EMBEDDING A DISTRIBUTED EVOLUTIONARY SYSTEM INTO A POPULATION OF AUTONOMOUS MOBILE ROBOTS

EDUARDO D. V. SIMÕES and KEITH R. DIMOND

Electronic Engineering Laboratory - University of Kent at Canterbury
Canterbury, United Kingdom, CT2 7NT

Abstract

This paper describes a fully embedded distributed evolutionary system that is able to achieve collision-free navigation in a few hundreds of trials. It reports the first experimental proof of the embedded evolution concept applied to the evolution of morphology and an unstructured control circuit of a population of six real robots in real time. Evolution selects the appropriate sensor configuration and speed levels that the robots should incorporate to perform specific tasks. The results show the influence of different mutation rates in the performance of the system. This work produced a genetic system where the population exists in a real environment.

Keywords
Evolutionary system, embedded evolution, adaptive robotics.

1 Introduction

This work is concerned with automated synthesis of robotic embedded controllers using Evolutionary Computation [1]. Evolutionary methods have been employed for developing robot controllers automatically in simulation [2], on physical systems [3], and combinations. Specifically, this work concentrates in embedding an evolutionary algorithm within a population of physical robots.

The term *Evolutionary System* in this work refers to an environment where the individuals physically exist and artificially breed and die, to give place to the next generation. It is important to distinguish it from the work of other authors that involve simulation in some of the evolutionary phases [4, 5].

To date, only two publications report the implementation of an evolutionary system fully on-board of a population of real robots, working completely independent of external computation and human intervention to evaluate, reproduce, and reposition the robots for new trials [6, 7]. These articles are: Watson et al. [8] (evolving only the controller) and the publication of the first results of our work [9] (evolving the controller and morphology). They provide, together with this work, the first experiments with a new concept: Embedded Evolutionary Systems.

What is proposed is the physical implementation of a genetic system containing the robots as the individuals and a genetic code (bits stored in the RAM memory of the robots) that specifies the configuration of their control device, the *evolvable controller*, and their speed and sensor organisation, defined here as *morphology* [10].

Evolving unstructured architectures for robot control has been attempted before by Thompson in [5], where he evolved a dynamic state machine to drive a small mobile robot and in [11], where he tried to evolve an FPGA connected directly to the motors to produce a pulse modulated signal. Both attempts employed simulation, and one real robot was used to evaluate the solutions. To date, no work has been reported where the evolution of an unstructured controller has been attempted with a real robot population in real time [6]. Therefore, these experiments provide the first results obtained from the evolution of an unstructured control architecture by an embedded evolutionary system.

1.1 Problem Delimitation

A simple task-behaviour was chosen: collision-free navigation [3] to allow the development of the system in relatively low-cost robots. Therefore, more robots could be built and evolution could benefit from more diversity in the population. The main issue considering functional specification in an evolutionary system is to tell evolution *what* the robots have to do, without telling it *how* they are going to achieve that [12]. In our case, the robots are encouraged to explore the environment, going as fast as possible without colliding into the obstacles or each other. Because the workspace contains various robots, the environment also includes some robot-to-robot interference [13] (e.g., collisions between robots and reflection of the infrared signals by approaching robots).

1.2 The Workspace

The workspace consists of six autonomous mobile robots working in a 2.50m × 2.50m domain, where they can navigate, avoid obstacles, and

perform specific tasks. The robots have eight infrared proximity sensors and 10 speed levels for each motor.

Fig. 1 shows the six robots navigating in their working domain. The robot architecture consists of a two-wheel differential-drive platform (20cm diameter), containing a Motorola 68HC11 - 2MHz with 64Kb of RAM. It exchanges information with the other robots at 1.2Kbps by a 418MHz AM radio. Both robots and workspace were specially built for the experiments. All robots have a binary bit string, the "chromosome", containing the genetic code that specifies their control device and physical features, such as speed and position of the sensors. All eight proximity sensors are connected to the sensor module (see Fig. 2), which is configured by the chromosome. The module can individually enable or disable the sensors, changing the number of active sensors and, consequently, their position in each generation. Therefore, the physical, embodied characteristics of the robot can be modified. The robot has four round bumpers attached to eight contact sensors (*Cs1* to *Cs8*). These sensors permit the supervisor algorithm to pinpoint the location of a collision.

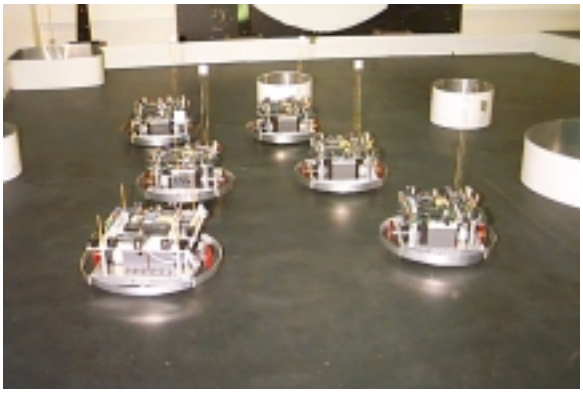

Fig. 1: The six robots and their working domain

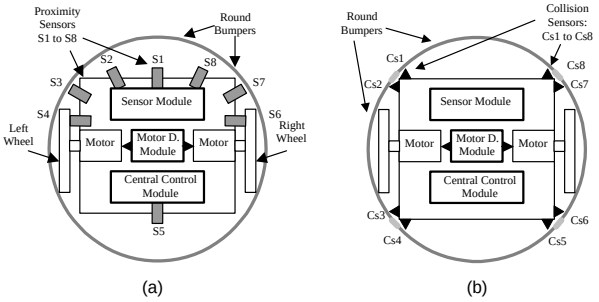

Fig. 2: Robot architecture representing the proximity sensors (a) and round bumpers (b)

2 Individual Control Strategy

The robot architecture can conceptually be seen as a central control module interfacing all other functional modules (see Fig. 4). The modules were implemented using a combination of dedicated hardware and software. The sensor and motor drive modules and the navigation control are configured by subsets of the chromosome that indicate the number of sensors used, their position in the robot periphery and the speed levels of the robot.

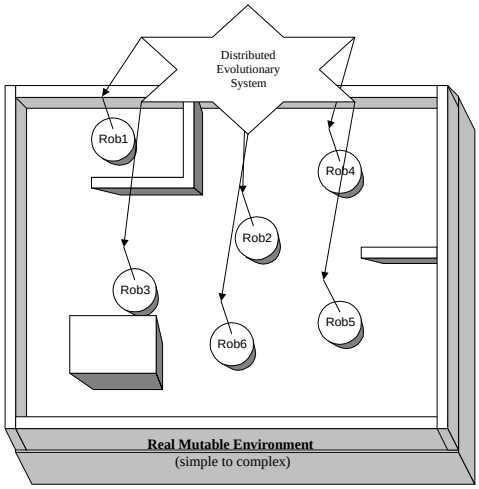

Fig. 3: The radio link, forming a decentralised evolutionary system

The goal of the implemented evolutionary system is to train automatically the team of robots to interact with an unforeseen environment in real time. Therefore, the robots must evolve while deployed "in the field", using the real world to act as "its best model" [4]. The robot individual capacity is quite simple, but presents the necessary evolutionary capabilities. The evolutionary system is not based in an external computer, but is distributed between the robots and coexists with their evolvable controller inside the microprocessor.

The evolutionary control circuits of all robots communicate through an embedded radio, as shown in Fig. 3, to control the complete evolutionary process. They combine and form a global decentralised evolutionary system [14]. They process the data stored in the chromosome and send the configuration parameters to the navigation control and the other modules. This global system controls the evolution of the robot population from generation to generation. It is responsible for selecting the fittest robots (the most-adapted to interact with the environment), mating them with the others by exchanging and crossing over their chromosomes, and finally reconfiguring the robots with the resultant data (the offspring).

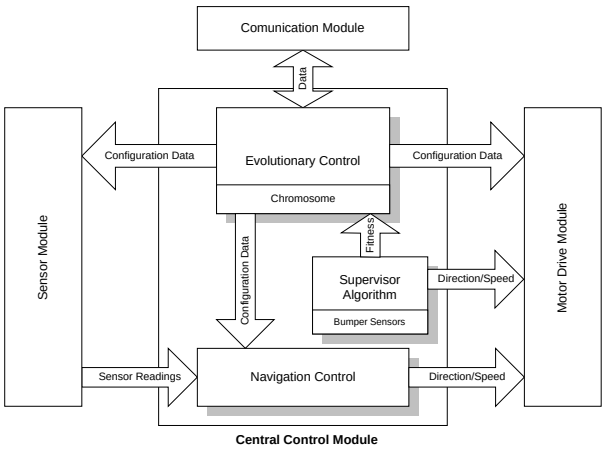

Fig. 4: The Central Control and main subsystems

A supervisor algorithm monitors the robot performance, informing the evolutionary control how well-adapted it is to the environment. The supervisor algorithm is responsible for activating a rescue routine, a built-in behaviour that is able to manoeuvre automatically the robot away from a dangerous situation once it is detected by the contact sensors that determine the occurrence and position of a collision. When activated, the rescue routine will take control of the robot until it is safely recovered. The robot is then punished by decreasing its fitness value.

The navigation control processes the information of the sensors and decides what the robot has to do. Then, it sends a command to the motor drive module, which will control the speed of the motors to make the robot manoeuvre accordingly. The navigation control is the centre of the autonomous navigation of the robot. Configured by the parameters stored in the chromosome, it drives the robot independently. Evolution is responsible for adjusting these parameters so that the robot performs well in the environment.

2.1 Navigating with an Unstructured Controller

In this work, the embedded evolutionary system will attempt to evolve a completely unstructured control circuit (i.e., a controller that has an unconstrained logic circuit, which can produce any binary function of its inputs). Instead of using a structured, modular neural network (which can produce limited functionality) to implement the navigation control circuit as shown in our previous paper [9], this work prefers an unstructured, undefined architecture, which we called the *black box* [15].

In this experiment, all sensors have only one bit of precision and they were set to work at the range of 15cm. The controller used eight commands to control the motor drive module: *Front Fast* (*FF*); *Front Slow* (*FS)*; *Turn Left Short1* (*TLS1*); *Turn Right Short1* (*TRS1*); *Turn Right Short2* (*TRS2*); *Turn Left Short2* (*TLS2*); *Turns Right Long* (*TRL*); and *Turn Left Long* (*TLL*). *FF* moves the robot forward with maximum speed. *FS* moves with half speed. To turn left/right short, the robot moves with reverse direction in one of its motors (with both motors at maximum speed), causing a spin around its own axis. The difference between *TRS1* and *TRS2* is that in the later, the robot keeps turning for 200ms, while the duration of the other commands is just one iteration (10ms more or less). In *TRL* and *TLL*, the robot turns right/left by breaking one wheel and turning around it with the other one in maximum speed. Fig. 5 shows how the black box is connected to the sensors and the motor drive module. These eight commands are encoded by three bits.

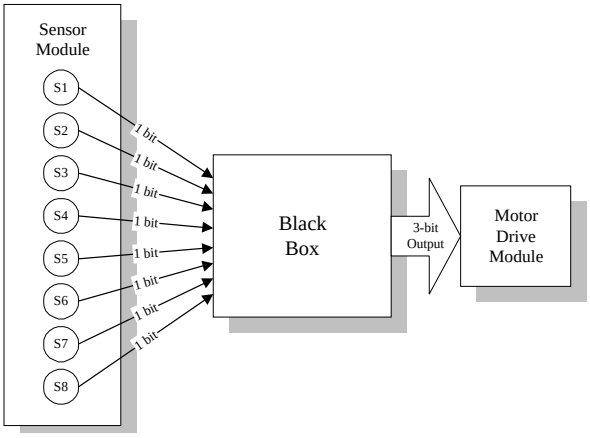

Fig. 5: The Black Box controller

The same strategy that was used to implement the circuit of the neurons in [15] as lookup tables is applied here to implement the navigation control circuit. The controller is represented by a black box that can generate any binary logic function of its eight binary inputs. Which logic circuit is actually used inside the black box to implement this logic function does not matter in this case, since only its behaviour is relevant to the problem.

The natural way to implement the black box controller is by using 256 bytes of the robot RAM memory, where the output of the black box controller is a byte that can store in its eight bits the contents of eight lookup tables. The first three bits, b_0, b_1, and b_2, are used to form the 3-bit command that controls the motors. This strategy resulted in a powerful and fast controller, since only a single step or command line is necessary to implement the whole controller: *Command = Memory[Sensors]*. Considering that *Sensors* is a 1-byte long variable containing the

sensor readings (8 bits) converted to an integer that addresses the memory array. *Memory[256]* is an array of 256 bytes that stores the contents of the lookup tables, and *Command* is the variable sent to the motors.

Evolution was allowed to manipulate the memory contents directly, until a navigation control circuit emerged. Therefore, every byte of the memory was ordered in the chromosome to form a 2048-bit string. Crossover and mutation can affect each one of these bits as a normal binary chromosome. After these operations are completed, the 2048 bits in the chromosome are grouped again into 256 bytes and stored in the black box memory. Only the first three bits in the byte (b_0, b_1, and b_2) are relevant for this experiment, and the other ones (b_3 to b_7) are ignored by the robot. Therefore, only 768 bits, which correspond to the first three tables are relevant to evolution. Hence, this is the size of the genotype of the robots, and considering that any one of them can produce a different phenotype (i.e., there is no neutrality in this case), the current search space is considerably large: $2^{768} = 1.55 \times 10^{231}$.

3 Evolutionary Control System

The cyclic procedure of the robots, a *generation* in evolutionary computation terms [10], is exemplified in Fig. 6. This cyclic procedure was inspired by the natural world where some birds, for example, have a working or foraging season and a mating season, where they concentrate their attention in finding a mate and reproducing. The robots do not pursue reproductive activities concurrently with their task behaviour. Instead, they perform a working season, where they execute the selected task in the environment (or working domain) and are evaluated according to their performance.

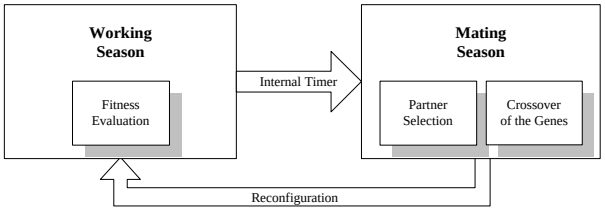

Fig. 6: The cyclic procedure of the robots

A internal timer indicates the beginning of the mating season. In this season, the robots communicate to let the others know their fitness value. They start emitting a "mating call" through the radio, where they "shout" their identification, fitness values, and chromosomes. The best robots survive to the next generation, breeding to become the "parents" of the new individuals. Assuming that new robots cannot really be created spontaneously, the offspring must be implemented by reconfiguring selected old individuals. The robots then start another cycle. In other words, the best-adapted robots "survive" to the next generation, while the others "die" after mating, to lend their bodies to their offspring.

3.1 Crossover Strategy

The genetic material specifies the configuration of the robot control device and morphological features, as shown in Fig. 7. A random exchange of the genes from the parents forms the resultant chromosome. This strategy is called *uniform crossover* [1], although here only one offspring is produced. Therefore, a gene is selected from the father or the mother to occupy the corresponding position in the offspring chromosome. After the crossover is completed, a mutation phase starts. Each gene in the chromosome has a probability of *M*% of being selected and binary inverted (e.g., new gene = *NOT*(gene)).

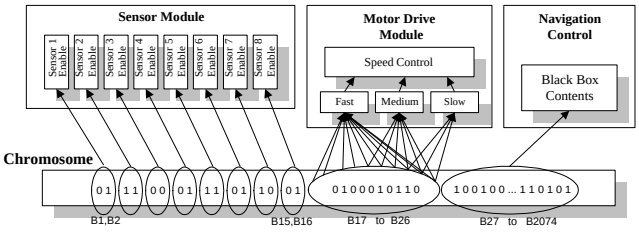

Fig. 7: The genes in the chromosome

4 The Experiments

The experiments applied a simple fitness function in order to prevent biasing evolution towards a pre-conceived solution. Rule 3 punishes the robots that keep turning for more than 15 seconds, encouraging them to move forward. Rule 4 does not punish the robots that are turning, for they may be attempting to avoid an obstacle when the collision occurred. The selected fitness function for this experiment is:

1- Start with 4096 points;
2- Reward: increase fitness by 10 points every 1 second of movement;
3- Punishment: decrease fitness by 30 points for every time command is not FF or FS for more than 15 seconds;
4- Punishment: decrease fitness by 10 points for every collision if command = FF or FS.

The duration of a generation is 60s and mutation rate is 0.5%. The bits in the chromosome are randomly initialised for all experiments. After each

working season, the robots are selected to breed according to the selection rule reproduced below:

The robot with the highest fitness survives to the next generation and breeds with all other robots, which reconfigure themselves with the new offspring.

Fig. 8 presents the behaviour of the six evolving robots in 200 generations of 60 seconds. They evolved from a random controller with random speed levels and sensor configuration to a group of competing efficient solutions containing four or five sensors (*S1,S3,S4,S6,S8*; *S3,S4,S6,S8*; *S2,S3,S4,S7*), and a controller that learned how to deal with them. Robot 5 started with four sensors enabled (*S3,S4,S6,S8*), and was an efficient combination that was soon transferred to the other robots in the first 20 generations. Next, their controllers had to adapt to work with small variants from this configuration. This took more than 80 generations.

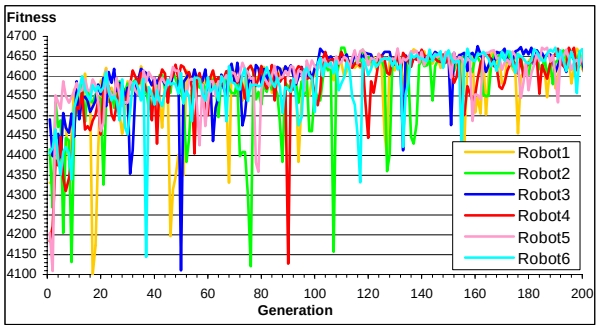

Fig. 8: Evolution of the *black box* controller and morphology

Since mutation was very low, all robots in the end of the experiment had the ability to produce similar performances. The small differences that still persisted were a consequence of the influence of noise and the interference between the robots. As the generations passed, there were fewer poorly-adapted robots crashing into the good ones, so the average performance increased.

Evolution made the robots travel slower in the beginning of the process, when their controllers were not well-developed to deal with all situations. An interesting event was observed in this experiment: after the population started to converge in the last 30 generations of the process, the sensor configurations still varied into a small set of solutions with four or five sensors enabled, but the controller configuration did not change much. In fact, it was able to perform the same manoeuvres regardless of being attached to four or five sensors. This helped to preserve a good performance while the sensor and speed configurations kept changing, since the controller did not need to adapt again after the robot morphology changed.

The chart shows that the population gradually converged to a small set of efficient configurations combining morphology and control that could drive the robots practically without colliding. The average fitness of the population oscillated in the beginning, but was very high in the end of the process. The system did not converge to an optimal solution, because it was not necessary, since many configurations obtained after 200 generations were able to produce the higher performance.

As a real environment is being used, the presence of stochastic noise will always introduce some variation. Nevertheless, the system presented a similar behaviour when the experiment was repeated 30 times. Fig. 8 was chosen for it was the closest trial to the average behaviour.

Fig. 9 shows a comparison of five mutation rates: 0.1%; 0.5%; 1%; 10%; and 50%. It presents the average results obtained by overlapping 20 experiments for each mutation rate. The figure suggests that with such a small population, after the first 20 generations or so the population relies only on mutation to increase its performance. Higher mutations help evolving in the beginning of the process, when there are more bad than good genes in the chromosome. It can be seen in the figure that 1% and 0.5% mutation evolved faster than 0.1% in the beginning of the process, but as the genes in the robot chromosomes became better, 0.1% mutation outperformed 1%. The best mutation was 0.5%, which showed a good performance during the whole process. It can be observed how the curves of the average fitness and the fitness of the best robot got more distant from each other when mutation was increased, showing that higher mutations make the population more disperse according to the fitness of the robots.

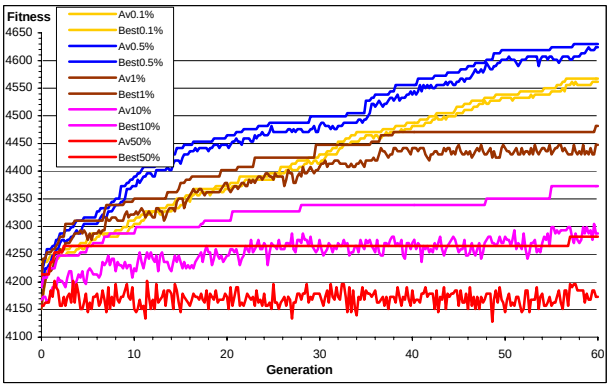

Fig. 9: Evolution of the *black box* controller for different mutation rates. *Av* is the average fitness of all robots and *Best* is the fitness of the best robot in the generation

5 Conclusion

During the experiments, the most important factor that interfered with the performance of the system was the stochastic noise arising from the interactions of real physical systems. The encountered noise in the experiments was:

1- The infrared signals reflected by the walls in every direction (e.g., two point reflection from one emitter in two walls and back to another receiver);
2- The dust on the floor, which can make the wheels slide differently, sometimes turning the robot differently than expected (this generates problems in performing programmed high-level actions);
3- The sensors are affected by environment conditions (such as intensity of illumination, colour of the walls, texture of the obstacles and floor, etc);
4- The inertia of motors and body, which vary with the robot speed causing difficulties to perform the expected turning routines.

The developed evolutionary system succeeded in evolving the real robots, initialised with random controllers and morphologies, reaching efficient solutions after 100 generations of 60 seconds, providing the first experiments of the embedded evolution of morphology and an unstructured controller. The system was able to detect the best position and range for the sensors, solving conflicts between sensors that interfere with each other and even incorporating the interference into the robot design. Evolution was also able to determine the ideal speed for the navigation of the robot according to the size and shape of the obstacles in the environment.

The most unexpected designs were produced and some proved efficient solutions that could not be achieved with the traditional top down design of robotic platforms. Although implemented into a specific group of 2-wheel differential-drive robots for a specific task, the described evolutionary system can be adapted to control other kinds of robots performing different tasks.

References

1. Layzell, P., Inherent Qualities of Circuits Designed by Artificial Evolution: A Preliminary Study of Populational Fault Tolerance, proceedings of the First NASA/DoD Workshop on Evolvable Hardware - EH99, pp. 85-86, 1999.

2. Lipson, H. and Pollack, J. B., Automatic Design and Manufacture of Robotic Lifeforms, in Nature, v. 406, pp. 974-978, 2000.

3. Keymeulen, D., Durantez, M., Konaka, K., Kuniyoshi, Y., and Higuchi, T., An Evolutionary Robot Navigation System Using a Gate-Level Evolvable Hardware, in Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science 1259, Springer-Verlag, pp. 195-209, 1997.

4. Brooks, R. A., Intelligence Without Reason, proceedings of the Twelfth International Joint Conference on Artificial Intelligence, Morgan Kauffman, San Mateo, CA, USA, pp. 569-95, 1991.

5. Thompson, A., Evolutionary Techniques for Fault Tolerance, proceedings of the UKACC International Conference on Control (IEE Conference Publication No.427), pp. 693-698, 1996.

6. Ficici, S. G., Watson, R. A., and Pollack, J. B., Embodied Evolution: A Response to Challenges in Evolutionary Robotics, proceedings of the Eighth European Workshop on Learning Robots, Wyatt, J. L. and Demiris, J. (Eds.), pp. 14-22, 1999.

7. Thompson, A. and Layzell, P., Evolution of Robustness in an Electronics Design, proceedings of the 3rd International Conference on Evolvable Systems, Springer Verlag, pp. 218-228, 2000.

8. Watson, R. A., Ficici, S. G., and Pollack, J. B., Embodied Evolution: Embodying an Evolutionary Algorithm in a Population of Robots, proceedings of the Congress on Evolutionary Computation, IEEE Press, pp. 335-342, 1999.

9. Simoes, E. D. V. and Dimond, K. R., An Evolutionary Controller for Autonomous Multi-Robot Systems, proceedings of the IEEE International Conference on Systems, Man and Cybernetics, v. 6, Oct., 1999, Tokyo, Japan, pp. 596-601, 1999.

10. Mataric, M. J., Challenges In Evolving Controllers for Physical Robots, in Evolutional Robotics: Special Issue on Robotics and Autonomous Systems, v. 19, n. 1, pp. 67-83, 1996.

11. Thompson, A., An Evolved Circuit, Intrinsic in Silicon, Entwined With Physics, proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware, Tsukuba, Japan, Springer-Verlag LNCS, pp. 390-405, 1996.

12. Floreano, D. and Mondada, F., Evolution of Homing Navigation in a Real Mobile Robot, in IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, v. 26, n. 3, pp. 396-407, 1996.

13. Seth, A. K., Interaction, Uncertainty, and the Evolution of Complexity, proceedings of the Fourth European Conference on Artificial Life, MIT Press, pp. 521-530, 1997.

14. Mataric, M. J., Reducing Locality Through Communication in Distributed Multi-Agent Learning, in Journal of Experimental and Theoretical Artificial Intelligence: Special Issue on Learning in Distributed Artificial Intelligence Systems, v. 10, n. 3, Weiss, G. (Ed.), pp. 357-369, 1998.

15. Simoes, E. D. V., Uebel, L. F., and Barone, D. A. C., Hardware Implementation of RAM Neural Networks, in Pattern Recognition Letters, n. 17, pp. 421-429, 1996.