Neural Network Controller against Environment: A Coevolutive approach to Generalize Robot Navigation Behavior

A. BERLANGA, A. SANCHIS, P. ISASI and J. M. MOLINA
ScaLab, Universidad Carlos III, Avda Universidad, 30, 28911, Madrid, Spain;
e-mail: aberlan@ia.uc3m.es, masm@inf.uc3m.es, isasi@ia.uc3m.es, molina@ia.uc3m.es

(Received: 5 December 2000; in final form: 3 April 2001)

Abstract. In this paper, a new coevolutive method, called Uniform Coevolution, is introduced to learn weights of a neural network controller in autonomous robots. An evolutionary strategy is used to learn high-performance reactive behavior for navigation and collisions avoidance. The introduction of coevolutive over evolutionary strategies allows evolving the environment, to learn a general behavior able to solve the problem in different environments. Using a traditional evolutionary strategy method, without coevolution, the learning process obtains a specialized behavior. All the behaviors obtained, with/without coevolution have been tested in a set of environments and the capability of generalization is shown for each learned behavior. A simulator based on a mini-robot Khepera has been used to learn each behavior. The results show that Uniform Coevolution obtains better generalized solutions to examples-based problems.

Key words: robot navigation problem, generalized behavior, competitive coevolution, learning examples-based, evolutionary strategies.

1. Introduction

Many robotic systems applied in industry are autonomous mobile robots working in stationary environments, i.e. automatic floor-cleaning, automatic assembly, transporting pieces in a factory, etc. Other applications of robotic systems involve interactions with dynamic environments, where the autonomous robot deals with unexpected events, such as tour-guiding robots. The successful operation in such environments depends on the ability of adaptation to the changes. Thus, for most agent-based tasks, having a perfect domain theory (model) of how the actions of the agent affect the environment is usually an ideal. Three ways of providing such models to agents (planners/controllers) are considered:

1. High level planning, learning and control. One of the first approaches to planning, learning, and executing within autonomous robotic tasks was the Shakey robot [49]. It had a planner, and a learning method based on compacting the solutions (plans) to given problems into new operators that could be used in future planning steps, as well as means for replanning in case things went

wrong. However, it was soon discovered that approaches based on the classical paradigms (abstraction, planning, heuristic search, etc.) were not completely suitable for unpredictable and dynamic environments.

2. Manual design of reactive planners. Other approaches consider reaction as the new paradigm to build intelligent systems. One classical instance of this kind of architecture is the subsumption architecture, which was proposed by Brooks and has been successfully implemented on several robots. The base of the subsumption architecture is a piece of code called "behavior" or "skill". Each behavior produces an action (reacts) in a given situation, and the global control of the "planning" system is a composition of behaviors. Different systems, from finite state machines to fuzzy controllers, have been used for the implementation of these behaviors. In most cases, the way in which these behaviors were built was by careful and painstaking "ad-hoc" manual design of skills by a human.
3. Automatically acquiring skills for reactive planners. Another approach that solves the above mentioned disadvantages of manual design consists on automatically learning those behaviors. There have been already many different approaches for learning skills in robotic tasks, such as neural networks [39], genetic techniques [14, 34, 48], evolutionary strategies for configuring neural networks [4, 31], inductive collaborative techniques [19], or reinforcement learning techniques [32, 53].

Some of the Machine learning techniques becomes inapplicable to the learning reactive behavior problem because they require more information than the problem constraints allow. In the last few years, new approaches that involve a form of simulated evolution have been proposed in order to build autonomous robots that can perform useful tasks in unstructured environments [9, 10, 33]. The main interest in this approach is due to the dissatisfaction with traditional robotic and Artificial Intelligence approaches, and their belief that interesting robots may be too difficult to design. Thus, it would appear reasonable to use an automatic procedure, such a genetic algorithm that gradually builds up the control system of an autonomous agent by exploiting the variations in the interactions between the environment and the agent itself. It remains to be determined if it is feasible.

In particular, two questions should be answered: what to evolve? And how to evolve it? The choice of what to evolve is controversial. Some authors have proposed to evolve controllers in the form of explicit programs in some high-level language. Brooks [8] proposes to use an extension of Koza's genetic programming technique [27]. Other authors propose to evolve controller rules using a form of classifier system [12–14, 48], or using a fuzzy controller [26, 34]. Finally, others authors propose to evolve weights in neural network controllers, fixing the architecture [3–5, 16, 36].

When a system is learning from examples, for testing a solution is necessary to face it with different situations (examples set). Moreover, in many cases, the solutions should not fit a particular examples set, they have to be generalized solu-

tions, useful over any possible example. This problem becomes harder when using evolutionary methods. An excessive adaptation to the examples set could abort the generalization capability of a solution. The evolution of the examples tries only to generate harder examples for the solutions. In this paper, a new method is proposed based on Hillis's ideas [24] to use coevolution ideas to learn generalized autonomous robot navigation behaviors. The new proposed coevolutive method considers the coevolutive approach from a different perspective of the previous coevolutive works. Previously, the theory of coevolution has been applied to solve specific robotic problems, basically the pursuit-evader problem. These approaches are a particularization of the general theory and the methods could not be translated directly to other robotics domains. In our proposal, the Uniform Coevolution method is a general framework for any robotic problem that could be defined by means of a set of possible solutions and a set of examples (navigation, transformation, assembly, etc.).

In Section 2, the general theory of coevolutive method is outlined. Section 3 is related to the experimental environment and the goals of the work, in Section 4 the Uniform Coevolution method has been described. The experimental results are shown in Section 5. The last section contains some concluding remarks.

2. Coevolution in Robotics

Coevolution refers to the simultaneous evaluation of several species, where the survival of each specie depends on one each other. When talking about coevolution in computational terms, this is referring to the ability of a system to improve its performance by means of mutual adaptation of its different constituents. The final performance of the system is improved as a consequence of the incremental adaptation among constituents. These ideas of coevolution were first explored in evolutionary computation in some works in the Iterated Prisioner's Dilemma [1, 2, 30]. One of the first authors in applying the coevolution in an optimization problem was Hillis with his work over the coevolution of parasites for improving solutions in a sorting network problem [24]. More recently, some works for establishing the theoretical basis in coevolution have been done [41] and [46]. All these previous works have proven the usefulness of coevolution to improve the evolutionary computation techniques from different perspectives.

Problems related with robotics have been one of the main fields of application of evolutive computation. A wide variety of robotic controllers, to resolve specific tasks, have been investigated; robot planning [23], wall following task [28], collision avoidance [50], arm control [29], etc. The traditional evolutive computation techniques have several disadvantages. Coevolution has been proposed as a way to evolve a learner and a learning environment simultaneously such that open-ended progress arises naturally, via a competitive arms race, with minimal inductive bias [15]. The viability of an arms race relies on the sustained learnability [43] of environments. The capability to obtain the ideal learner, the better environ-

ments where the learning takes place, is the main advantage of the coevolutive method.

There are two main reasons to apply a competing coevolutive approach in robotics: that does not require specification of complex fitness functions and the intrinsic complex dynamics of an elegantly simple architecture. Several authors have been used competitive coevolution in robotics. Floreano and Nolfi [17] have studied the feasibility of co-evolutionary pursuit-evasion for evolving useful neurocontrollers for two Khepera robots. They have observed spontaneous evolution of obstacle avoidance, visual tracking and other navigation skills without any effort in fitness design. Cliff and Miller [11] have developed a coevolution simulator to study coevolutionary arms race with robot in a pursuit-evasion role. They conclude that coevolution evolves quickly smart pursuers and evaders, unpredictable behavior will emerge quickly, robustly and permanently in any simulation and mainly the details of fitness function, trial initialization methods and genetic algorithm parameters will not matter much because coevolution is so robust.

Those applications have not significant differences between the learner agent and the environment. To apply a competitive coevolutive method to robot navigation problem a new generalized framework is needed.

3. Robot Controller and Experimental Environment

The concept of designing new computational structures using all the available information from the very efficient world of biology, specially the case of neural networks is a fascinating approach [18, 7]. The most interesting feature of this approach is that biological neural networks have been designed to be adapted to the real world [22, 40]. This suggests the biological approach as the most appropriate to solve problems existing in mobile robotics field [20, 37]. The redundancy, the parallelism and the resulting robustness of the computational structure are the interesting points of the biological approach. These mechanisms are not only exploited inside the brain structure, but also at the level of groups of animals, giving very interesting results in collective behavior of insects, for instance.

The control architecture used to evolve the reaction (adaptation) is based on a neural network. The neural network controller has several advantages [36]:

- Neural Networks, NN, are resistant to noise, those exist in real environment, and are able to generalize their ability in new situations.
- The primitives manipulated by the Evolutionary Strategy, ES, are at the lowest level in order to avoid undesirable choices made by the human designer.
- A NN could easily exploit several ways of learning during its lifetime.

The use of a feed forward network with eight input units and two output units directly connected to motors appears in previous works [36] as an efficient way to learn a behavior: “avoid obstacles” using genetic algorithms. In this work, the NN ought to learn more complex behavior: “navigation”. Another successful approach to learning a robotic skill by using neural networks has been the Ralph and

Alvinn systems [42]. In this case, the learning task was to learn how to drive an autonomous vehicle in a highway.

In this work, the task faced by the autonomous robot is to reach a goal in a complex environment while avoiding obstacles found in its path. In the proposed model, the robot starts without information about the right associations between environmental signals and actions responding to those signals. The number of inputs (robot sensors), the range of the sensors, the number of outputs (number of robot motors) and its description is the only previous information. From the initial situation the robot is able to learn through experience the optimal associations between inputs and outputs. Different environments have been used to find the connections of the neural network.

3.1. ENVIRONMENT

In this work, a simulator based on an autonomous robot named Khepera [39] is used. The mini-robot Khepera is a commercial robot developed at LAMI (EPFL, Laussanne, Switzerland). The robot characteristics are 5.5 cm of diameter in circular shape, 3 cm of height and 70 g of weight. The sensory inputs come in from eight infrared proximity sensors. These sensors are composed of two devices: an IR emitter and a receiver. The emitter and the receiver are independent, then it is possible to use the receiver to measure the reflected light (with the emitter active) or to measure the environmental light (without emission). The reflected light measurement can give some information about the obstacles. In fact, this measure is not only a function of the distance to an object in front of the emitter but also the environmental light and the object nature (color and texture). So the value of distance is modified by the measure of the ambient light and the object nature, the light used is constant and all the obstacles used have the same color and texture. The robot has two wheels controlled by two independent DC motors with incremental encoder that allow any type of movement. A speedometer could read each wheel velocity.

Using the ambient sensors it is possible to measure the distance and the angle to a light source. The distribution of the amount of light coming into the eight sensors is used to evaluate the distance and the angle to the source, see Figure 1. The amount of light received in the sensor depends on the distance of the light source. The response curve of each real sensor is described by a sigmoid function [39].

Experiments take a long time of continuous functioning of the hardware. In order to prove the different configurations of the controllers, a simulator developed in a previous work [51] have been used, the SimDAI one. In the simulator, the characteristics of the turtle robot model [35] and the physical restrictions of the Khepera robot have been considered. The proximity sensors (in front) are grouped in three sets, see Figure 1. SimDAI is a working prototype of a mobile robot simulation environment for experimenting with robot navigation and control algorithms. Each mobile robot is completely independent, can navigate and interacts with other

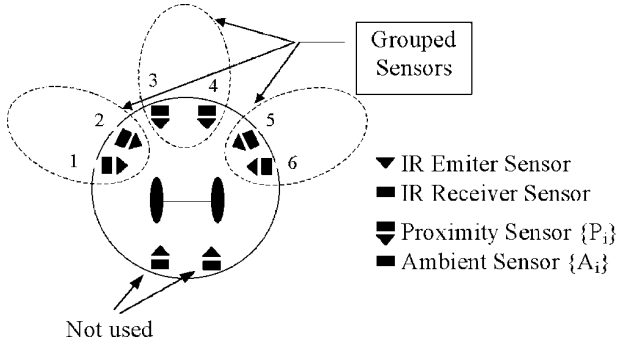
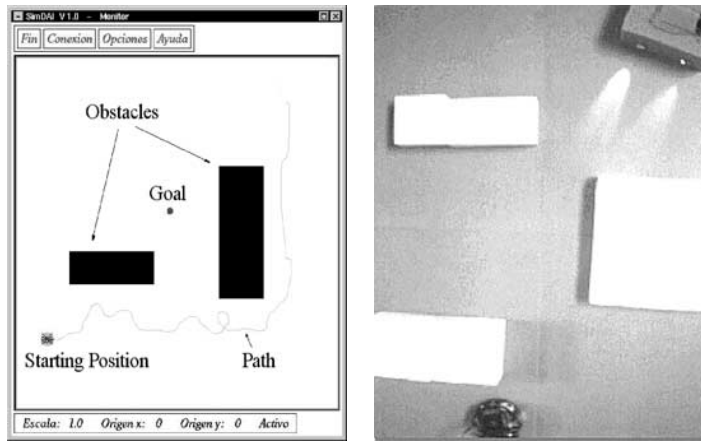


Figure 1. Sensors considered in the real robot.



(a) (b)

Figure 2. (a) SimDAI Simulator (Example of one simulated environment). (b) Example of a real experimental environment.

robots in a 2-D simulated world of obstacles, which is separately monitored. This simulator has been used in many other works [25, 34, 38, 47]. The simulation world consists of a rectangular map of user defined dimensions where particular objects are located. In this world it is possible to define a final position for the robot. In this case the robot is represented with three proximity sensors and two special sensors to measure the distance and the angle to the goal (Figures 2(a) and (b)).

3.2. ROBOT CONTROLLER

It has been proven that by means of connections between sensors and actuators, a controller is able to solve any autonomous navigation robotic behavior [7]. This theoretical approach is based on the possibility of finding the right connections of a feed-forward NN without hidden layers for each particular problem, see Figure 3.

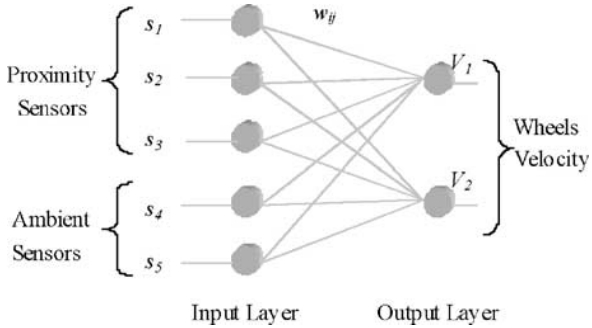


Figure 3. Neural network architecture.

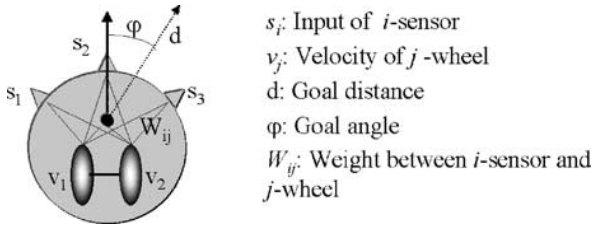


Figure 4. Connections between sensors and actuators in the Braitenberg representation of a Khepera robot.

The input sensors, s_i, considered in this approach, are calculated from the ambient and proximity sensors, (P_i, A_i), in this way: $s_1 = P_1 + P_2, s_2 = P_3 + P_4, s_3 = P_5 + P_6, s_4 =$Max{Light received in A_i}, $s_5 =$ Angle of the light received in $\{A_i\}$. The NN outputs are the wheel velocities. The NN architecture is shown in Figure 3.

The velocity of each wheel is calculated by means of a linear combination of the sensor values, Equation (1), using those weights (Figures 3 and 4):

$$v_j = f\left(\sum_{i=1}^{5} w_{ij} s_i\right), \tag{1}$$

where w_{ij} are searched weights, s_i are sensor input values and f is a function for constraining the maximum velocity values of the wheels.

Weight values depend on problem features. To find them automatically, an evolutionary strategy, ES is used [6]. In this approach each individual is composed of a 20-dimensional real-valued vector, representing each one of the above mentioned weighs and their corresponding variances. The individual represents one robot behavior consequence of applying the weights to Equation (1). The evaluation of behaviors is used as fitness function.

Evolution strategies (ES) developed by Rechenberg [44, 45] and Schwefel [52], have been traditionally used for optimization problems with real-valued vector representations. As Genetic Algorithms, GA, [21], and the ES are heuristic search techniques based on the building blocks hypothesis. Unlike GA, however, the search

is basically focused in the gene mutation. This is an adaptive mutation based on the likely the individual represents the problem solution. The recombination plays also an important role in the search, mainly in the adaptive mutation.

4. Uniform Coevolution Applied to the Robot Navigation Problem

In this work, a coevolutive approach called Uniform Coevolution (UC) is presented. The architecture of the UC is composed by a population of solutions and a set of populations of examples (one population of examples for each individual in the population of solutions), see Figure 5.

The solutions and examples systems are described above:

- Solutions Generator System (SGS). A population of solution individuals (SI_i) composes it. For computing their fitness, is necessary to face each individual with a set of different situations, examples, represented by a population in the examples generator system (further explained). The objective of this system is to gradually generate better solutions to a particular problem. Any evolutionary computation method can be used, where an individual represents one problem solution. The evolution of the SGS follows the dynamics of the evolutionary computation method selected.
- Examples Generator System (EGS). The population of EGS is composed of evaluation (training) examples. This population is arranged in a special way. The examples are grouped in small blocks. Thus the EGS is a population of blocks of training examples. Every individual of SGS has a set of blocks related only with it. The evaluation of the individual of SGS is calculated over its related m-block, each composed with r evaluation examples. Thus, the evaluation is performed over $m \times r$ evaluation examples. This general

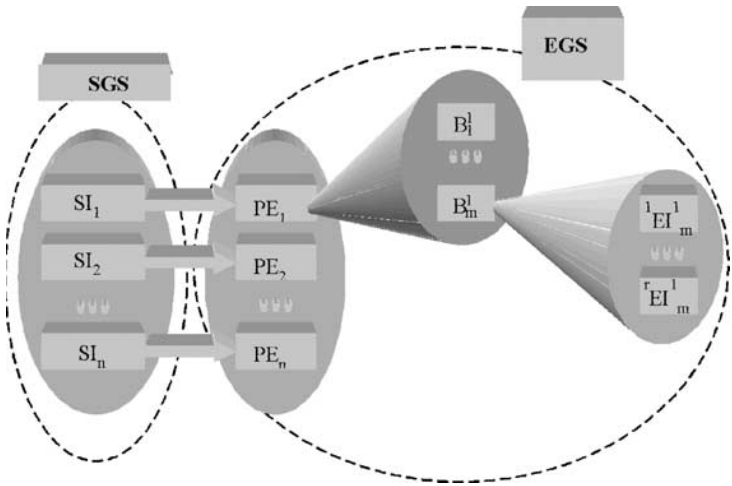


Figure 5. Uniform coevolutive architecture.

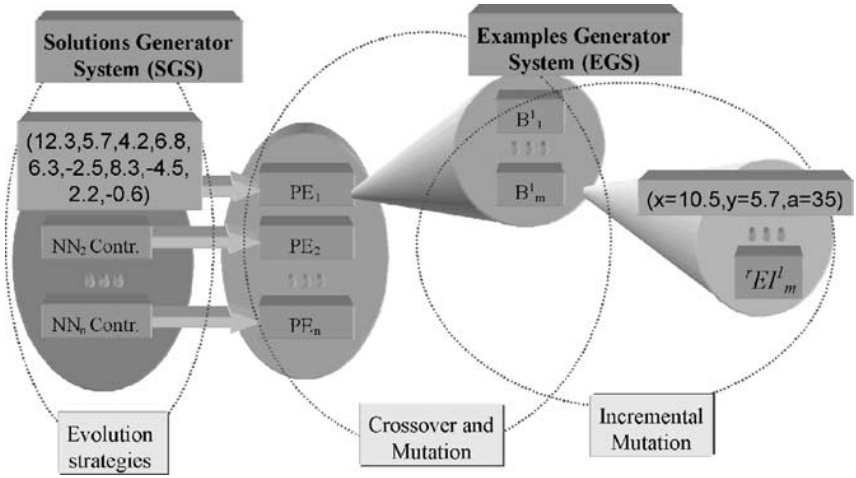


Figure 6. Uniform Coevolution applied to robot navigation problem.

scheme is shown in Figure 5. In order to make more comprehensible the blocks structure, the different blocks have been called as following:

- ${}^{k}EI_{j}^{i}$: Is the evaluation k-example of the j-block related with the i-individual of SGS.
- B_j^i: Is the j-block related with the i-individual of SGS.
- PE_i: The set of all B_j^i related with the i-individual of SGS.

In the robot navigation problem the SGS is the population of controllers (each SI_i represents the weights of a neural network) and EGS is composed with different training examples or environments (each ${}^{k}EI_{j}^{i}$ represents a robot starting angle and position). In order to explore the solutions space, an evolutive strategy is applied to the SGS. Following the UC scheme, each solution of SGS has a given set of evaluation example associated. In Figure 6, a short scheme of the system and the evolutive algorithms applied is shown.

The UC method evolves automatically solutions and examples. The evolution of each system depends on the other's evolution. The general procedure is as follows:

1. Initialization of the populations:
   (a) SGS initialization (n-SI individuals)
   (b) EGS initialization (n-PE of m-B blocks with r-EI training examples)
2. Computation of the fitness
   (a) Evaluation of each SI_i over each individual ${}^{k}EI_{j}^{i}$ in its related PE_i
   (b) The fitness of each SI_i is a combination of the above evaluations, this calculation is explained in more detail in Section 4.1.
   (c) The Fitness of each PE_i is the fitness value of the correspondent SI_i
3. Generation of new populations
   (a) Selection of individuals from SGS.

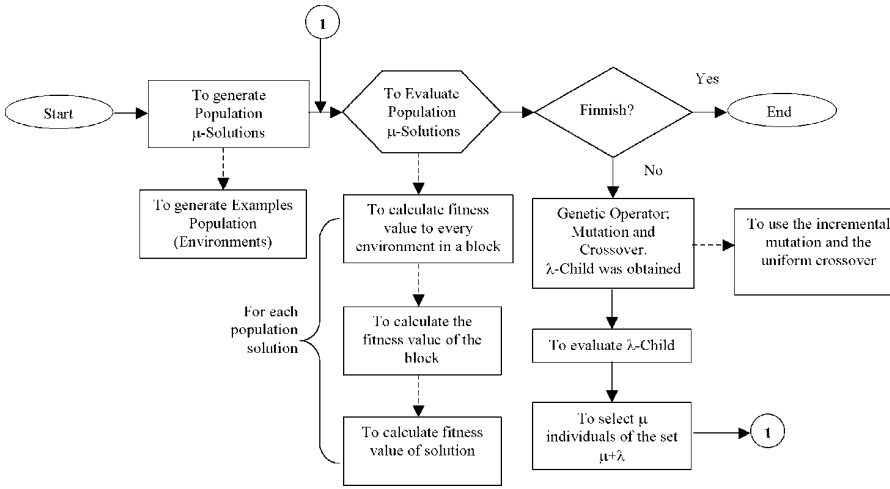


Figure 7. Uniform Coevolution scheme to solve the robot navigation problem.

(b) New SGS population, in this problem a $(6+4)$-evolutionary strategy, was applied to evolve neural network controllers.

(c) The evolution of EGS is related with the generation of new *EI*'s for the new solutions. Two ad-hoc genetic operators: the Incremental Genetic Operator (IGO), and a modified uniform crossover operator are applied, both operators will be explained below.

A brief scheme of the process is shown in Figure 7.

The evolutive process involves the generation of two new populations with one evaluation of the fitness value. Thus, the first step is to generate a new population for both systems. The selection of a SGS individual following the classical rules of $(\mu + \lambda)$ evolutionary strategy. In Figure 8 the individual SI_i and SI_k has been selected to mate.

The second step is the recombination of the genotype of selected solution (SI_i and SI_k). This operation generates two new solution SI'_i and SI'_k (see Figure 9). The recombination of SGS individual produces the recombination of the first evaluation example of its related blocks. In Figure 9 the uniform recombination of the first examples is shown. In this recombination the genetic code does not change just the individual of EGS interchanges the first evaluation examples.

The third step is the generation of new blocks of evaluation examples. The IGO is applied to the first example to obtain a new one, applying successively IGO to the last example generated produces a new one until the block is completed. This process is shown in Figure 10. The IGO uses the genotype of evaluation examples and the fitness value calculated for this block. The IGO will be explained in more detail in Section 4.2. These steps are repeated until a new population of SGS has been obtained, then in addition, a new EGS has been generated too.

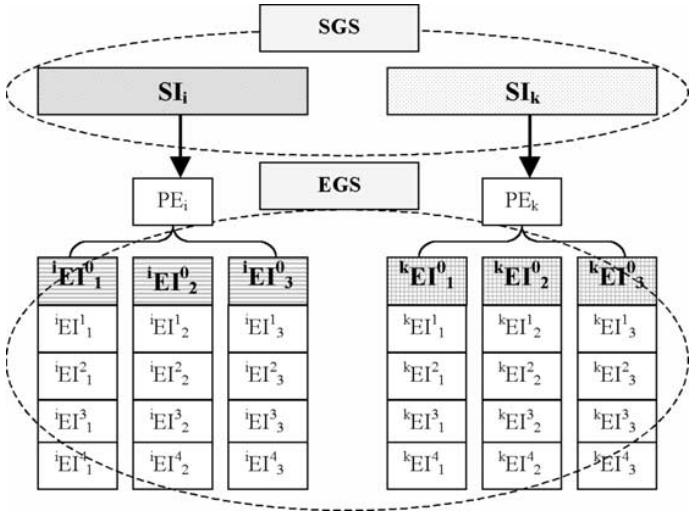


Figure 8. Generation of new population in SGS and EGS.

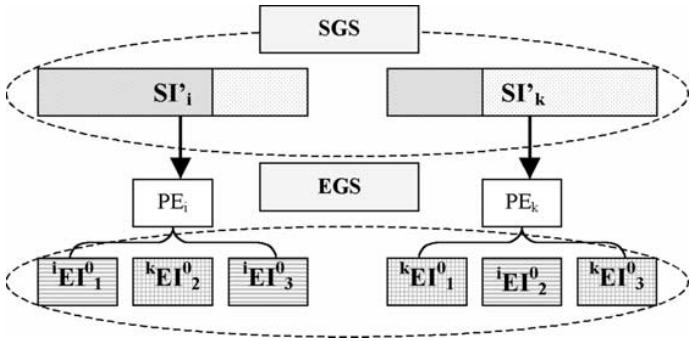


Figure 9. Recombination of SGS and EGS individuals.

4.1. FITNESS VALUE CALCULATION

The Uniform Coevolutive method requires some measures in order to calculate the final fitness value of every solution chromosome. In the following, the detailed calculation of the final fitness value is given.

The calculation of fit_i^j, Equation (2), is obtained using an evaluation function, Ψ, the genotype of the solution system, *ssol*, and the i-example of the j-block of EGSs individual associated with it are the evaluation function parameters. In Section 5.1 the evaluation function is described in more detail.

$$\Psi\left(ssol, seje_i^j\right) = fit_i^j. \tag{2}$$

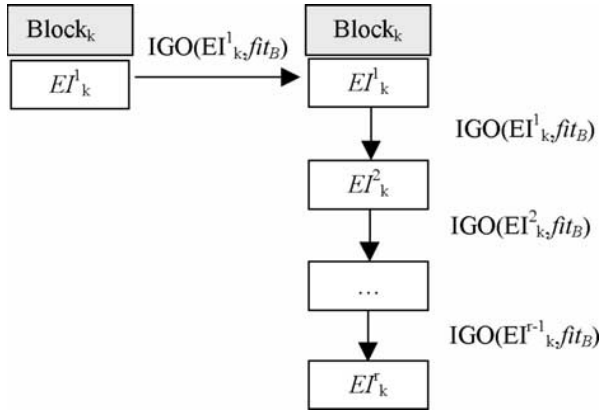


Figure 10. Generation of a new block of evaluation examples.

This measure, fit_i^j, is the classically fitness value of the evolutive computation techniques. To get fit_{SOL}, some previous measures are needed. First, the fitness value of all block of evaluation examples associated with it, fit_B^j, Equations (3)–(7) obtain the block fitness value.

Let associate each solution with m blocks, each one of them made with r learning examples. The first calculated parameter is α^j (calculated to j-block). This parameter is an absolute measure, related with how close, in mean, the solution proposed by SGS solves the proposed problem.

$$\alpha^j = 1 - \frac{\overline{fit^j} - F_{\min}}{F_{\max} - F_{\min}}, \quad \text{where} \quad \overline{fit^j} = \frac{\sum_{i=1}^{r} fit_i^j}{r} \tag{3}$$

The highest and the lowest fitness value a solution can ever reach are $F_{\max}$ and $F_{\min}$. It is not necessary to give exactly the extreme of the range values. The α^j parameter value is inside the interval [0,1]. Figure 11 shows the values of parameter α^j from the fitness value $\bar{f}$.

The second parameter is β_i^j (calculated over each evaluation example). This parameter is a relative reference. It measure is referred to the highest and lowest values of the fitness value in the set of evaluation examples to which it belongs, see Equation (4).

$$\begin{cases} fit_{\max}^j = fit_{\min}^j, & \beta_i^j = 0, \\ fit_{\max}^j \neq fit_{\min}^j, & \beta_i^j = \dfrac{fit_i^j - fit_{\max}^j}{fit_{\min}^j - fit_{\max}^j}. \end{cases} \tag{4}$$

The $fit_{\max}^j$ and $fit_{\min}^j$ values are the extremes fitness values obtained by solutions over their set of evaluation examples. Therefore, each set of examples has a $fit_{\max}^j$, the highest value and a $fit_{\min}^j$ the lowest fitness value obtained. The parameter β_i^j is calculated for every example of a set of evaluation examples. It measures how

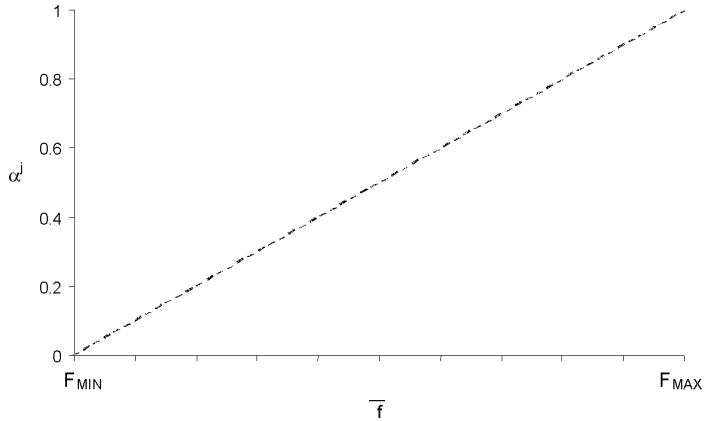


Figure 11. Plot of the parameter α^j.

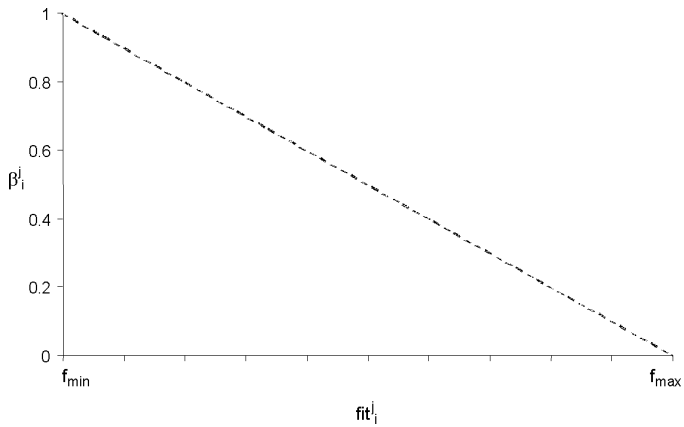


Figure 12. Plot of parameter β_i^j.

good the behavior of the solution over an example has been, related with other of its own set of evaluation examples. The possible values of parameter β_i^j are in the range [0,1]. Figure 12 shows the shape of parameter β_i^j.

With parameters α^j and β_i^j the parameter w_i^j for each evaluation examples of each set associated with the i-solution is calculated (see Equation (5)). This parameter w_i^j takes values in the interval [0,1]. The normalization of this parameter produces the γ_i^j parameter. This parameter has a fundamental importance in the UC method and has been named as Selection Pressure Control (SPC), see Equation (6).

$$w_i^j = \frac{(e^{\alpha^j} - 1)(e^{\beta_i^j} - 1) + (e^{1-\alpha^j} - 1)(e^{1-\beta_i^j} - 1)}{(e - 1)^2}, \tag{5}$$

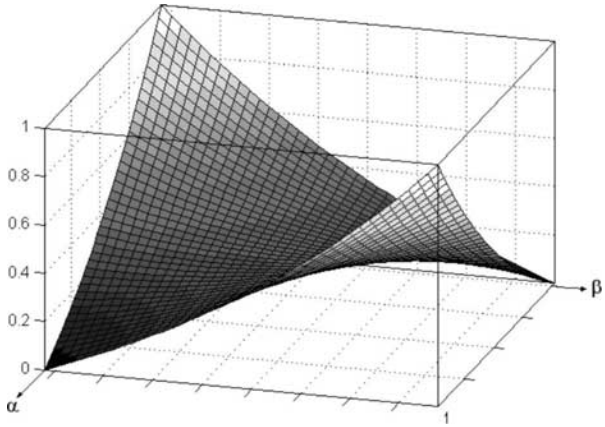


Figure 13. Plot of parameter w_i^j.

$$\gamma_i^j = \frac{w_i^j}{\sum_{k=1}^{r} w_k^j}. \tag{6}$$

At this point, the fitness value of the set of evaluation examples, fit_B^j, can be calculated as:

$$fit_B^j = \sum_{i=1}^{r} \gamma_i^j fit_i^j. \tag{7}$$

The SPC weights the fitness value of an evaluation example in its own set. The effect is to control the selection pressure to avoid premature convergence. At the beginning of the evolutive process, the solutions have been randomly generated, then they will obtain high fitness values (in a minimization problem). At this evolutive point, the strategy is to reward any advantages, even the slightly little ones that a solution could acquired. Therefore the best fitness values, obtained on a set of evaluation examples, must have much more weight in the calculation of fit_B^j. This is a way to increase the importance of the evaluation examples in which the solution shows some right behavior. The fitness values of the set, fit_B^j, are therefore biased to best fitness values. As the evolutive process develops, the contribution of all examples tends to be the same. When the solutions have very good behaviors, then calculation of fit_B^j is biased to the worst fitness values, that is, the performance over the most difficult examples is more important. In this phase the objective is to adjust details of the generalized solution. Figure 13 plots the shape of the parameter w_i^j.

Each set of evaluation examples has a fitness value fit_B^j at this point the fitness value of the solution, fit_{SOL}, related with them can be calculated, following Equation (8):

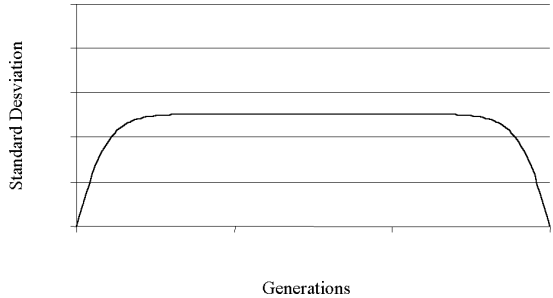


Figure 14. Idealized evolution of standard deviations of fitness values.

$$fit_{\mathrm{SOL}} = \left(\frac{1}{m}\sum_{j=1}^{m} fit_B^j\right) \pm K\sigma_{fit_B}. \tag{8}$$

The selection operator uses the fit_{SOL} value to choose the solutions that will generate new ones. The fit_{SOL} value is calculated with two factors: the average and the standard deviation of the fitness values of the set of evaluation examples related with it. The parameter K is an experimental constant. The standard deviation has the effect of improving the solutions with homogenous behavior. The solutions obtained must be general, and the average fitness value is not enough to drive the evolution to get this objective. The standard deviation is very different than the fitness value. At the beginning of the evolutive process, all solutions have similar poor behaviors, therefore, the standard deviation has small values. Near the end of the evolutive process something similar happens, all the solutions have reach a high level of performance. In Figure 14 shown how must be the evolution of standard deviation of the best solution fitness values measures over its evaluation examples.

A summary of the process to calculate the fitness value of a solution fit_{SOL} is shown in Figure 15.

4.2. GENERATION OF EVALUATION ENVIRONMENTS

As was already seen, the global evaluation of the fitness value of a solution is performed over an object configuration. The robot starts in different positions, the so-called evaluation examples. A new genetic operator, incremental mutation operator (IGO), was proposed to generate new evaluation examples. The IGO uses a measure distance between examples. The distance function is shown in Equation (9). The parameters used in the incremental mutation operator are $a = 400$, $b = 0.01$, $F_{\max} = 65000$ and $F_{\min} = 0$. These parameters have been experimentally adjusted:

$$\theta\left(fit_B^i\right) = 400\left(1 - \frac{1 - \mathrm{e}^{-(2/65000)fit_B^i \ln((400-0.01)/0.01)}}{1 - \mathrm{e}^{-2\ln((400-0.01)/0.01)}}\right). \tag{9}$$

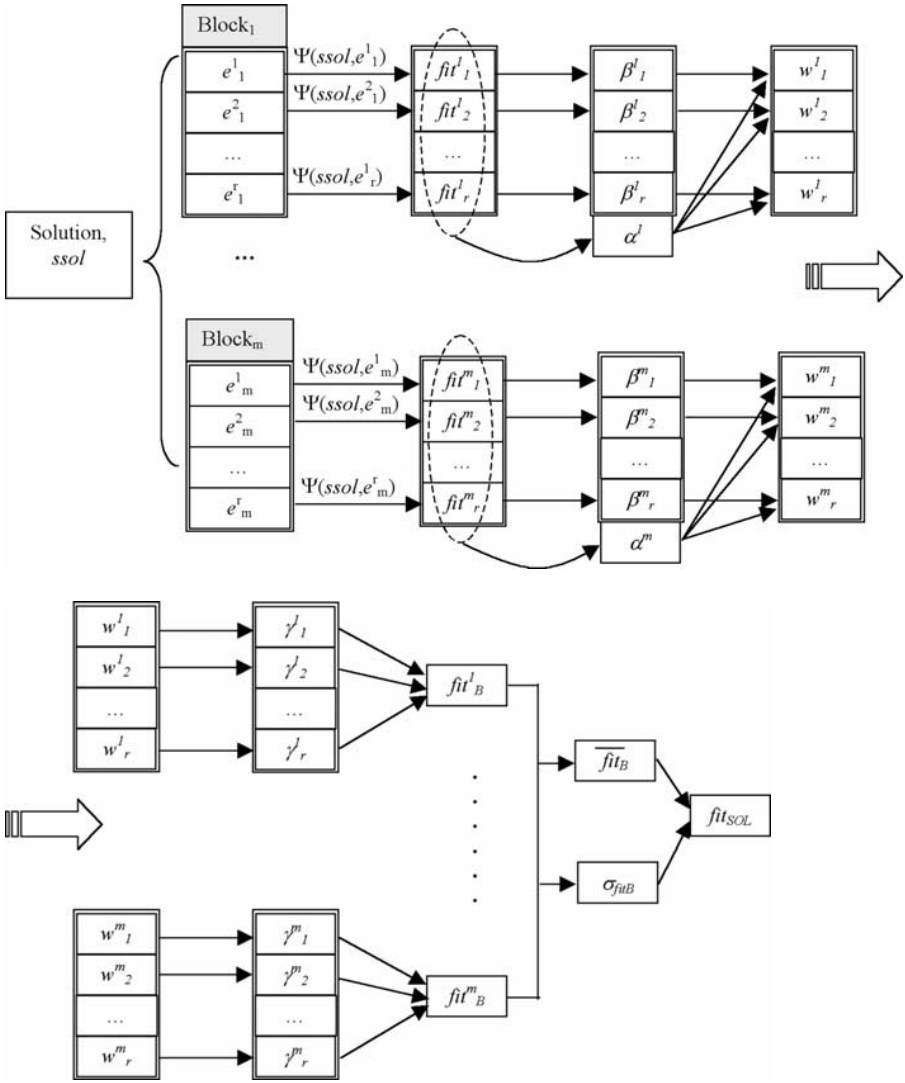


Figure 15. Calculus of fitness value of an individual, fit_{SOL}.

Figure 16 shows the shape of Equation (9). It can be observed that when the fitness values of a set of evaluation examples, get the half value of the possible amount, then the distance is 1%. This value smooths the convergence to prevent the disorientation in the learning process. The maximum distance value has been fixed to 400 because the object configuration has a 500×500 positions size.

To generate a new evaluation example, the function distance is applied to each parameter of an evaluation example. Let x_l^r, y_l^r, a_l^r be the initial position and ori-

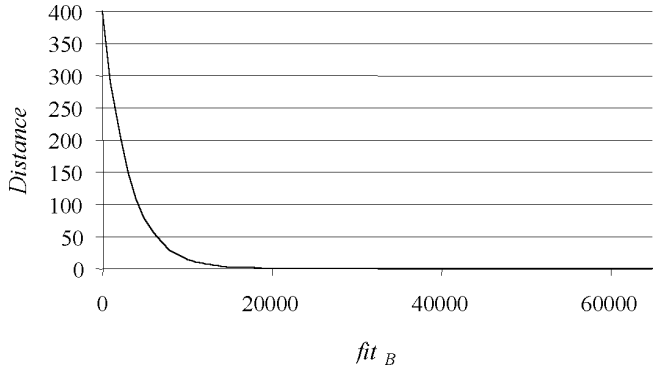


Figure 16. Distance between evaluation examples.

entation of the r-evaluation example of the l-set of evaluation examples. First the distance θ_l is calculated, then the new example parameters are given by Equation (10):

$$\begin{aligned} x'^r_l &= x^r_l + N(0, \theta_l), \\ y'^r_l &= y^r_l + N(0, \theta_l), \\ a'^r_l &= a^r_l + N(0, \theta_l). \end{aligned} \tag{10}$$

The $N(0, \theta)$ function is a gaussian distribution with mean 0 and standard deviation θ.

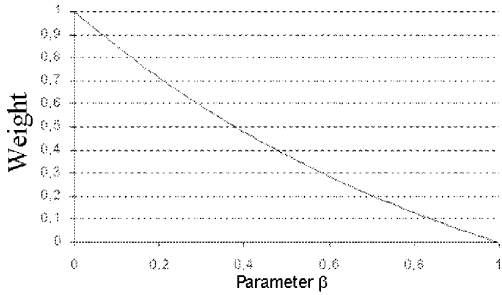
4.3. DYNAMIC OF THE EVOLUTIVE PROCESS

The dynamic of the UC method is the following: in the initial generations the controllers have poor navigation behavior (a lot of collisions, restricted movements), its fitness value in the evaluation function will be close to the worst value $F_{\max}$, therefore the examples (the initial position) must be unchanged till the controller develops some skill. The necessary parameter to calculate the fitness value is w (Equation (11)):

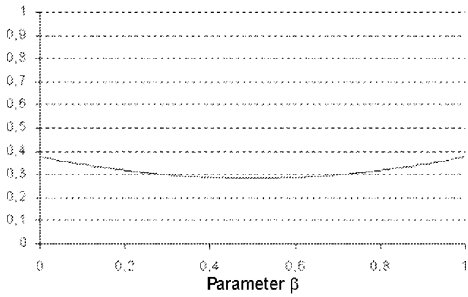
$$fit^i_j \approx F_{\max}, \qquad \alpha \approx 0, \qquad w \approx \frac{e^{1-\beta} - 1}{e - 1}, \quad 0 \leqslant \beta \leqslant 1. \tag{11}$$

In Figure (17) the weights of the fitness value of examples in a block in different evolutive steps are shown. At the beginning of the evolutive process, Figure 17(a), the better behaviors are the most weighted.

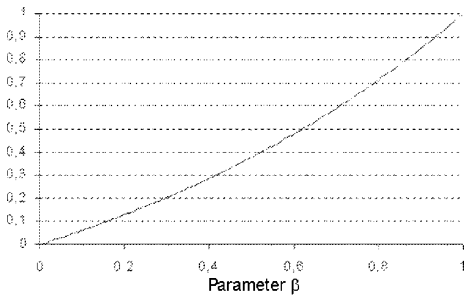
The selection pressure is applied in the direction of better controllers, those who have some minimal navigation skill. The controller evolves and when the fitness value is close to the half of the maximum possible the parameters of control pressure have the values (Equation (12)).



(a)



(b)



(c)

Figure 17. Parameter w in different evolutive steps.

$$\begin{aligned} fit_j^i &\approx \frac{F_{\max} + F_{\min}}{2}, \qquad \alpha \approx \frac{1}{2}, \\ w &\approx \frac{(e^{\beta} - 1) + (e^{1-\beta} - 1)}{(e - 1)(e^{1/2} + 1)}, \quad 0 \leqslant \beta \leqslant 1. \end{aligned} \tag{12}$$

In this situation, the weights in a block are the same for all training examples, Figure 17(b). At the end of evolutive process the controller has acquired a high

navigation capability. The fitness value of a controller in an environment example is close to 0; the contribution to the total fitness value is mainly far the worst behavior, Figure 17(c). In this step is important to weight the worst behavior to move the selection pressure to less adapted controllers, that is, the most general ones. With this mechanism the controller avoid the overadaptation to environment and the disorientation on the search process. The parameters of control pressure at this point are given by Equation (13).

$$fit_j^i \approx F_{\min}, \qquad \alpha \approx 0, \qquad w \approx \frac{(e^{\beta} - 1)}{(e - 1)}, \quad 0 \leqslant \beta \leqslant 1. \tag{13}$$

5. Experimental Results

The experiments are focused to the automatic learning controllers for the robot navigation problem. The objective is to obtain a controller able to navigate in any environment, that is, independently of the initial and goal positions and the objects configuration.

Two different sets of experiments have been performed called *fixed* and *coevU*. In both experiments, an Evolutionary Strategy is used, $(\mu+\lambda)$-*ES*, $\mu = 6$, $\lambda = 4$, in order to find the network connections weights. Experiments differ in the way they are evaluated on the learning environments. One of the experiments, which will be referred as *fixed*, is trained in the same environment during all the evolutive process; that means that starting and goal positions, as well as the obstacles configuration are constant. On the other hand, those experiments that use the UC method, called *coevU*, evolve the robot starting position and orientation, while they keep the goal position and obstacles configuration fixed.

Figure 18 shows the training object configuration. The dot in the middle of the configuration indicates the goal position, that is the same for all experiments performed. These object configurations have been used in both *fixed* and *coevU* experiments.

The *fixed* experiments are summarized in Table I and the *coevU* ones in Table II.

A *coevU* experiment has been done for each objects configuration, in *fixed* experiments the same object configuration is used in some cases, just varying theirs robot start position. Thus, in the *fixed* experiments 3, 6, 7 and 9 the learning process takes place in the object configuration 3 and *fixed* experiments 4 and 5 in objects configuration 4. In those experiments just the angle and starting position of the robot change.

Each experiment has been repeated five times with different random seed, the best controller obtained was validated. The evolutive process has performed 200 generations. In each evolutive process a different initial population for the evolutionary strategy is generated. The best controller achieved in this training process is used later in the validation process.

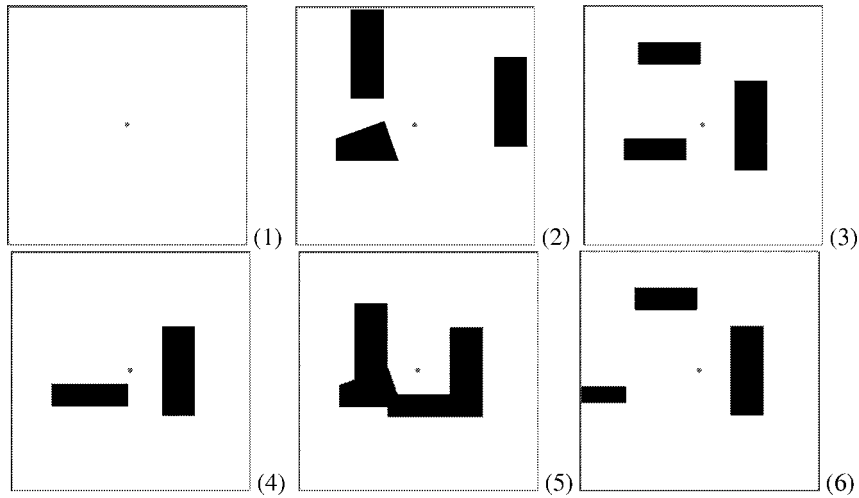


Figure 18. Objects configuration used in training and validating process.

Table I. Object configuration and starting robot position in *fixed* experiments

Experiment	Training example. Robot initial position: (x, y), angle	Object configuration
1	(50, 50, 315)	1
2	(450, 400, 90)	2
3	(50, 400, 0)	3
4	(50, 300, 0)	4
5	(450, 300, 270)	4
6	(300, 400, 90)	3
7	(200, 50, 0)	3
8	(250, 400, 0)	5
9	(425, 250, 115)	3
10	(200, 20, 180)	6

Table II. Objects configuration and starting robot position in *coevU* experiments

Experiment	Training example. Robot initial position: (x, y), angle	Object configuration
1	Evolve	1
2	Evolve	2
3	Evolve	3
4	Evolve	4
5	Evolve	5
6	Evolve	6

5.1. MEASURE OF THE CONTROLLERS' FITNESS

The evaluation process Ψ is applied on the genotype of a solution, *ssol*, on a specific evaluation example, $seje_i^j$ (i-example of j-block), in order to obtain de evaluation measure fit_i^j. This evaluation proccess measures some quality skill of robot behavior:

- Number of cycles necessary to reach the goal, T. If the goal is not reached the value is 2000.
- Trajectory length of the robot path, L.
- Number of collisions, C.
- Number of cycles in which the robot stayed in the same position, S.
- Euclidean distance between the robot's starting position and the goal position, D_m.
- Euclidean distance between the robot's starting and final position, D_o.

Equation (14) shows the lineal combination and weights used to obtain the fitness value of a controller, obtained from the measurements of its behavior.

$$f_j^i = 20T - 1.5L + 10C + 10S + 10D_m - 1.5D_o. \tag{14}$$

The direction of the learning process is to minimize the fitness value. For the *fixed* experiment the fitness function is the base measurement used to apply the selection operator. For the *coevU* experiment, this is the value applied in Equations (3), (4), and (7) to calculate the block fitness value. In these experiments, constant K in Equation (8) has an experimental value of 0.25.

5.2. EVOLUTION OF CONTROLLERS IN *fixed* AND *coevU* EXPERIMENTS

A very different learning process will be expected in both types of experiments. The Figure 19 shows the evolution of average fitness values in the learning process in experiments 1 and 3 for both *fixed* and *coevU* experiments.

The comparison of the fitness values evolution in the learning process between *fixed*, Figures 19(a), (b) and *coevU*, Figures 19(c), (d), shows significant differences. The evolution in coevolutive method has been smoothed and the shape, in detail, is very rough. The sharp-edged shape of Figures 19(c), (d) appears as a consequence to alternating the selection pressure between solutions and evaluation examples systems, in spite of this fact, the tendency of the plots converges exponentially to the optimum fitness value. In the classical method (*fixed*) the convergence is too abrupt, that means that the learning process occurs in few steps. It is important to remark that the distance between the minimum, mean and maximum plots goes down at the same rate. This is a consequence of maintaining the genetic diversity at a high level. Therefore the learning process avoids local minimal and overadaptation. The global effect is the achievement of solutions with good generalization behaviors, as will be demonstrated in the validation process (Sections 5.3 and 5.4).

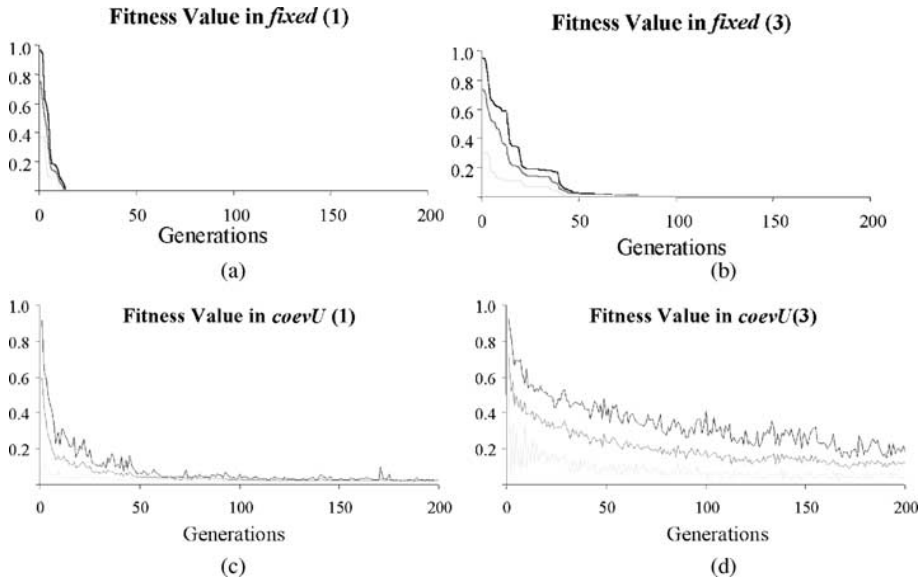


Figure 19. Fitness value evolution in object configuration 1 and 3.

5.3. GENERALIZATION CAPABILITIES OF UNIFORM COEVOLUTION METHOD

Once a controller has been learned in a specific environment, it is interesting to know how well this controller behaves when the object configuration and starting position is changed randomly. The capability to show similar behavior, no matter the environment we are dealing with, is referred as generalization capability. To test this generalization capability of the Uniform Coevolutive method a validation process was performed.

The validation process has been carried out making 1000 executions over all object configurations. The best controller obtained in every training execution (five per objects configuration) is used in the validation process. Each execution has different randomly generated initial position and orientation for the robot. In order to make some quantitative comparison some numerical results have been compiled from *fixed* experiments in Table III and from *coevU* in Table IV.

In Tables III and IV the average percentage of best controllers that reach the goal are summarized. At the bottom of Tables III and IV the average value over the column is calculated. This value can be considered as a measure of the difficult for the controllers to navigate in a specific objects configuration. Thus is a measure of the complexity in terms of navigation through it. Higher value in column average implies higher probability of arriving, therefore this objects configuration is easy to navigate, for instance, in *fixed* experiments, objects configuration 1 is the easiest and number 5 is the hardest. In *coevU* experiments, the values in the average column show similar results for all the configurations. The average over the row

Table III. Probability of a robot trained in a specific object configuration reach the goal in all object configurations, *fixed* experiments

		Validation object configuration						
		1	2	3	4	5	6	Average
Experiment	1	0.024	0.02	0.02	0.02	0.02	0.02	0.02
fixed	2	0.63	0.32	0.23	0.35	0.14	0.31	0.33
	3	0.73	0.34	0.35	0.43	0.23	0.44	0.42
	4	0.86	0.42	0.32	0.57	0.32	0.38	0.48
	5	0.84	0.34	0.29	0.47	0.16	0.38	0.41
	6	0.88	0.34	0.20	0.42	0.12	0.44	0.40
	7	0.87	0.38	0.34	0.58	0.39	0.47	0.51
	8	0.68	0.42	0.34	0.34	0.29	0.58	0.44
	9	0.49	0.35	0.23	0.34	0.34	0.37	0.35
	10	0.82	0.42	0.38	0.63	0.22	0.57	0.51
Average		0.68	0.34	0.27	0.41	0.22	0.40	0.39

Table IV. Probability of a robot trained in a specific object configuration reach the goal in all object configurations, *coevU* experiments

		Validation object configuration						
		1	2	3	4	5	6	Average
Experiments	1	1.00	0.50	0.81	0.90	0.71	0.82	0.79
coevU	2	0.97	0.77	0.84	0.91	0.83	0.86	0.86
	3	0.99	0.82	0.79	0.87	0.61	0.84	0.82
	4	0.98	0.44	0.77	0.89	0.70	0.86	0.77
	5	1.00	0.81	0.80	0.89	0.80	0.80	0.85
	6	0.99	0.80	0.80	0.90	0.73	0.80	0.84
Average		0.99	0.69	0.80	0.90	0.73	0.83	0.82

in Tables III and IV, can be also be calculated and considered as the generalization level obtained by the best controller evolved in an object configuration. The higher value in row average, the highest probability of arriving, and considering different objects configuration, the most general controller obtained. So in *fixed* experiments, objects configuration 1 generates the worst controller (less general) and the small value obtained, 0.02, suggests an absence of navigation skills for controller 1. The *coevU* experiments show a high independence of learning objects configuration used and in all the cases the percentage is higher than *fixed* experiments.

Table V. Averages of reach the goal of best controllers evolved in six objects configurations

Objects configuration	Average (*fixed*)	Average (*coevU*)	Improve
1	0.02	0.79	0.77
2	0.33	0.86	0.53
3	0.51	0.82	0.31
4	0.48	0.77	0.29
5	0.44	0.85	0.45
6	0.51	0.84	0.33

In Table V a comparison among both experiments of the best percentage in average of reaching the goal is shown. In *fixed* experiments with the same objects configuration the value presented is the best so far.

The column "Improve" of Table V indicates the difference between the average values of percentages of reaching the goal of *coevU* and *fixed* methods. The improvement value corresponding to objects configuration 1 is surprising, considering the results obtained in the *fixed* experiments. In the *coevU* results, the controller from objects configuration 1 has obtained the same navigation capability than other controllers evolved in objects configurations with obstacles. This feature seems to point out that the learning process in *coevU* method is independent of the object configuration used. Even if an environment with no objects at all is used in the learning process, the controller obtained behaves very well in an environment with an arbitrary objects configuration. If the average of improve column is calculated, a value of 0.42 is obtained, that means a 42% of average improvement when using the UC method.

In Figure 20 the probability of reach the goal in average obtained in both experiments types are represented and the distance between the two lines represents the mentioned improvement.

Finally, in Figure 21 a histogram of all fitness values obtained by all the controllers learned in all the objects configuration when tested in the validation process, both in *fixed* and *coevU* experiments, is represented. Lines in light color corresponding to the classical evolutive technique, *fixed*, and the dark ones to *coevU*. The values of fixed experiments are clustered in two areas. The area of values $[0-0.2]$ a 38% of the total area, corresponding to right behavior, reach the goal avoiding obstacles. The area above 0.8 (53%) belongs to controllers that hit the obstacles or wall, stand in the same place and does not reach the goal at all.

Analyzing the areas under the plot of *coevU* experiments, it can be seen that under fitness value 0.2 (good behaviors) constitutes the 80% of all obtained fitness values in *coevU* method. In the area of bad behaviors, above 0.8, to *coevU* are more

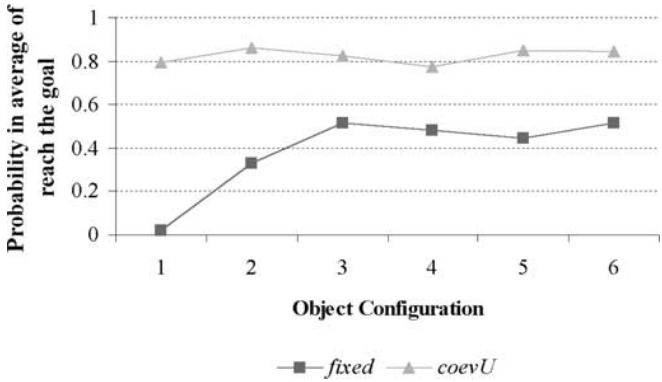


Figure 20. Probability of a robot trained in a specific object configuration reach the goal.

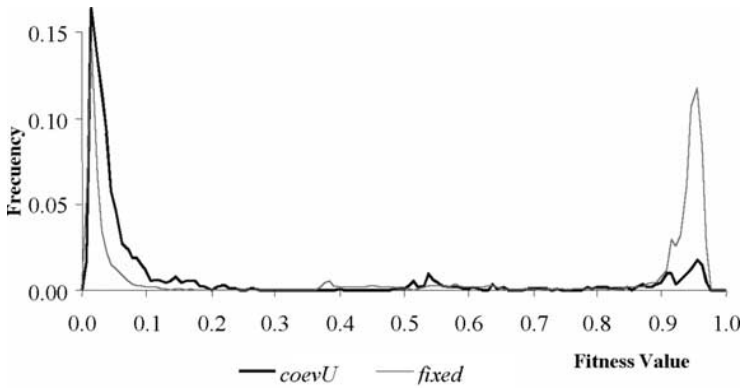


Figure 21. Histogram of the fitness values of UC and classical method.

or less the 12%, less of a fifth of the region occupied by the classical method *fixed*. The percent of exploratory strategies, the robot navigate without reaching the goal, corresponding to fitness value in the interval of 0.2–0.8; they have the same level in both methods. The total improvement when the coevolution uniform method is applied is very notable.

6. Conclusions

In this paper a new coevolutive method, called uniform coevolution, has been introduced to solve the robot navigation problem. The main objective of UC is to obtain general solutions applying an evolutionary computation technique, in this problem domain, to find a neural network controller evolved with evolutionary strategies. The algorithm proposed has many characteristics that improve the search of generalized solutions. The calculation of fitness value takes into account the examples used to learn and the global population evolution. The evolutive dynamic of uniform coevolution alternates the selection pressures in both systems. The training

examples system evolves to best trainer and the solutions system to generalized behavior. These characteristics produce smooth convergence, permit avoid local minima.

Two kinds of experiments have been done: a classical evolutionary strategy and the coevolutive one. A comparison of the generalization achieved with both methods was performed. The results have shown the high level of generalization obtained with uniform coevolution. This is a great improvement in evolutionary algorithms because classical evolutionary algorithms have a lack in generalization problem. Furthermore, the uniform coevolution is a general method, is easy to apply without effort to other different problems.

References

1. Axelrod, R.: *The Evolution of Cooperation*, Basic Books, New York, 1984.
2. Axelrod, R.: Evolution of strategies in the iterated prisioner's dilemma, in: L. Davis (ed.), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufman, 1989.
3. Baluja, S.: Evolution of an artificial neural network based autonomus plan vehicle controller, *IEEE Trans. Systems Man Cybernet.* **26**(3) (1996), 450–463.
4. Berlanga, A., Isasi, P., Molina, J. M., and Sanchis, A.: Competitive evolution to find generalized solutions: the arms race perspective, in: *Proc. of Intelligent Engineering Systems INES*, Austria, 1998, pp. 61–65.
5. Berlanga, A., Isasi, P., Sanchis, A., and Molina, J. M.: Distance modulation competitive co-evolution method to find initial configuration independent cellular automata rules, in: *IEEE Internat. Conf. on Systems, Man and Cybernetics*, Japan, 1999, pp. 607–612.
6. Berlanga, A., Sanchis, A., Isasi, P., and Molina, J. M.: Neural networks robot controller trained with evolution strategies, in: *Proc. of 1999 Congress on Evolutionary Computation, CEC99*, 1999.
7. Braitenberg, V.: *Vehicles: Experiments on Synthetic Psychology*, MIT Press Cambridge, MA, 1984.
8. Brooks, R. A.: Intelligence without representation, *Artificial Intelligence* **47** (1991), 139–159.
9. Brooks, R. A.: Artificial life and real robots, in: *Toward a Practice of Autonomous Systems: Proc. of the 1st European Conf. on Artificial Life*, MIT Press, Cambridge, MA, 1992.
10. Cliff, D. T., Husband, P., and Harvey, I.: Explorations in evolutionary robotics, *Adaptive Behaviour* (1993), 73–110.
11. Cliff, D. and Miller, G. F.: Tracking the red queen: Measurements of adaptive progress in coevolutionary simulations, in: F. Morán, A. Moreno, J. J. Merelo and P. Chacón (eds), *Proc. of the 3rd European Conf. on Artificial Life*, Springer, Berlin, 1995, pp. 200–218.
12. Dorigo, M. and Snepf, U.: Genetics based machine learning and behavior based robotics: A new sysnthesis, *IEEE Trans. Systems Man Cybernet.* **23** (1993), 141–153.
13. Dorigo, M. and Colombetti, M.: Robot shaping: Developing autonomous agents through learning, *Artificial Intelligence* **71**(2) (1994), 321–370.
14. Dorigo, M.: Alecsys and the autonomouse: Learning to control a real robot by distributed classifier systems, *Machine Learning* **19** (1995), 209–240.
15. Ficici, Sevan, G. and Pollack, J. B.: Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states, in: Adami, Belew, Kitano and Talor (eds), *Proc. of the 6th Internat. Conf. on Artificial Life*, MIT Press, Cambridge, MA, 1998.
16. Floreano, D. and Mondada, F.: Evolution of homming navigation in a real mobile robot, *Proc. IEEE Trans. Systems Man Cybernet.* **26**(3) (1996), 396–407.

17. Floreano, D. and Nolfi, S.: Adaptive behavior in competing co-evolving species, in: *Proc. of the 4th European Workshop on Evolutionary Robotics*, Springer, Berlin, 1997.
18. Franceschini, N., Pichon, J. M., and Blanes, C.: Real time visuomotor control: from flies to robots, in: *Proc. of the 5th Internat. Conf. on Advanced Robotics*, 1991, pp. 91–95.
19. García-Martínez, R. and Borrajo, D.: An integrated approach of learning, planning, and execution, *J. Intelligent Robotic Systems* (2000), accepted for publication.
20. Gaudiano, P., Zalama, E., and Lopez, J.: An unsupervised neural network for low level control of a wheeled mobile robot: noise resistance stability and hardware implementation, *IEEE Trans. Systems Man Cybernet.* **26**(3) (June 1996), 485–495.
21. Goldberg, D.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, New York, 1989.
22. Graf, D. H. and LaLoncle, W. R.: A neural controller for collision-free movement of general robot manipulators, in: *Proc. of the IEEE 2nd Internat. Conf. on Neural Networks*, Vol. I, 1988, pp. 77—84.
23. Handley, S. G.: The genetic planner: The automatic generation of plans for a mobile robot via genetic programming, in: *Proc. of IEEE Internat. Symp. on Intelligent Control*, Chicago, 1994, pp. 190–195.
24. Hillis, W. D.: Co-evolving parasites improve simulated evolution as an optimization procedure, in: C. G. Langton (ed.), *Artificial Life II*, Santa Fe Institute, Addison-Wesley, Readin, MA, 1991, pp. 313–324.
25. Isasi, P., Berlanga, A., Molina, J. M., and Sanchis, A.: Robot controller against environment, a competitive evolution, in: *Special Session on Evolution Computation, 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics*, Germany, 1997.
26. Ishikawa, S.: A method of autonomous mobile robot navigation by using fuzzy control, *Adv. Robotics* **9**(1) (1995), 29–52.
27. Koza, J.: *Genetic Programming*, MIT Press, Cambridge, MA, 1992.
28. Koza, J.: Evolution of subsumption architecture that perform a wall following task for an autonomous mobile robot via genetic programming, in: *Computational Learning Theory and Natural Learning Systems*, Vol. 2, MIT Press, Cambridge, MA, 1994, pp. 321–346.
29. Kwok, D. P., Leung, T. P., and Feng, S.: Genetic algorithms for the optimal dynamic control of robot arms, in: *Proc. of the 19th Annual Conf. of IEEE Industrial Electronic Society*, Vol. 1, Maui, 1993, pp. 381–385.
30. Lindergren, K. and Nordahl, M. G.: Artificial food webs, in: *Artificial Life III*, Addison-Wesley, Reading, MA, 1994, pp. 73–103.
31. Maes, P. and Brooks, R.: Learning to coordinate behaviors, in: *Proc. of the 8th National Conf. on Arfificial Intelligence*, Morgan Kaufmann, San Mateo, CA, 1990, pp. 796-802.
32. Mahavedan, S. and Connell, J.: Automatic programming of behavior-based robots using reinforcement learning, *Artificial Intelligence* **55** (1992), 311–365.
33. Mataric, J. and Cliff, D.: Challenges in evolving controllers for physical robots, *J. Robotics and Autonomous Systems* **19**(1) (1996), 67–83.
34. Matellán, V., Fernández, C., and Molina, J. M.: Genetic learning of fuzzy reactive controllers, *Robotics Autonom. Systems* **25**(1/2) (1998), 33–41.
35. McKerrow, P. J.: *Introduction to Robotics*, Addison-Wesley, Reading, MA, 1991.
36. Miglino, O., Hautop, H., and Nolfi, S.: Evolving mobile robots in simulated and real environment, *Artificial Life* **2** (1995), 417–434.
37. Millán, J. R.: Rapid, safe, an incremental learning of navigation strategies, *IEEE Trans. Systems Man Cybernet.* **26**(3) (June 1996), 408–420.
38. Molina, J. M., Sanchis, A., Berlanga, A., and Isasi, P.: An enhanced classifier system for autonomous robot navigation in dynamic environments, in: *Intelligent Automation and Soft Computing*, Autosoft Press, 1998, in press.

39. Mondada, F. and Franzi, P. I.: Mobile robot miniaturization: A tool for investigation in control algorithms, in: *Proc. of the 2nd Internat. Conf. on Fuzzy Systems*, San Francisco, USA, 1993.
40. Nagata, S., Sekiguchi, M., and Asakawa, K.: Mobile robot control by a structures hierarchical neural network, *IEEE Control Systems Mag.* (April 1990), 69–76.
41. Paredis, J.: Coevolutionary computation, *Artificial Life* **2** (1996), 355–375.
42. Pomerleau, D.: *Neural Network Perception for Mobile Robot Guidance*, Kluwer Academic, Dordrecht, 1993.
43. Pollack, J. B., Blair, A. D., and Land, M.: Coevolution of a backgammon player, in: V. C. G. Langton (ed.), *Proc. of Artificial Life*, MIT Press, Reading, MA, 1996.
44. Rechenberg, I.: *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog, Stuttgart, 1973.
45. Rechenberg, I.: Evolution strategy: Nature's way of optimization, in: *Optimization: Methods and Applications, Possibilities and Limitations*, Springer, Berlin, 1989, pp. 106–126.
46. Rosin, C. D. and Belew, R. K.: New methods for competitive coevolution, *Evolutionary Computation* **5** (1997), 1–29.
47. Sanchis, A., Molina, J. M., Isasi, P., and Segovia, J.: RTCS: A reactive with tags classifier system, *J. Intelligent Robotic Systems* (1999), in press.
48. Molina, J. M., Sanchis, A., Berlanga, A., and Isasi, P.: An enhanced classifier system for autonomous robot navigation in dynamic environments, *Intelligent Automat. Soft Comput.* **6**(2) (2000), 113–124.
49. Shakey, N.: Shakey the robot, Technical Report, SRI A.I., 1984.
50. Solano, J. and Jones, D. I.: Generation of collision-free paths, a genetic approach, in: *Proc. of the IEEE Colloquium on Genetic Algorithms for Control and Systems Engineering*, London, 1993.
51. Sommaruga, L., Merino, I., Matellán, V., and Molina, J.: A distributed simulator for intelligent autonomous robots, in: *4th Internat. Symp. on Intelligent Robotic Systems–SIRS96*, Lisboa, Portugal, 1996.
52. Schwefel, H. P.: *Numerical Optimization of Computer Models*, Wiley, New York, 1981.
53. Stone, P. and Veloso, M.: A layered approach to learning client behaviors in the robocup soccer server, *Appl. Artificial Intelligence J.* **12** (1998), 165–188.