INFORMATICA, 2004, Vol. 15, No. 1, 63–76
© 2004 *Institute of Mathematics and Informatics, Vilnius*

Evolving in Real Time a Neural Net Controller of Robot-Arm: Track and Evolve

Ahmed LEHIRECHE
Computer Science Department, University of Sidi Bel-Abbes
22000, Algeria
e-mail: elhir@univ-sba.dz

Abdellatif RAHMOUNE
Faculty of Planning and Management
King Faisal University, KSA
e-mail: arahmoun@kfu.edu.sa

Received: December 2002

Abstract. Evolutionary Engineering (EE) is defined to be “the art of using evolutionary algorithms approach such as genetic algorithms to build complex systems”. This paper deals with a neural net based system. It analyses ability of genetically trained neural nets to control Simulated robot arm, witch tries to track a moving object. In difference from classical Approaches neural network learning is performed on line, i.e., in real time. Usually systems are built/evolved, i.e., genetically trained separately of their utilization. That is how it is commonly done. It’s a fact that evolution process is heavy on time; that’s why Real-Time approach is rarely taken into consideration. The results presented in this paper show that such approach (Real-Time EE) is possible. These successful results are essentially due to the “continuity” of the target’s trajectory. In EE terms, we express this by the Neighbourhood Hypothesis (NH) concept.

Key words: evolutionary engineering, genetic programming, genetic algorithm, tracking, real time, neighborhood hypothesis, artificial intelligence.

1. Introduction

Intelligent systems have been extensively investigated in the few last years, and represent a real challenge for software system developers for industrial applications of AI. The fusion of emerging concepts such as neural networks, fuzzy systems, genetic algorithms, expert systems has proven to be a useful way to develop real-world applications, including the areas of control systems, robotics, diagnosis systems, and industrial operations. Evolutionary Engineering (EE) is a discipline of soft computing engineering. EE is defined (De Garis, 1993b) to be “the art of using evolutionary algorithms approach such as genetic algorithms (Goldberg, 1989) to build complex systems”. Essentially this discipline aims to solve the problem of building complex systems without going through any design process. By imitating nature, the evolutionary engineering scientists describe an elementary structure of the system and then evolve this structure toward the desired system. Genetic algorithms are used for evolving such systems. One may see that EE, Genetic Programming (GP) (Koza *et al.*, 1999; De Garis, 1993b) and Evolutionary Computation (EC) are basically equivalent. EE is used to bring out “systems building ”. Usually, In EE approach, systems are built/evolved separately of their utilization. That is how it is commonly done. We refer to this method by a Non-Real-Time system evolution approach or an OFF-LINE system evolution approach. In contrast, the Real-Time system evolution approach or the ON-LINE system evolution approach deals with the fact that evolution and utilization are undertaken together. It’s a fact that evolution process is heavy on time; that is why the on-line approach is rarely taken into consideration. This paper deals with a Real-Time EE Application: Evolving in Real Time a neural net controller of a two-eyed, two-jointed, single robot-arm to track a target. the challenge is how to speed up the evolution process such that real time delays are respected, i.e., the target is caught as rapidly as it moves. Our solution consists in a new manner of using GA that reduces considerably the evolution time. In EE terms, to capture fundamentally this solution we present a concept: the neighborhood hypothesis (NH). We consider the NH as a criterion if satisfied makes ON-LINE evolution effective and realistic. In Sections 2 and 3 the problem specification and the adopted approach are set out. Genetic programming of the joint and the control module is described in Sections 4 and 5. Subsections 5.1 and 5.2 detail the solution in question. The results are shown in Section 6.

2. Problem Specification

We are concerned by a two–eyed, two–jointed, single robot–arm positioning simulation problem. Fig. 1 shows the basic setup. The aim of the task is to move the robot arm from its starting position X to the targeting position Y, where the position of Y is specified by the viewing angles of two eyes. J1 and J2 are the joints, E1 and E2 are the eyes positions, and JA1, JA2, EA1 and EA2 are the joints and eyes angles of the point Y. Eyes angles

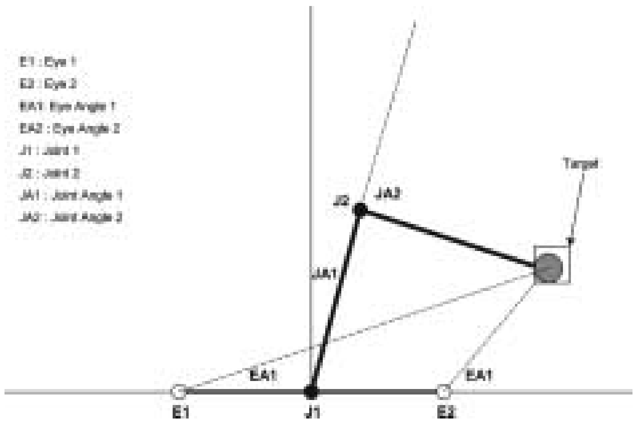

Fig. 1. Arm positionning.

values EA1 and EA2 are mapped to the joints angles values JA1 and JA2, so that the actual positions Y' and the desired position Y are as closer as possible (De Garis, 1993b). In the tracking process, the target is able to move according to a trajectory and the robot arm must point at the target. So, at each target move, the robot arm controller detects the viewing angles EA1, EA2, determines the target position and generates the joints angles JA1, JA2. For simulation's sake, the target's trajectory is generated by means of mathematical functions ($ax + b$, $\sin$ e, cosin e, etc. . .).

3. Approach to the Problem

A "modular", hierarchical (2 layered) approach to solving this problem, as described in (De Garis, 1993b), is illustrated by specifying two different neural net modules. The first is called the "Joint Module", it controls the joint angle JA$_i$ that a given joint opens to, for an input control signal of a given strength. And the second, called the "Control Module", receives inputs EA1 and EA2 from the two eyes and sends control signals to the joints J1 and J2 to open to angles of JA1 and JA2. Fig. 2 shows the basic circuit design. The joint modules, identical copies, are placed under the control of the control module. Each is fully connected, including connections from each neuron to itself. Between any two neurons there are two connections in opposite directions, each with a corresponding (signed) weight. The input and output neurons also have "in" and "out" external connections but these have fixed weights of 1 unit. The outputs of the control module are the inputs of the joint modules, as shown in Fig. 2.

The aim of the exercise is to use Genetic programming to evolve the values of the signs and weights of the various modules, such that the overall circuit performs as desired. This is done in a modular fashion: first the joint module is evolved apart. This mean that the joint module is evolved in Non-Real Time manner (off-line). Then the joint module characteristics are frozen, and the overall system is evolved in Real Time (on-line). Thus, the control module characteristics are evolved for each target position so that the arm moves as close as possible to the specified goal point.

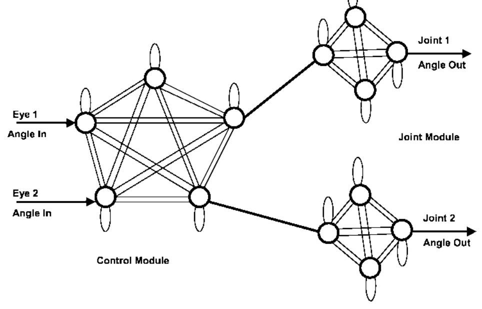

Fig. 2. Control and joint modules neural nets.

4. Genetic Programming of the Joint Module

4.1. *The Neural Network Model*

The Neural Network (NN) is completely specified by its topology and the functionality of the artificial neuron.

4.1.1. *Topology of the NN*

Recurrent NN is chosen. A recurrent NN is a self fully connected NN. This topology has the advantage to do not deal with any specific details of the NN, such as number of layers, number of neurons in each layer, number of hidden layers, how neurons are connected and so on. Only the number of neurons has to be chosen. For the joint module the NN contains 4 neurons as in Fig. 2.

4.1.2. *Artificial Neuron Functionality*

Figs. 3 and 4 show in details the behavior of the artificial neuron. The output is a sigmoid. External inputs are used to control the NN (De Garis, 1993b). We use them to inject input data into the NN.

$S_j \equiv$ input signal "j".

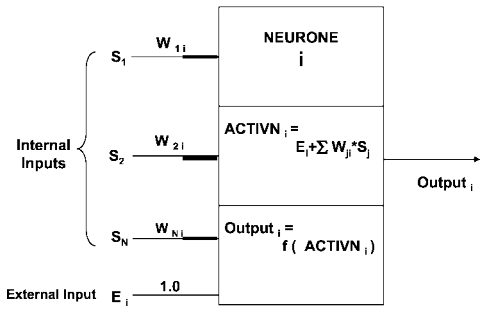

Fig. 3. Artificial neuron.

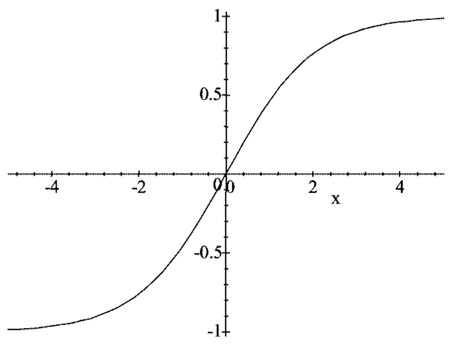

Fig. 4. Neuron output function: $\frac{2}{1+e^{-Activ_i}} - 1$.

W_{ji} $\equiv$ weight associated to S_j for the neuron "i".

E_i $\equiv$ external input of the neuron "i"; when used its weight is clamped to 1.0 if not to 0.0.

4.2. *Genotype and Phenotype of the Joint Module*

The set of weights determines fully the behavior of the recurrent NN. Each weight is coded (onto a GA chromosome) with a sign, (where 0 means an excitatory synapse, 1 means an inhibitory synapse) followed by a specified number of bits. Thus the chromosome witch represents the joint module genotype is simply: the set of weights in code. Fig. 5 describes the chromosome structure. The phenotype of the joint module is obtained by decoding the chromosome into a 4×4 table. Each table entry (i, j) contains the weight value of the connection from the neuron j to neuron i.

W_{ij} denotes the weight associated to the signal S_j coming from neurone j to neurone i.

Each weight is coded with 6 bits and has its value in $[-1, +1]$.

The chromosome length is 96 bits$= 4 \times 4 \times 6$.

In Fig. 5 W_{11} is interpreted as follow:

Bit 0 $= 1 \Rightarrow$ weight is a negative value.

Bit 1 $= 1 \Rightarrow$ weight $=$ weight $+1 \times 2^{-1} =$ weight $+1 \times 0.5$,

Bit 2 $= 1 \Rightarrow$ weight $=$ weight $+1 \times 2^{-2} =$ weight $+1 \times 0.25$,

Bit 3 $= 0 \Rightarrow$ weight $=$ weight $+0 \times 2^{-3} =$ weight $+0 \times 0.125$,

Bit 4 $= 1 \Rightarrow$ weight $=$ weight $+1 \times 2^{-4} =$ weight $+1 \times 0.0625$,

Bit 5 $= 1 \Rightarrow$ weight $=$ weight $+1 \times 2^{-5} =$ weight $+1 \times 0.03125$.

So: $W_{11} = -(0.5 + 0.25 + 0.0625 + 0.03125) = -0.84375$.

4.3. *The Adaptation Function of the Joint Module*

Since a genetic algorithm is used to evolve the values of the weights, such that the actual output was as close as possible to the desired output, 21 input values ranging from -1 to $+1$ by steps of 0.1, were used. The desired output values were chosen to be half of the (clamped) input values, thus ranging from -0.5 to $+0.5$, and were interpreted as being the number of turns of a joint, (e.g., $+0.5$, i.e., half a turn, would mean a joint angle of 180 degrees. A positive angle was clockwise). During the evolution process, these 21 input values were presented one at a time to the joint module NN. The NN output value was recorded, as well as the error between this value and the desired value (i.e., half the input value). This procedure was repeated for each of the 21 clamped input values, for each

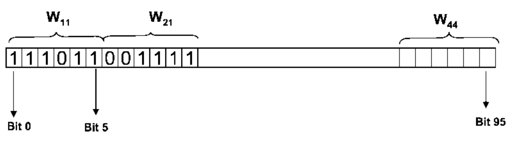

Fig. 5. Chromosome structure.

chromosome. The quality measure (fitness) used in the evolution of the Joint Module was the inverse of the sum of the squares of the differences between the desired and the actual output values A_i, i.e.:

$$\text{Fitness} = \frac{1}{\sum_{-10}^{+10} (i \times 0.05 - Ai)^2}.$$

4.4. *The Joint Module Evolution Process*

Basically a GA guides this process. At each evolution step, the GA generates a new population of offspring by mean of genetic operators (selection, crossover, and mutation). For each individual of the offspring, i.e., a genotype (chromosome) is associated a phenotype (weight table). The weight table and the NN simulator brought together represents a potential solution. The adaptation function, i.e., the fitness of this solution is then computed as in 4.3.

The evolution process continues until a good solution is found. Fig. 6 describes the overall joint module evolution process. Because the joint module is evolved apart we call this manner a Non-Real Time Evolution or an OFF-LINE Evolution and we say that the joint module is evolved in OFF-LINE.

4.4.1. *The Recurrent NN Simulation Algorithm*

In neural net terms our application is known as a neural net with Time Independent Input (TII) and Time Independent Output (TIO); for this reason the recurrent NN must run until stabilization. The Stabilization cycle number must be determined separately, if not the result is unpredictable.

Giving the weight table W, the external input data vector E and the output signal vector S, the recurrent NN simulation stands as in Fig. 7.

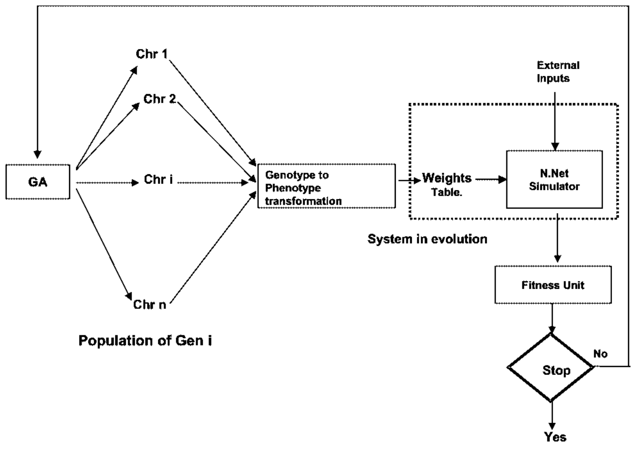

Fig. 6. Joint module evolution schema.

```
Loop
 {
    For i= 1 to neuron number
    {
      For j=1 to neuron number
      { ACTIVi = ACTIVi + Wji * Sj }
      ACTIVi = ACTIVi + Ei;
      Si = 2 / (1 +  exp ( - ACTIVi ) ) -1
    }
 } Until Stabilization
```

Fig. 7. Recurent NN simulator algorithm.

4.4.2. *Joint Module Evolution Process Parameters*

The following parameter values concern the off-line, i.e., non-real-time evolution process.

Neuron number = 4,
Number of input neurons = 2,
Number of output neurons = 1,
Weight code length = 6 bits,
Chromosome length = 96 bits,
NN stability cycles number = 50,
Type of crossover: Uniform crossover,
Crossover probability = 0.6,
Mutation probability = 0.001,
Selection strategy: Roulette wheel,
Evolution strategy: Elitism,
Scaling constant = 2.0,
Population size = 100,
Number of generation = 100.

5. Genetic Programming of the Control Module

The Neural Network model, the Genotype and the phenotype of the control module are identical to those of the joint module. The neuron number of the control module is set to 6 so the chromosome length is $6 \times 6 \times 6 = 216$ bits. Our main goal is to approach the tracking problem by mean of Real Time Evolutionary Computation (Real Time GP). So, Once the joint module is evolved, the weights are frozen. The joint modules, identical copies, are placed under the control of the control module as in Fig. 2. Therefore, Real time evolution concerns the control module only.

5.1. *The Adaptation Function of the Control Module*

In tracking we need only to evaluate the distance between the target position and the arm position. Usually computation of the distance, i.e., $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ is used; but in real time context we need also that the adaptation function, i.e., the Fitness improves the GA convergence delays too. So the Fitness computation **must give advantage** to the individuals who are in the immediate neighborhood of the solution; and **must devaluate**, i.e., **exclude** the others. Below we present the Algorithm used in our application. This Algorithm computes two fitness. The fitness associated to the x coordinate: FITNESS_X and the fitness associated to the y coordinate: FITNESS_Y. The final fitness is their product.

Fitness Computation Algorithm:

Let	x_t, y_t	the coordinates of the target position,
	x_a, y_a	the coordinates of the arm position,
	x_t, y_t	computed according to the eye angles EA1 and EA2,
	x_a, y_a	computed according to the joint angles JA1 and JA2,

S_x be the x coordinates product. Its sign determines if x_t and x_a are in the same half plan side according to the x axis.

S_y be the y coordinates product. Its sign determines if y_t and y_a are in the same half plan side according to the y axis.

So:

$$S_x = x_t \times x_a,$$
$$S_y = y_t \times y_a.$$

Let

$$SD_x = (x_t - x_a)^2,$$
$$SD_y = (y_t - y_a)^2,$$
$$\text{if } (S_x \prec 0) \text{ then } SD_x = SD_x + 4;$$
$$\text{if } (S_y \prec 0) \text{ then } SD_y = SD_y + 4.$$

We add 4 to SD_x or SD_y to devaluate the individuals not in the immediate neighborhood of the solution.

The value 4 is the maximum distance; because the arm is two (2) units long.

Let

FITNESS_X	the fitness associated to the x coordinates,
FITNESS_Y	the fitness associated to the y coordinates.

Then,

$$\text{if } (SD_x \prec 1) \text{ then } \text{FITNESS}_X = 1 - SD_x; \quad \text{else } \text{FITNESS}_X = 1/SD_x.$$

and

$$\text{if } (SD_y \prec 1) \text{ then } \text{FITNESS}_Y = 1 - SD_y; \quad \text{else } \text{FITNESS}_Y = 1/SD_y.$$

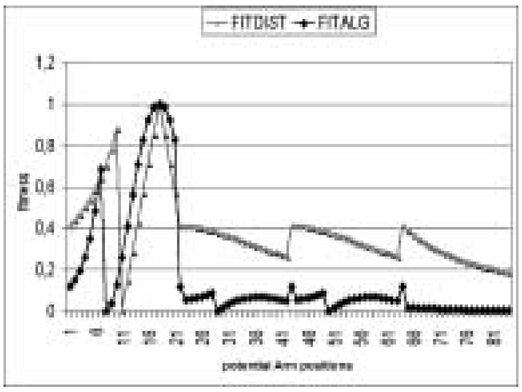

Fig. 8. Fitness over 100 arm positions using author algorithm: FITALG, and the standard distance method FITDIST.

And finally the overall fitness is:

$$\text{FITNESS} = \text{FITNESS}_X \times \text{FITNESS}_Y.$$

Fig. 8 shows that the fitness computed by mean of the algorithm, i.e., FITALG is more **restrictive** than the fitness computed by mean of the distance, i.e., FITDIST. This means, in GP terms, that the *solutions search space* is **significantly reduced**. Also, if we consider that the best solutions have their fitness $\in [0.8, 1]$, FITALG presents 7 solutions while FITDIST presents only 3 solutions. Real time evolution process must be rapid and very elitist, we think that the algorithm presented above is more suitable for this kind of task.

5.2. *The Control Module Real Time Evolution Process*

The control module neural net parameters must be learned during the target tracking. For each target position the evolution process configures the control module neural net. This is an ON-LINE or a Real-time learning. It is well known that the evolution processes are heavy on time; this is due to the GA convergence delay. Therefore the challenge is to reduce these delays. In the Neighborhood Hypothesis s/section we explain the adopted solution. Fig. 9 describes the overall Real Time evolution process.

5.2.1. *Neighborhood Hypothesis (NH)*

The main task is to evolve a new version of the control module according to each target position along the trajectory target . This means for each target position we start up a new evolution process. Since real time process requires rapidity especially in tracking and because the GA convergence is time consuming ($>$ 100 generations), the major problem is “HOW GENETIC ALGORITHM CONVERGENCE DELAY COULD BE REDUCED?”

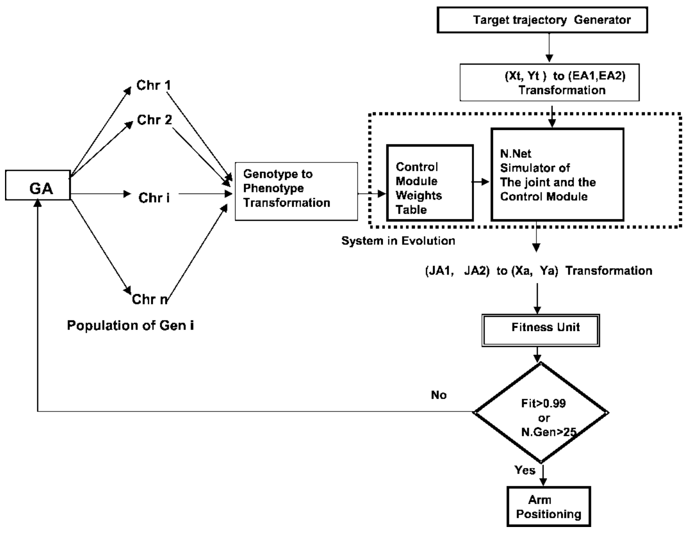
Target trajectory Generator
(Xt, Yt) to (EA1,EA2) Transformation
Chr 1
Chr 2
Chr i
Chr n
GA
Genotype to Phenotype Transformation
Control Module Weights Table
N.Net Simulator of The joint and the Control Module
System in Evolution
(JA1, JA2) to (Xa, Ya) Transformation
Population of Gen i
Fitness Unit
No
Fit>0.99 or N.Gen>25
Yes
Arm Positioning

Fig. 9. Control module real time evolution schema.

Analyzing the tracking problem, one can easily see that the target trajectory, mathematically speaking, must be continuous.

We define the notion of two adjoining evolution processes to speak about evolution processes associated to two adjacent targets positions.

According to the continuity property, we put the assumption that: two adjoining evolution processes are close to each other. In genetic programming terms this assumption means that for two adjoining evolution processes the actual evolution process could **inherit** from the preceding one. This is what we call the Neighborhood Hypothesis (NH).

To put in practice the NH our solution consists of the following: in using the GA, Instead of generating the first population randomly for each target position, we assign the last population of the preceding target position to the first population of the actual target position. Except for the initial target position.

Our investigations had shown that this technic reduces considerably the GA convergence delays. Also, the notion of Neighborhood can be applied to more than two target positions. A neighborhood is characterized by its size; the Neighborhood size depends on the way of sampling the target's trajectory.

5.2.2. *Control Module Real Time Evolution Process Parameters*

To make the final solution more efficient, some GA parameters as population size, generation number, crossover and mutation probability were tuned. The following parameters values concern all the real time evolution processes.

Neuron number = 6,
Number of input neurons = 2,
Number of output neurons = 2,
Weight code length = 6 bits,
Chromosome length = 216 bits,

NN stability cycles number = 50,
Type of crossover: Uniform crossover,
Crossover probability = 0.7,
Mutation probability = 0.01,
Selection strategy: Roulette wheel,
Evolution strategy: Elitism,
Scaling constant = 2.0,
Population size = 50,
Number of generation = 25 or until Fitness$\succ$0.99.

6. Experiments and Results

Implementation has been done with the Neural Net Evolution Software (**NES**); developed by our team. Seven experiments were performed to verify the correctness of our approach. Each experiment concerns the application of the Real Time evolution process to a specific target's trajectory. Target trajectories are generated by mean of mathematical functions. For each experiment the NES gives a graph and an illustration. Figs. 10, 11, 12 and 13 exhibit two experiments. The graph displays a positions target/generations numbers curve. This curve points out for each target position the number of generation for which the GA converges, i.e., fitness > 0.99. The illustration shows the tracking process simulation. The target is represented by a square and a circle ends the robot arm. The square enclosing the circle illustrates that the arm position is closer to the Target position. Fig. 14 shows overall results.

7. Conclusion

As shown in Section 6, experiments were organized according to the type of the target's trajectory and the neighborhood size. Perturbations were added by using random effect. For each experience the real-time evolution process is stopped when the fitness is greater

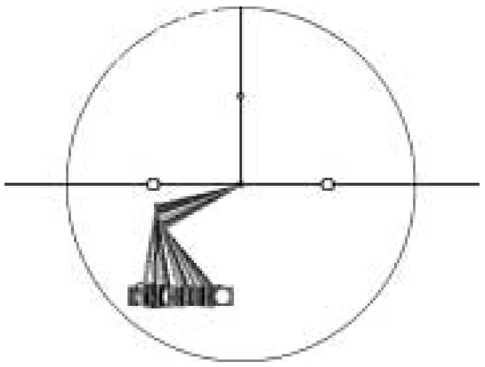

Fig. 10. Linear target trajectory (Y constant, $X = X + 1$). Tracking was successful at each target position (100 positions).

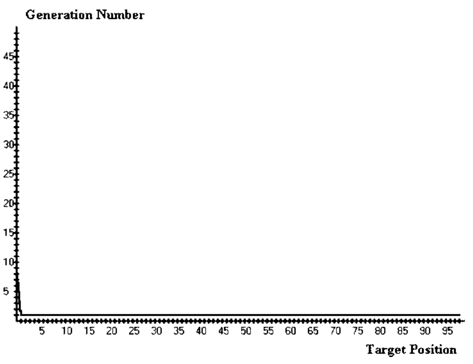

Fig. 11. Graph associated to Fig. 10. The evolution process succeed, i.e., the GA converges at the first generation for all the Target positions except the initial.

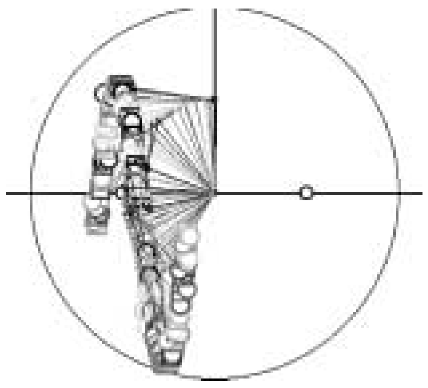

Fig. 12. Sinusoidal target trajectory: step $<= 10$, progression: PI/50. Tracking was succesful at each target position (100 positons).

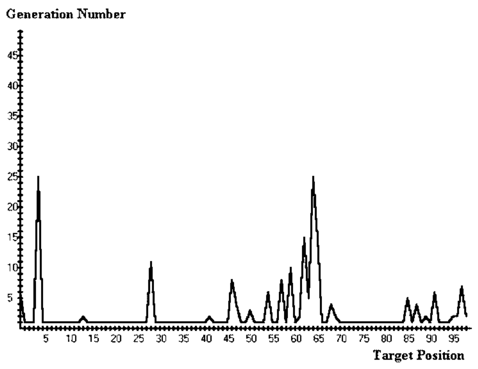

Fig. 13. Graph associated to Fig. 12 the evolution process succeed, i.e., the GA converges at the first generation for all the Target positions except: (11 pos at gen $<= 5$), (5 pos at $15 >=$ gen $>= 5$) and (2 pos at gen $= 25$).

		Target trajectory type							
		Linear Y constant $X{=}X{+}1$	Linear $X{=}X{+}\mathrm{Rand}\ (6)$ $Y{=}Y{+}\mathrm{Rand}\ (6)$	Sinusoidal step <=3 progression of PI/50	Sinusoidal step <=10 progression of PI/50	Cosine step <=3 progression of PI/50	Random neighbor-hood size =10	Random neighbor-hood size =25	Rate
Convergence delays	First generation	99	91	99	81	94	99	95	**94%**
	>1st& <=10 generation	1	4	1	15	5	1	5	**5%**
	>10&<=25 generation	0	5	0	4	1	0	0	**1%**

Fig. 14. Overall results.

than 0.99 or generation number is greater than 25. Statistics of the overall experiments indicate that over 700 target positions 658 real-time evolution processes converge at the first generation, 32 converge at generation number less than 10; therefore this gives an efficiency rate of 99%. These results are positive and prove that the neighborhood hypothesis (NH) is consistent. Also this proves that the GA as used in subsection 5.2 is well adapted to the NH. An important problem remains: how to detect that a specific application verify the Neighborhood Hypothesis (NH).We believe that NH can highly improve Real-Time EE.

Acknowledgments

We are grateful to the INFORMATICA referees for their insightful comments and suggestions. Also, we would like to thank especially the Algerian National Scientific Research Council for his support of this research under contract No: B2201/01/02.

References

Brooks, R.A. (1992). Artificial life and real robots, in toward a practice of autonomous systems. In F.J. Varela and P. Bourgine (Eds.), *Proceedings of the First European Conference on Artificial Life*. MIT Press.

De Garis, H. (1993a). Circuits of production rule gennets: the genetic programming of artificial nervous systems. *Int. Conf. on Neural Networks and Genetic Algorithms. Lecture Notes in Computer Science*. Springer Verlag.

De Garis, H. (1993b). *Genetic Programming: GenNets, Artificial Nervous Systems, Artificial Embryos*. Phd thesis.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley Publishing Company.

Hertz, J., A. Krogh and R.G. Palmer (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City CA.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, Univ of Michigan Press.

Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, USA 79. pp. 2554–2558. Reprinted in [ANDERSON & ROSENBERG 1988].

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, **43**, 59–69. Reprinted in [ANDERSON & ROSENBERG 1988].

Koza, J.R., and F.H. Bennett III, D. Andre and M.A. Keane (1999). *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers.

Langdon, W.B. (2001). *Genetic Programming Bibliography*. Revision Date: 2001/05/05. `ftp://ftp.cs.bham.ac.uk/pub/authors/W.B.Langdon/biblio/gpsubmit.html.`

Lehireche, A., A. Rahmoun and A. Gafour (2001). Highlights on the evolutionary engineering approach: the EE-method. ACS/IEEE, AICCSA'01, *International Conference on Computer Systems and Applications*, Beirut. Copyright 2001 IEEE.

Macias, N.J. (1999). Ring around the PIG: a parallel ga with only local interactions coupled with a self-reconfigurable hardware platform to implement an O(1) evolutionary cycle for evolvable hardware. In *Proceedings of 1999 Congress on Evolutionary Computation*. Copyright IEEE.

McCulloch, W.S, and W. Pitts (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 115–133. Reprinted in [ANDERSON & ROSENBERG 1988].

Minsky, M.L., and S.A. Papert (1969). *Perceptrons*. MIT Press, Cambridge Mass.

Rahmoun, A., and M. Benmohammed (1998). On deriving optimal fuzzy expert systems by genetic algorithms and competitive learning. In *IEE Proceedings on Control Theory*, Great Britain.

Rojas, R. (1996). *Neural Networks: a Systematic Introduction*. Springer, Berlin, Heidelberg.

Spears, W.M, and K.A. De Jong (1991). On the virtues of parameterized uniform crossover. In *Proceedings of the 4th Int. Conf. on Genetic Algorithms*. San Diego, Morgan Kaufmann.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

A. Lehireche received the engineer degree in computer science from National Institute of Computer Science Algiers, and the magister diploma in computer science from the University of Science and Technology of Oran (USTO), in 1981 and 1993 respectively. He is currently a PhD candidate in the Computer Science Department at the University of Sidi Ble Abbes(UDL), Algeria. He is a senior lecturer at the U.D.L Computer Science Department, and a head of the Evolutionary Enginering Project. His research interests include genetic programming, evolutionary engineering, neural networks, and so far computer science theory.

A. Rahmoune received the PhD degree computer science from UDL University of Sidi Ble Abbes, in 1999. He is an associate professor at Faculty of Planning and Management, King Faisal University, KSA. His research interests include neural networks, genetic algorithm, fuzzy systems and control theory.

Neuroninio tinklo kontroliuojamo roboto manipuliatoriaus evoliucinis vystymas realiame laike

Ahmed LEHIRECHE, Abdellatif RAHMOUNE

Autoriai pristato evoliucinio mokymo principais paremtą algoritmą, skirtą roboto manipuliatoriui kontroliuoti. Algoritmo veiksmingumas yra grindžiamas simuliaciniais tyrimais. Simuliacinis modelis susideda iš virtualaus manipuliatoriaus su 2 sąnariais, kontrolinio bloko, kurį atstoja genetiškai mokomas neuroninis tinklas ir dviejų sensorių, gebančiu nustatyti kampą iki sekamo objekto. Šiame straipsnyje autoriai demonstruoja, kaip jų pateikiamas algoritmas evoliucionuoja ir laikui bėgant vis geriau ir geriau sugeba sekti judantį objektą.