Robotic Behavior Implementation Using Two Different Differential Evolution Variants

Víctor Ricardo Cruz-Álvarez[1], Fernando Montes-Gonzalez[1], Efrén Mezura-Montes[2], and José Santos[3]

[1] Departmet of Artificial Intelligence, Universidad Veracruzana (Mexico),
victor.g2004@gmail.com, fmontes@uv.mx
[2] Laboratorio Nacional de Informática Avanzada (Mexico),
emezura@lania.mx
[3] Computer Science Department, University of A Coruña (Spain),
santos@udc.es

Abstract. In Evolutionary Robotics (ER), Bioinspired Algorithms are used to generate robotic behavior. Several researchers used classic Genetic Algorithms (GA) or adaptations of GAs for developing experiments in ER. Here, we use Differential Evolution (DE) as an evolutionary alternative method to automatically obtain robotic behaviors, selecting a wall-following behavior as a representative example. We used an e-puck robot and the Player-Stage simulator for the experiments. We detail the results and the advantages when using the DE variants in our application with the simulated and the real robot. In order to optimize time for evolution, and test the resultant behavior in the e-puck robot, for our experiments we employed the Player-Stage simulator.

Key words: Evolutionary Robotics, Differential Evolution, Genetic Algorithm, Behavior Based Robotics

1 Introduction

In robotics, Evolutionary Robotics (ER) is an area that uses Bioinspired Algorithms, especially evolutionary algorithms, to develop the control architecture of an autonomous robot, and particularly the behavior controllers[3]. ER operates creating an initial population of candidate controllers and the population is repeatedly modified based on the fitness-function to optimize the robot controller[14]. The most used approach in ER was the classic Genetic Algorithm (GA)[5, 6], which generates solutions to optimization problems using operators inspired by natural evolution. These operators are crossover and mutation. Take for instance the work of Trefzer, et ál.[2] where an e-puck has been provided with a general purpose obstacle-avoidance controller both in simulation and in the real robot using GA. In similar works like[15, 16, 19–25] , the use of the GA is preferred over existent evolutionary methods like Evolutionary Strategies[12] and Differential Evolution[9]; some works in ER uses Genetic Programming (GP)[17] to evolve the robot controller[19, 18]. In general, ER employs GAs as optimization method, whereas uses Artificial Neural Network (NNs) as the elements to

evolve, since the NNs implement the robot controllers. The evolutionary method, after several generations, produces the fittest individuals (NNs) that generate the required behavior. The calibration of the associated parameters of the evolutionary method increments the chances to find near-optimal solutions.

The use of a standard platform in ER is commonly preferred. The e-puck[1] is an educational and research robot developed in the EPFL (École Polytechnique Féderale de Lausanne, Switzerland). In our experiments we use this robot. The size of the e-puck is about 7 cm diameter; the battery provides movement autonomy for about 3 hours, two stepper motors with a 20 steps per revolution and a 50:1 reduction gear. The e-puck has a ring of eight infrared sensors measuring ambient light and proximity of obstacles in a range of 4 cm. and has a VGA camera with a field of view of 36; for communication, the robot uses a wireless Bluetooth. The robot is controlled by a Microchip microprocessor dsPIC 30F6014A at 60MHz. The e-puck can be simulated under free and commercial applications, in this work, we use Player-Stage.

In the experiments, to evolve the wall-following behavior, we use an e-puck robot, the Player-Stage simulator and two Bioinspired Algorithms for the evolutionary optimization. These algorithms are the GA and the DE; in the DE, two variants are used to evolve the behavior, the *DE/rand/1/bin* and the *DE/best/1/bin*. We compare the results of using these differents algorithms. In Section 2, we discuss the methods mentioned. In Section 3, some topics related to Player-Stage are discussed. Section 4 describes the parameters for the neural network, the evolutionary methods and the e-puck robot. Results are presented in Section 5. In Section 6 we present a general discussion about our work. Finally, in Section 7 we provide a general conclusion.

2 Evolutionary Robotics and Bioinspired Algorithms

Generally, ER relies on the use of a Bioinspired Algorithms (generally the GA with binary representation [14, 20, 15]). The population is a set of candidate controllers. Most of the times a single objective evaluation function (fitness function) is employed due to the fact that the resultant robot behavior comes from a dynamic system made with the robot an its environment[7]. The use of Bioinspired Algorithms and Artificial Neural Networks offers a good solution to the problem of modeling behavior in maze-like environments[3]. Artificial Neural Networks have many applications in robotics due to their benefits as powerful classifiers, they are both noise and fault tolerant, which facilitates the robot to be driven in dynamical environments[4].

The optimization of neural controllers with an evolutionary method requires a representation, as a vector of the weights of the neural controller. Therefore, the codification of the weights vectors using an array representation is a common practice. This array represents the genetic material to be manipulated by artificial evolution. A single neural controller represents one of the many individuals that form a population, which in turn are candidates for providing a good solution for the task to be solved.

The quality of a candidate solution (*fitness*) is measured to acknowledge whether a solution is a good solution to the task we are trying to solve. The quality-space of all possible solutions forms the fitness landscape, with mountains and valleys, where landmarks in the mountains represents good solutions and landmarks in valleys are poor solutions. The search of the best solution in the landscape depends on the chosen evolutionary method. The search of the best solution requires the guide of an evolutionary method to move uphill to find better solutions, and variations in the landscape may cause the search to revolve around local minima and maximum areas.

In ER, the GA uses a binary population to represent the solutions; for exploration the GA employs the next operators: the reproduction of the individuals selected in pairs by the *crossover* of their genetic material, and *mutation* of some of the genetic material of the new individuals in the next generation. Also the GA uses *selection*, where a subset of population is selected to produce the next generation. Finally the GA *replace* the population with the new individuals. The iterative application of these operators to the genetic material (the NN weights) will produce refined solutions over time.

On the other hand, DE uses a population of real numbers (called vectors) that in our case represents the weights. The main idea behind DE is a scheme for generating trial parameter-vectors based on the population distribution. The basic idea of the DE is very simple, it works with two populations: P (the old generation) and Q (the new generation) of the same size N(the size of population). A new trial vector y is composed of the current point x_i of the old generation and a new point u obtained by using *mutation*. If $f(y) > f(x_i)$ (in case of fitness maximization), the point y is inserted, instead of x_i, into the new generation Q. After completion of the new generation Q, the old generation P is replaced by Q and the search continues until the stopping condition is reached[10]. In related work different variations exist for producing the mutation vector. In this work we use the *DE/rand/1/bin* and *DE/best/1/bin*, which generate the point u by adding the weighted difference of two points.

$$u = r_1 + F(r_2 - r_3). \tag{1}$$

Equation 1 shows how a new point u is created; r_1, r_2 and r_3 in the *DE/rand/1 /bin* are randomly selected from P. For *DE/best/1/bin*, r_1 is the best individual in the population, r_2 and r_3 are both randomly selected. As for the DE parameters, we have F, CR and N, which respectively represent the differential weight, F that takes the next values $F \in [0, 1]$; CR which in turn represents the crossover probability with possible values in $CR \in [0, 1]$; and the last parameter represents the population size N. A general description of the DE pseudocode is presented below (Algorithm 1). For our work we have selected Differential Evolution due to an easy parameter calibration, easy encoding, and a refined exploration search in the fitness landscape. In [26], the authors demonstrate the effectiveness of the algorithm establishing a comparison with other different evolutionary methods related to artificial neural networks. In some works like [28, 29] the DE had been

used in robotics for mapping and localization tasks, in [27] the DE was used to find a minimal representation for a multi-sensor fusion.

Algorithm 1 Differential Evolution Algorithm

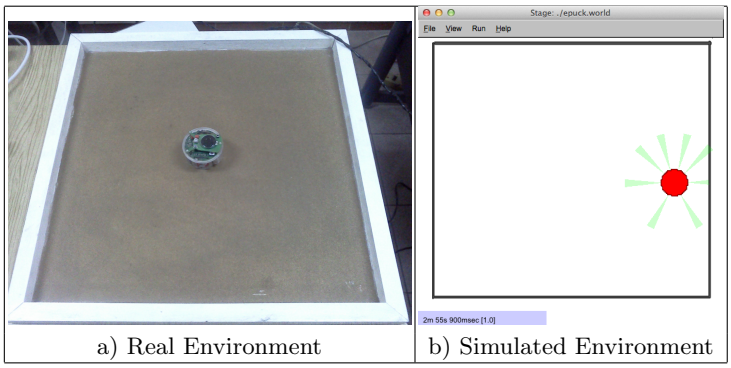
```
Initialize the population with random positions, P = (x1; x2; ...; xN)
repeat
    for each individual, i = 1 → N do
        select randomly r1, r2, r3 of P
        select randomly R ∈ 1, ..., n (n is the number of dimensions of the problem)
        for each position, j = 1 → n do
            select an uniformly distributed number rand_j ∈ U(0, 1)
            if rand_j < CR or j = R then
                generate a mutated position u_ij, being
                y_j = u_ij ← r1j + F(r2j − r3j)(Eq.1)
            else
                y_j = x_ij
            end if
        end for
        if f(y) > f(x_i) then
            insert y in Q
        else
            insert x_i in Q
        end if
    end for
    P = Q
until reaching the stop condition
```

The main problem of the GA methodology is the need of tuning a series of parameters for the different genetic operators like crossover or mutation, the decision of the selection (tournament, roulette,...), and tournament size. Hence, in a standard GA is difficult to control the balance between exploration and exploitation. On the contrary, DE reduces parameter tuning and provides an automatic balanced search. As Feoktistov [13] indicates, the fundamental idea of this algorithm is to adapt the step-length ($F(x_2 - x_3)$) intrinsically along the evolutionary process. For the initial generations the step-length is large because individuals are far away from each other. As evolution goes on, population converges and the step-length becomes smaller and smaller, providing an automatic search exploration level.

3 Robot Software Simulator

Player-Stage is a free software tool for robot and sensor applications[8]. Player provides a network interface to a variety of robot and sensor hardware such as: the Khepera, Pioneer, Lego mindstorm, E-puck, etc. Player's client/server model allows robot control programs to be written in any programming language and

to run in any computer with a network connection to the robot. Also it supports multiple concurrent client connections to devices, creating possibilities for distributed and collaborative sensing and control. On the other hand, Stage simulates a population of mobile robots moving and sensing in a two-dimensional bitmapped environment. Various sensor-models are provided, which includes a sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection, and an odometry sensor. It is important to notice that Stage devices offer a standard Player interface with little required changes for using the same code in simulation and hardware. For our experiments we implemented, in Player-Stage, a squared arena delimited by four walls which includes an e-puck robot, This simulated arena is similar in dimensions to a real arena set in our laboratory (see Fig.1).



a) Real Environment b) Simulated Environment

Fig. 1. a) The physical robot is set inside an arena delimited by four walls. b) The virtual scenario is modeled as a squared arena, in Player-Stage, where we set the e-puck robot (red circle) with eight proximity sensors (in green).

4 Robotic Experimental Setup

The robot controller consists of a neural controller fully connected (see Fig.2) with 8 input neurons that encode the activation states of the corresponding 8 infrared sensors. Values for activation are normalized between 0 and 1, where 0 means that an obstacle is detected nearby, and 1 represents no obstacle detection. Additionally, 4 internal neurons receive connections from sensory neurons and send connections to 2 motor neurons. The first motor neuron is used to define the linear velocity, whereas the second one defines the angular velocity. The output and the internal neurons use a sigmoid transfer function.

The robot controllers were evolved, using DE and GA as the evolutionary approaches, for automatically obtain the wall-following behavior. Based on the work of Mezura et ál.[11], for solving a high-dimensionality problem, we chosen

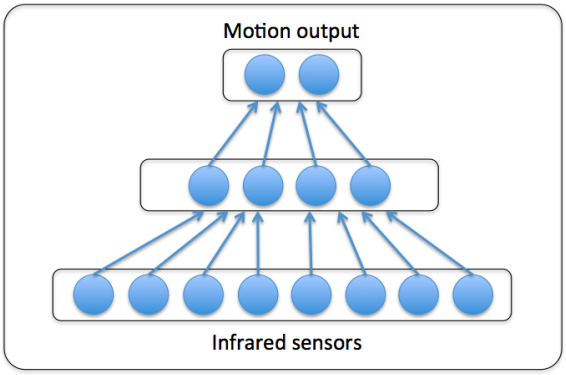


Fig. 2. The Neural Network used as robot controller for the e-puck robot, composed by 8 neurons in the input layer for the infrared sensors, 4 neurons in the hidden layer and the output layer made of 2 neurons for controlling robot-motion (notice that not all connections are shown).

the next DE variants: *DE/rand/1/bin* and *DE/best/1/bin*, with the following parameters: *DE/rand/1/bin* with F=0.7, CR=1.0, N=120; and the *DE/best/1/bin* with: F=0.8, CR=1.0, N=150. For the GA, the parameters, based on [16], are: *Population size* = 100, *Crossover rate* = 85%, *Mutation rate* = 1%, *Tournament size* = 5.

For our experiments we evolved neural controllers in the Player-Stage simulator. The evolutionary process is computed by the GA and the DE with the above variants and parameters. The algorithm is left to run until an optimal level is reached. The fitness function looks for those individuals able to locate and follow a wall without crashing. The fitness function is defined in equation (2). Fitness is averaged during 5 different trials for an individual tested with initial random positions and orientations. The average fitness is scored as the individual fitness.

$$f = \sum_{i=0}^{n} [(linearVelocity) * (1 - abs(angularVelocity) * (1 - min_i r))] \quad (2)$$

where: $linearVelocity \in \{0, 1\}$, 0 represents the minimum value of velocity and 1 its maximum value; $angularVelocity \in \{-1, +1\}$, -1 represents a left-turn, +1 a right-turn and 0 forward-movement; $min\_ir \in \{0, 1\}$ is the minimum reading of the 8 infrared sensors. Infrared sensors take values of $\in \{0, 1\}$, where 1 represents that no-obstacle is detected; in contrast 0 is obtained when an object is detected in the vicinity of the robot body. When a robot crashes, calculation of the fitness is stopped and its current value is returned as the scored fitness.

The fitness formula is set to maximize linear velocity while minimizing both the angular velocity and robot-distance to the wall. In other words, the fitness

function looks for moving forward quickly and running parallel to walls. In summary, a neuro-controller was evolved in order to develop wall-following behavior using the GA and two DE variants.

5 Results

In order to verify the quality of the evolved solutions, experiments are replicated at least three times due to the expensive time execution. The best results are obtained with the *ED/best/1/bin*, which reaches a maximum fitness of 246.66 (see Fig.5), while the *ED/rand/1/bin* reaches a maximum fitness of 215.217 (see Fig.4) and the GA reaches 202.775 (see Fig.3). For all cases each individual is evaluated 5 times with random positions and orientations. In general, the *ED/best/1/bin* makes a better exploitation of candidate neural-controllers in the fitness landscape than *ED/rand/1/bin* and GA.

Figures 3, 4 and 5 denote the evolution of the average quality of the population and the evolution of the quality of the best individual. In the case of the *ED/best/1/bin* and *ED/rand/1/bin* alternative, the average quality approaches progressively to the best quality through generations, though maintaining an adequate diversity in the population able to obtain better solutions. On the contrary, using the GA, the two curves of quality evolutions are very far from each other. As a consequence individuals are not appropriately approaching the finest solutions, since exploration dominates completely in DE searching.

In total, evolution was carried out for 30,000 evaluations. After evolution is stopped the wall-following behavior can be described as follows: robot is located in the arena, first it starts by looking for the nearest wall, when the wall is found, the e-puck follows the wall while it avoids crashing into walls (see Fig.6 and Fig.7).

6 Discussion and Conclusions

In ER, the most common approach relies on the use of GA and Neural Networks for evolving robot controllers. In this work we evolve neuro-controllers using two DE variants and GA. The work on this paper shows the results of using DE and GA to evolve a wall-following behavior, using the Player-Stage simulator, which facilitated testing individuals during evolution. The resultant behavior is equivalent for both the real and the simulated e-puck robot. Here, we have shown that the use of DE produces better fine candidate neuro-controller solutions for robot behavior than the GA. DE offers easy parameter calibration, fast implementation, fine exploration/exploitation of candidate solutions in the fitness landscape, and also production of mature individuals during the initial generations of the evolutionary process.

As future work, we propose to combine or hybridize DE with local search methods, like supervised and reinforcement learning in the neural network controllers. This will allow the integration of the advantages of the two search meth-

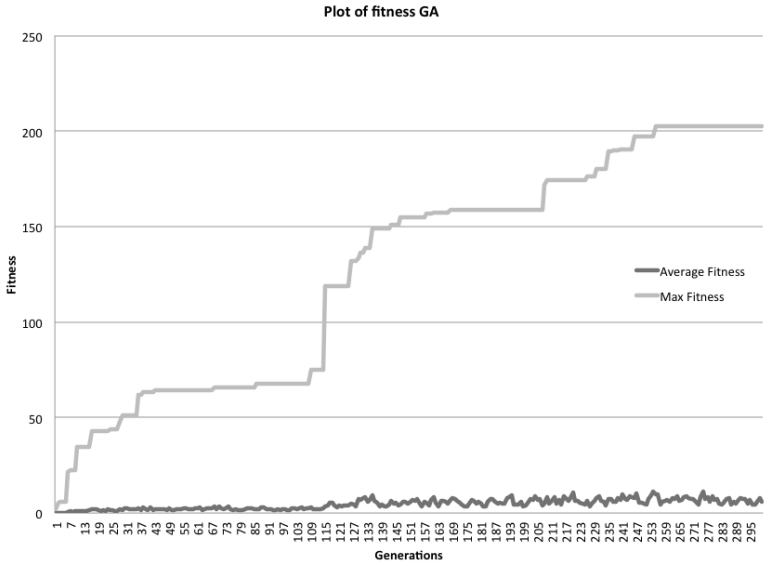


Fig. 3. Maximum and average fitness is plotted across 300 generations using the GA.

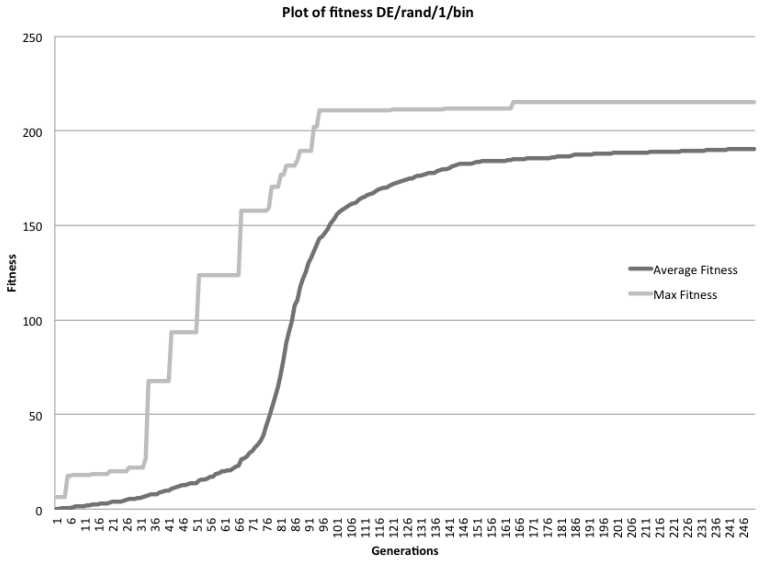


Fig. 4. Maximum and average fitness is plotted across 250 generations using the *ED/rand/1/bin* method.

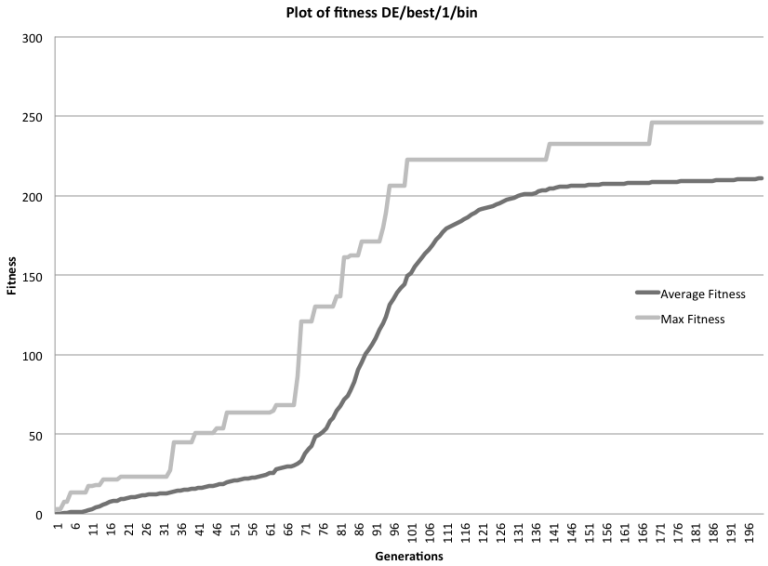


Fig. 5. Maximum and average fitness is plotted across 250 generations using the *ED/best/1/bin* method.

Fig. 6. Robot behavior in the real e-puck robot. The robot starts searching for a wall, when a wall is found, the epuck follows the wall while avoids crashing into walls.

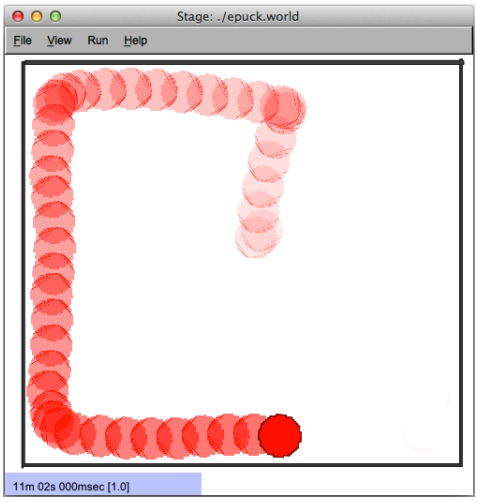


Fig. 7. The Player-Stage simulated e-puck robot for the best individual of evolution using the *ED/best/1/bin* method. A wall-following behavior is exhibited by the best individual.

ods: the global search provided by the DE method with the fast local search of the neural learning methods.

Acknowledgments. This work has been sponsored by CONACYT-MEXICO grant SEP No. 0100895 and by the Ministry of Science and Innovation of Spain (project TIN2011-27294).

References

1. Mondana F., Bonani M., The e-puck education robot, `http://www.e-puck.org/`
2. Trefzer, M., Kuyucu, T., Miller, J. F. and Tyrrell, A. M.: Evolution and Analysis of a Robot Controller Based on a Gene Regulatory Network. In: Proceedings of ICES, pp. 61–72 (2010)
3. S. Nolfi, D. Floreano, Evolutionary Robotics, The Biology, Intelligence, and Technology of Self-Organizing Machines, The MIT Press, Cambridge Massachusetts (2000).
4. Santos, J. & Duro, R.: Artificial Evolution and Autonomous Robotics (in Spanish). Ra–Ma Editorial, Spain (2004)
5. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
6. D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989.
7. Doncieux, S., Mouret, J.: Behavioral diversity measures for Evolutionary Robotics. In: IEEE Congress on Evolutionary Computation (2010)
8. The Player Project. Free Software tools for robot and sensor applications, `http://playerstage.sourceforge.net/`

9. Rainer Storn and Kenneth Price. Differential Evolution a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. J. of Global Optimization vol. 11, no. 4, pages 341-359(1997).
10. Price, Kenneth V. An introduction to differential evolution. In New ideas in optimization, David Corne, Marco Dorigo, Fred Glover, Dipankar Dasgupta, Pablo Moscato, Riccardo Poli, and Kenneth V. Price (Eds.). McGraw-Hill Ltd., UK, Maidenhead, UK, England, pages79-108 (1999).
11. Efrén Mezura-Montes, Mariana Edith Miranda-Varela, Rubí del Carmen Gómez-Ramón, Differential evolution in constrained numerical optimization: An empirical study, Information Sciences, Volume 180, Issue 22, 15 November 2010, pages 4223-4262
12. Rechenberg I. "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution." Dr.-Ing. Thesis, Technical University of Berlin, Department (1971) of Process Engineering
13. V. Feoktistov, Differential Evolution: In Search of Solutions, Springer (2006)
14. Cliff D. T., I. Harvey and P. Husbands. Explorations in Evolutionary Robotics. Adaptive Behavior 2, 73-110 (1993).
15. Stefano Nolfi, Evolving non-trivial behaviors on real robots: A garbage collecting robot, Robotics and Autonomous Systems, Volume 22, pages 187-198 (1997).
16. Montes-Gonzalez, F., Aldana-Franco, F. The Evolution of Signal Communication for the e-puck Robot. Advances in Artificial Intelligence, pages 466-477 (2011).
17. Koza, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press (1992).
18. Christopher L., Huosheng H. Using Genetic Programming to Evolve Robot Behaviours. Proceedings of the 3rd British Conference on Autonomous Mobile Robotics & Autonomous Systems (2001).
19. Kodjabachian J., Meyer J., Ecole A., Superieure France. Evolution and Development of Neural Networks Controlling Locomotion, Gradient-Following, and Obstacle-Avoidance in Artificial Insects (1997).
20. Mondada F., Floreano D. Evolution of Neural Control Structures: Some Experiments on Mobile Robots. Robotics and Autonomous Systems, vol. 16, pages 183-195 (1995).
21. Floreano D., Mondada F, Evolution of homing navigation in a real mobile robot. IEEE Transactions on Systems, Man,Cybernetics Part B: Cybernetics 26 pages 396-407 (1996).
22. A.L. Nelson, E. Grant, J.M. Galeotti, S. Rhody, Maze exploration behaviors using an integrated evolutionary robotics environment. Journal of Robotics and Autonomous Systems, vol. 46, no. 3, pages 159-173 (2004).
23. Nolfi S., Floreano D., Miglino O., Mondada F. How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics. R.A. Brooks, P. Maes eds., Proceedings of the IV International Workshop on Artificial Life, Cambridge, MA, MIT Press (1994).
24. M. Quinn, L. Smith, G. Mayley, P. Husbands, Evolving team behavior for real robots, in EPSRC/BBSRC International Workshop on Biologically-Inspired Robotics: The Legacy of W. Grey Walter (WGW '02), Aug. 14-16, 2002
25. Nolfi S. Evolving non-trivial behavior on autonomous robots: Adaptation is more powerful than decomposition and integration. In T.Gomi (Ed.), Evolutionary Robotics, Ontario (Canada): AAI Books, 21-48 (1997).
26. A. Slowik, M. Bialko, Training of artificial neural networks using differential evolution algorithm, in Proc. IEEE Conf. Human Syst. Interaction, Cracow, Poland, pages 60-65 (2008).

27. Rajive Joshi, Arthur C. Sanderson. Minimal representation multisensor fusion using differential evolution. IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI , vol. 29, no. 1, pages 63-76 (1999).
28. Luis Moreno, Santiago Garrido, Fernando Martín and M. Luisa Munoz. Differential Evolution approach to the grid-based Localization and Mapping problem. Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference, pages 3479-3484 (2007).
29. Ali R. Vahdat, Naser NourAshrafoddin, Saeed S. Ghidary. Mobile robot global localization using differential evolution and particle swarm optimization. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC) , pages 1527-1534 (2007).