and evolution," International Journal of Intelligent Automation and Soft Computing, Special Issue: Autonomous Control Engineering at NASA ACE Center, Vol. 3, pp. 37-49, 1997.
24. J. Godjevac, "Comparative Study of Fuzzy Control, Neural Network Control and Neuro-Fuzzy Control", In Fuzzy Set Theory and Advanced Mathematical Applications,D. Ruan Ed., Kluwer Academic, Chapter 12, pp. 291-322, 1995.
25. S. Marapane, M. Trivedi, N. Lassiter and M. Holder, "Motion Control of Cooperative Robotic Teams through Visual Observation and Fuzzy Logic Control," Proceedings of IEEE International Conference on Robotics and Automation, Vol. 2, pp. 1738-1743, 1996.

26. F. Hoffmann and G. Pfister, "Evolutionary Design of a Fuzzy Knowledge Base for a Mobile Robot," International Journal of Approximate Reasoning, Vol. 17, pp. 447-469, 1997.
27. D. Goldberg, B. Krob and K. Deb, "Messy Gentic Algorithms Motivations, Analysis and First Results," Complex Systems, Vol. 3, pp. 493-530, 1989.
28. M. A. Lewis, M. Hartmann, R. Etienne-Cummings, and A. Cohen, "Biomorphic Control of a Running Robot Leg using a Custom aVLSI CPG Chip," Neurocomputing, Vol. 38-40, pp. 1409-1421, June 2001.