Generalization Capabilities of Co-evolution in Learning Robot Behavior

A. Berlanga,[*] A. Sanchis,[†] P. Isasi,[‡] and J.M. Molina[§]
SCA-LAB, Universidad Carlos III
Avda Universidad, 30, 28911, Madrid, SPAIN

In this article, a co-evolutive method is used to evolve neural controllers for general obstacle-avoidance of a Braitenberg vehicle. During a first evolutionary process, Evolution Strategies were applied to generate neural controllers; the generality of the obtained behaviors was quite poor. During a second evolutionary process, a new co-evolutive method, called Uniform Co-evolution, is introduced to co-evolve both the controllers and the environment. A comparison of both methods shows that the co-evolutive approach improves the generality of controllers.

1. INTRODUCTION

Navigation is a fundamental requirement for autonomous mobile robots. Approaches based on classical paradigms (abstraction, planning, heuristic search, etc.) were not completely suitable for unpredictable and dynamic environments. Other approaches consider "reaction" as the new paradigm to build intelligent systems. One classical instance of this kind of architecture is the subsumption architecture that was proposed by Brooks[1] and has been successfully implemented on several robots of MIT and other institutes. The base of the subsumption architecture is "behavior." Each behavior reacts in a situation and the global control is a composition of behaviors. Different systems, from finite state machines to fuzzy controllers,[2] have been used for the implementation of these behaviors. The rules of these behaviors could be designed by a human expert, designed "ad-hoc" for the problem, or learned by using different artificial intelligence techniques. Machine learning has been applied to shape the behavior of autonomous robots; some approaches use Genetic Algorithms to evolve fuzzy controllers,[3] Classifier Systems to teach controllers,[4,5] or Neural Networks to learn behaviors.[6]

In this work, a neural network ought to learn a behavior, "navigation," by using Evolution Strategies[7,8] (ES) as a machine learning technique to obtain the right association between inputs and outputs. The

*To whom all correspondence should be addressed; e-mail: aberlan@ia.uc3m.es.
†masm@inf.uc3m.es.
‡isasi@ia.uc3m.es.
§molina@ia.uc3m.es.

neural network is a feed forward network with eight input units and two output units directly connected to motors. This architecture has appeared in previous works[9] as an efficient way to learn a simple behavior: "avoid obstacles" by using Genetic Algorithms.

A general problem of machine learning techniques is the overadaptation to the environment used in the learning phase. In this way, an overadapted controller is able to navigate only in this environment. This problem becomes harder when evolutionary methods are used. An excessive adaptation to the environment could abort the generalization capability of a solution. Overadapted behaviors are not useful when a robot has to work in environments that differ from the ones used in the learning phase. Therefore, the robot ought to learn a behavior useful for any situation, what is called a generalized behavior.

A new co-evolutive method, Uniform Co-evolution (UC), is proposed to obtain a general navigation behavior. Some previous works have proven the usefulness of co-evolution[10–13] to improve the evolutionary computation techniques from different perspectives. The final performance of the system is improved as a consequence of the incremental adaptation among constituents. More recently, some works aimed at establishing the theoretical basis in co-evolution have been reported.[14–16]

In Section 2, a description of the problem is outlined. Section 3 is related with the application of Evolution Strategies in predefined environments and the overadaptation of the learned controllers is shown. The Uniform Co-evolution method is described in Section 4. In Section 5, the generalization capabilities of co-evolutive controllers and the comparison of the behaviors obtained with a simple ES and UC method is presented, and finally, some conclusions are outlined in Section 6.

2. PROBLEM DESCRIPTION

In this work, a simulator based on an autonomous robot named Khepera[6] is used. The sensory inputs come in from eight infrared proximity sensors that give some information about the obstacles and the distance and the angle to a light source. The robot has two wheels controlled by two independent motors. In order to prove the different configurations of the controllers, a simulator (SimDAI) developed in a previous work[17] has been used. In the simulator, the characteristics of the turtle robot model[18] and the physical restrictions of the Khepera robot have been considered. 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 robots in a 2-D simulated world of obstacles, which is separately monitored. This simulator has been used in many other works[3–5,19] in machine learning domains. Six environments (different objects configurations, Figure 1) are used in the learning phase. In Figure 1, the final goal position appears as a spot.

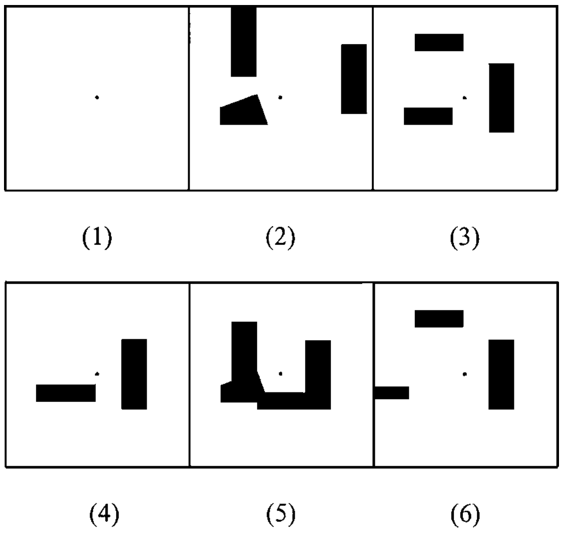


Figure 1. Objects configuration (environments).

It has been proven that by means of connections between sensors and actuators, a controller is able to solve any autonomous navigation robotic behavior.[20] This theoretical approach is based on the possibility of finding the right connections of a feed-forward Neural Network (NN) without hidden layers for each particular problem. (See Fig. 2.) The input sensors considered in this approach are the ambient and proximity sensors, s_i, in Figure 3. The NN outputs are the wheel velocities. The NN architecture is shown in Figure 2.

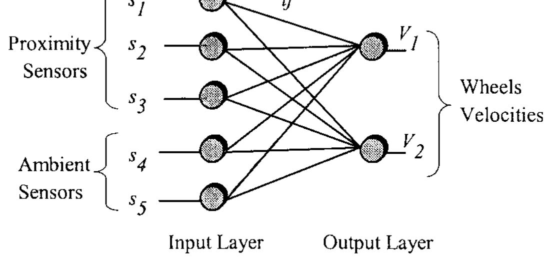
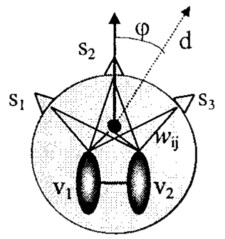


Figure 2. Neural network architecture.



s_i: Input of i-sensor
v_j: Velocity of j-wheel
d: Goal distance
φ: Goal angle
W_{ij}: Weight between i-sensor and j-wheel

Figure 3. Connections between sensors and actuators in the Braitenberg representation of a Khepera robot.

The velocity of each wheel is calculated by means of a linear combination of the sensor values (Figs. 2 and 3):

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

Where w_{ij} are searched weights, s_i are sensor input values, and f, described in Eq. 2, is a function for constraining the maximum velocity values of the wheels.

$$f(x) = \begin{cases} -1, & x < -1 \\ x, & -1 \le x \le 1 \\ 1, & x > 1 \end{cases} \tag{2}$$

The aim of the autonomous robot is to reach a goal in a complex environment while avoiding obstacles found in its path. Optimization methods do easily find a good controller for a specific environment. The objective of this work, however, is to obtain robust controllers with good performance in any environment. 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 their description are the only previous pieces of information. From the initial situation, the robot is able to learn through experience the optimal associations between inputs and outputs.

The learning of weights could be defined as a search problem in a domain of real values. This problem could be re-defined as an optimization problem, and then ES will be used. ES have proven a high degree of successful in this kind of problems. Two different methods will be applied: a simple ES (in a fixed environment) and a co-evolutive method.

3. APPLYING EVOLUTION STRATEGIES FOR LEARNING NAVIGATION

ES developed by Rechenberg[7] and Schwefel,[8] have been traditionally used for optimization problems with real-valued vector representations. As Genetic Algorithms[21] (GA), ES are heuristic search techniques based on the building blocks hypothesis. Unlike GA, however, in ES the search is focused in the gene mutation. This adaptive mutation is based on the fitness value of the individual. Recombination plays an important role in the search also, nevertheless, not as important as that of the adaptive mutation.

In this approach, each individual is composed of a 20 dimensional, real-valued vector, representing each of the weights and their corresponding variances. The individual represents one robot behavior, the consequence of applying the weights to Eq. 1. The evaluation of behaviors is used as the fitness function.

3.1. Experiments

The experiments are focused on the automatic learning of 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.

The first type of experiments, called *fixed*, a $(\mu + \lambda)$-ES, $\mu = 6$, $\lambda = 4$, was applied in order to find the neural network weights. The same environment was used during all the evolutive process. That means that the starting and goal positions, as well as the obstacles configuration, remained constant. Two hundred generations were used as a termination criterion. Five runs with different random seeds were performed in each evolutive process. The $(6 + 4)$-ES has been performed over ten different environments. An environment consists of an objects configuration (Fig. 1) and a robot starting position and initial orientation (Table I).

3.2. Measure of the Controllers' Fitness

To obtain the controller fitness value, the simulation has been run for a period of 2,000 cycles. Simultaneously, a log of its behavior is recorded. The measures that will be taken into account to calculate the fitness value are the following:

- Number of cycles necessary to reach the goal, T. If the goal is not reached, then the value is 2,000.
- Length of the robot's trajectory, L.

Table I. Object configuration and robot starting 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

- Number of collisions, C.
- Number of cycles in which the robot stayed in the same position, S.
- Euclidean distance between the robot's starting and final positions, D_o.
- Euclidean distance between the robot's starting position and the goal position, D_m.

Equation 3 shows the lineal combination and weights used to calculate the fitness value of a controller, obtained from the measurements of its behavior.

$$fit_j^i = 20T - 1.5L + 10C + 10S + 10D_m - 1.5D_o \quad (3)$$

To adjust the weights of Eq. 3 some previous experiments were performed.

3.3. Generalization Capabilities of Evolution Strategies

Table II shows the percentage of controllers obtained in the learning process that have been able to reach the goal. The average of the column must be a measure of the navigation difficulty of the environment. Thus, the environment of experiment 1 is the easiest (as it was expected) and number 5 is the hardest. The average of the row is the generalization level obtained by the best controller evolved in an object's environment. The controller evolved in experiment 1 was the worst; it has no navigation skill. Controllers evolved in experiments 7 and 10 are the best. It is not possible to give a correlation between the difficulty of an environment and the navigation skill that a controller is able to get. The generalization of controllers decreases due to overadaptation to training environments.

4. UNIFORM CO-EVOLUTION

New extensions to classical evolutive techniques were proposed to improve the generalization level. One of these is co-evolution, which was proposed[23] as a method to obtain more general solutions without an increment of domain information. In this work, we propose a new co-evolutive method called Uniform Co-evolution (UC) to treat this problem. The architecture of UC is composed of 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 4.

Table II. Probability of a robot trained in a specific objects configuration reaching the goal in all object configurations, *fixed* experiments.

		Validation object configuration						
		1	2	3	4	5	6	Average
Training object configuration	1	0.024	0.02	0.02	0.02	0.02	0.02	0.02
	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

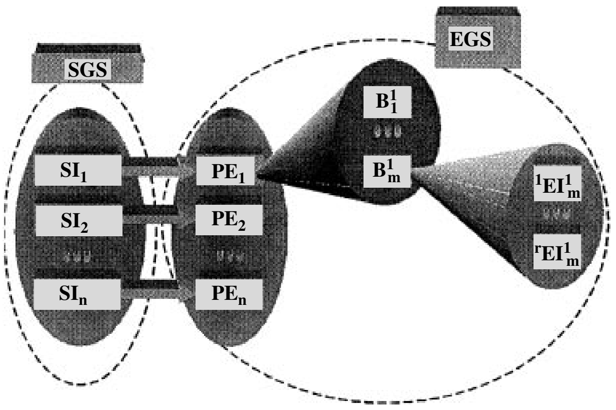


Figure 4. Uniform Co-evolution architecture.

The solutions and examples systems are described below:

- The Solutions Generator System (SGS) consists of a population of solution individuals (SI_i). To compute their fitness, it is necessary to face each individual with a set of different situations, examples, represented by a population in the Examples Generator System (explained below). The objective of this system is to gradually generate better solutions for a particular problem. Any evolutionary computation method can be used, in which an individual represents one problem solution. The evolution of the SGS follows the dynamics of the evolutionary computation method selected.
- The population of the Examples Generator System (EGS) is composed of 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. Each individual of SGS is related with only one set of blocks. The evaluation of each individual of SGS is calculated over its related m-blocks, each composed of r training examples. Thus, the evaluation is performed over $m \times r$ training examples. This general scheme is shown in Figure 4. In order to make the structure more comprehensible, the different blocks have been called as follows:
  - ${}^kEI^i_j$: the training k-example of the j-block related with the i-individual of SGS.
  - B^i_j: the j-block related with the i-individual of SGS.
  - PE_i: the set of all B^i_j 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 of different training examples (each ${}^kEI^i_j$ represents a robot starting angle and position). In order to explore the solutions space, an $(\mu + \lambda)$-ES is applied to the SGS. Following the UC scheme, each solution of SGS has a given set of training examples associated. In Figure 5, a short scheme of the system and the evolutive algorithms applied is shown.

The UC method automatically evolves solutions and examples. 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 ${}^kEI^i_j$ 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 corresponding SI_i.
3. Generation of new populations
   - **(a)** Selection of individuals from SGS.
   - **(b)** New SGS population, in this case a $(6+4)$-ES, was applied to evolve neural network controllers.
   - **(c)** The evolution of EGS is related with the generation of new EIs for 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.

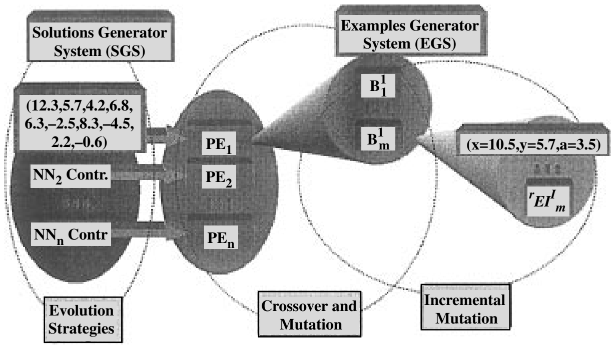


Figure 5. Uniform Co-evolution applied to the robot navigation problem.

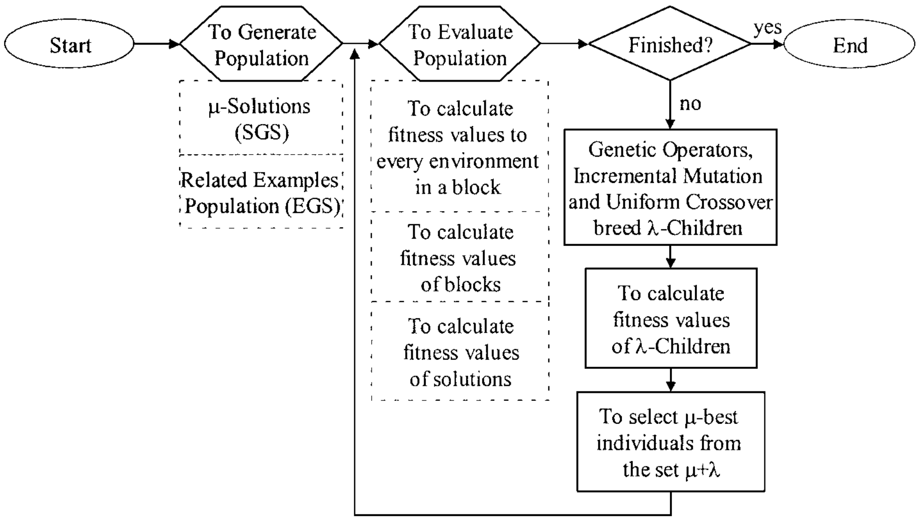

Start
To Generate Population
μ-Solutions (SGS)
Related Examples Population (EGS)
To Evaluate Population
To calculate fitness values to every environment in a block
To calculate fitness values of blocks
To calculate fitness values of solutions
Finished?
yes
End
no
Genetic Operators, Incremental Mutation and Uniform Crossover breed λ-Children
To calculate fitness values of λ-Children
To select μ-best individuals from the set μ+λ

Figure 6. Uniform Co-evolution scheme to solve the robot navigation problem.

A brief scheme of the process is shown in Figure 6.

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 an SGS individual is carried out following the classical rules of a ($\mu + \lambda$)-Evolution Strategy. In Figure 7 the individuals SI_i and SI_k have been selected to mate.

The second step is the recombination of the genotype of the selected solution (SI_i and SI_k). This operation generates two new solutions SI_i' and SI_k'. (See Fig. 8.) The recombination of the SGS individuals is carried out by recombination of the first training example of its related blocks. In Figure 8 the uniform recombination of the first examples is shown. In the recombination of the SGS individuals, the genetic code does not change; only individuals of the EGS interchange their first training examples.

The third step is the generation of new blocks of training examples. The IGO is applied to the first example to obtain a new one. The successive application of IGO to the last example generated produces a new one until the block is completed. This process is shown in Figure 9. The IGO uses the genotype of training 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 the SGS is obtained. In addition, during this process, the new EGS has been generated, too.

4.1. Fitness Value Calculation

The UC method requires some measures in order to calculate the final fitness value of every individual of SGS, fit_{SOL}. Below, the different steps followed to obtain fit_{SOL} are given in more detail. A summary of this process is shown in Figure 10.

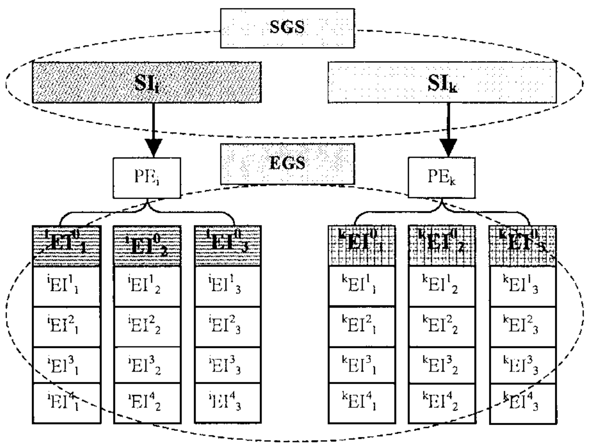


Figure 7. Generation of a new population in SGS and EGS.

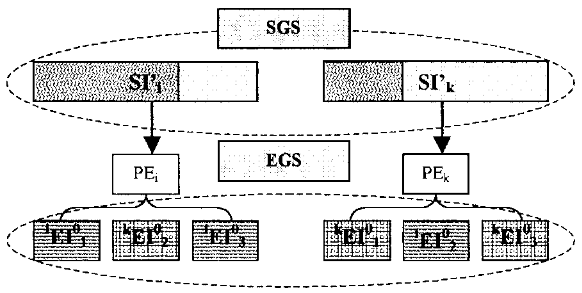


Figure 8. Recombination of SGS and EGS individuals.

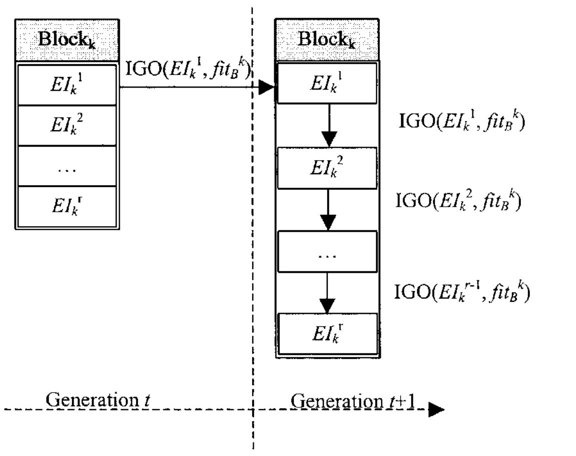


Figure 9. Generation of a new block of training examples.

First, the calculation of fit_i^j, Eq. 4, is carried out using the evaluation function, Ψ, where *ssol* is the genotype of the solution system and $seje_i^j$ is the i-example of the j-block associated with the individual of the SGS.

$$\Psi(ssol, seje_i^j) = fit_i^j \tag{4}$$

This value, fit_i^j, is the fitness value of the classical evolutionary computation techniques.

Next, the fitness value of each block of training examples, fit_B^j, Eqs. 5–9, has to be calculated.

Let us associate each solution of the SGS with m-blocks, each one of them composed of r training examples. The first calculated parameter is α^j

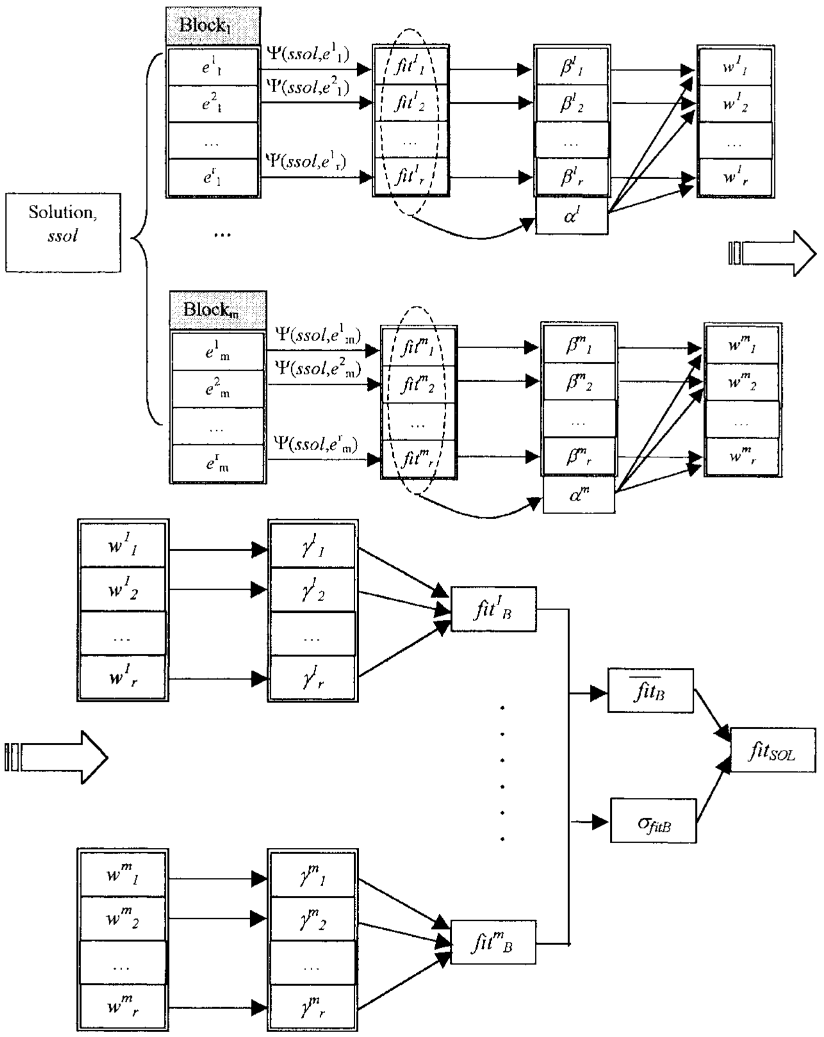


Figure 10. Calculation of the fitness value of an individual of the SGS, fit_{SOL}.

(calculated for j-block). This parameter is an absolute measure, related with how closely, in average, the solution proposed by SGS solves the problem in Eq. 5.

$$\alpha^j = 1 - \frac{\overline{fit}^j - F_{\text{MIN}}}{F_{\text{MAX}} - F_{\text{MIN}}}, \quad \text{where} \quad \overline{fit}^j = \frac{\sum_{i=1}^{r} fit_i^j}{r} \tag{5}$$

The highest and the lowest fitness values a solution can ever reach are F_{MAX} and F_{MIN}. It is not necessary to give exactly the boundary values of the range; an estimation of F_{MAX} and F_{MIN} and is enough.

Parameter β_i^j, calculated over each training example, is a relative reference. It gives a measure of the proximity of fit_i^j to the highest and lowest values of the fitness values in the block of training examples, to which it belongs. See Eq. 6.

$$\begin{cases} fit_{\max}^j = fit_{\min}^j, & \beta_i^j = 0 \\ fit_{\max}^j \neq fit_{\min}^j, & \beta_i^j = \frac{fit_i^j - fit_{\max}^j}{fit_{\min}^j - fit_{\max}^j} \end{cases} \tag{6}$$

The $fit_{\max}^j$ and $fit_{\min}^j$ values are the boundary fitness values obtained by solutions over their set of training examples. Therefore, each set of examples has a $fit_{\max}^j$, the highest value, and a $fit_{\min}^j$, the lowest fitness value. The parameter β_i^j is calculated for every example of a set of training examples. It measures how good the behavior of the solution over an example has been, with regard to the solution behavior over the other examples of the same set of training examples. Values of parameter β_i^j are in the range [0, 1].

The parameter w_i^j for each training example of each set associated with the i-solution is calculated with parameters α^j and β_i^j. (See Equation 7.) 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 is of fundamental importance in the UC method and has been named as Selection Pressure Control (SPC), as in Eq. 8.

$$w_i^j = \frac{\left(e^{\alpha^j} - 1\right)\left(e^{\beta_i^j} - 1\right) + \left(e^{1-\alpha^j} - 1\right)\left(e^{1-\beta_i^j} - 1\right)}{\left(e - 1\right)^2} \tag{7}$$

$$\gamma_i^j = \frac{w_i^j}{\sum_{k=1}^{r} w_k^j} \tag{8}$$

At this point, the fitness value of the set of training examples, fit_B^j, can be calculated, as in Eq. 9.

$$fit_B^j = \sum_{i=1}^{r} \gamma_i^j fit_i^j \tag{9}$$

The SPC weighs the fitness value of a training example within its own set. The effect is to control the selection pressure to avoid premature convergence.

At the beginning of the evolutive process, the solutions are generated randomly; therefore, they will obtain poor fitness values. At this evolutive point, the strategy consists in rewarding any advantage, even the slight ones that a solution could acquire. Therefore, the best fitness values, obtained over a set of training examples, must have much more weight in the calculation of fit_B^j. This is a way to increase the importance of the training examples in which the solution shows any good behavior. Therefore, the fitness values of the set, fit_B^j, are biased to the 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.

At this point, each set of training examples has a fitness value, fit_B^j, and the fitness value of the solution, fit_{SOL}, can be calculated finally, following Eq. 10.

$$fit_{SOL} = \left(\frac{1}{m} \sum_{j=1}^{m} fit_B^j\right) \pm K\sigma_{fitB} \tag{10}$$

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 training examples related with it. The parameter K is an experimental constant. The standard deviation has the effect of improving the solutions with a 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 from 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 reached a high level of performance.

Table III. Objects configuration and robot starting 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

4.2. Generation of Training Environments

As has already been seen, the global evaluation of the fitness value of a solution is performed over a meta-set of examples. The robot starts from different positions, the so-called training examples. A new genetic operator, the Incremental Genetic Operator (IGO), was proposed to generate new training examples. The IGO uses a measure of distance between examples. The distance function, $\theta(fit_B^i)$, is shown in Eq. 11. The parameters used in the incremental mutation operator are $a = 400$, $b = 0.01$, $F_{\mathrm{MAX}} = 65000 \quad yF_{\mathrm{MIN}} = 0$. These parameters have been adjusted experimentally.

$$\theta\left(fit_B^i\right) = 400\left(1 - \frac{1 - e^{-\frac{2}{65000} fit_B^i \ln\left(\frac{400-0.01}{0.01}\right)}}{1 - e^{-2\ln\left(\frac{400-0.01}{0.01}\right)}}\right) \tag{11}$$

This distance function smooths the convergence in order to prevent the disorientation of the learning process. The maximum distance value has been fixed to 400, because the environment has a size of 500 × 500 positions.

To generate a new training example, the function of distance is applied to each parameter of a training example. Let x_l^r, y_l^r, a_l^r be the initial position and orientation of the r-training example of the l-set of training examples. First, the distance, θ_l, is calculated and then the new example parameters are given by Eq. 12.

$$\begin{aligned} x_l'^r &= x_l^r + N(0, \theta_l) \\ y_l'^r &= y_l^r + N(0, \theta_l) \\ a_l'^r &= a_l^r + N(0, \theta_l) \end{aligned} \tag{12}$$

Where the $N(0, \theta)$ function is a Gaussian distribution with average 0 and standard deviation θ.

5. GENERALIZATION CAPABILITIES OF CO-EVOLUTION

In order to measure the generalization capability of the co-evolutive method, an Evolution Strategy, as that described in Section 3, $(\mu + \lambda)$-ES, $\mu = 6, \lambda = 4$, was used. These new experiments that use the uniform co-evolution algorithm *coevU*, evolve the robot starting position and orientation, while they keep the goal position and obstacles configuration fixed. (See Table III.) Five learning processes with 200 generations in each one were performed over each environment.

In the same way as in *fixed* experiments, to obtain controllers fitness values, the simulation has been run for 2,000 evolutive cycles, and Eq. 3 has been used to obtain the fitness value of a controller. This is the value applied in Equations 4, 5, and 8 to calculate the block fitness value. In these experiments, the constant K in Eq. 10 has an experimental value of 0.25.

5.1. Comparison of Generalization Capability

The *coevU* experiments show a high independence from the objects' configuration. In all the cases, the probability to reach the goal is higher than in *fixed* experiments. (See Table IV.) The Improvement column

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

		Validation object configuration							
		1	2	3	4	5	6	Average	Improvement
Training object configuration	1	1.00	0.50	0.81	0.90	0.71	0.82	0.79	0.77
	2	0.97	0.77	0.84	0.91	0.83	0.86	0.86	0.53
	3	0.99	0.82	0.79	0.87	0.61	0.84	0.82	0.40
	4	0.98	0.44	0.77	0.89	0.70	0.86	0.77	0.29
	5	1.00	0.81	0.80	0.89	0.80	0.80	0.85	0.45
	6	0.99	0.80	0.80	0.90	0.73	0.80	0.84	0.33
Average		0.99	0.69	0.80	0.90	0.73	0.83	0.82	0.42

in Table IV shows the difference between the average of the probability of reaching the goal with *coevU* and *fixed* methods, respectively. An improvement of 40%, on average, has been obtained. The solution found in the objects configuration 1 is quite surprising. It has obtained the same navigation capability than those solutions evolved in objects configurations with obstacles. This fact points out that the learning process is independent of the objects configuration used in the *coevU* case.

The above results probe that the UC is able to improve the capability of reaching the goal. However, in order to probe the generality of the controllers, the probability of reaching the goal will be calculated. The best controller obtained with experiments *fixed* and *coevU* are compared (experiment 10 of Table I and experiment 2 of Table III). The probability of reaching the goal is calculated by quadrant. The objects environment has been divided into four quadrants, as can be seen in Figure 11.

If the solution is a general one, then the probability of reaching the goal in every quadrant, in all different training objects configurations, must be similar. The generalization capability has been considered as the probability of reaching the goal. The calculation of this probability has been carried out, making 1,000 executions over all object configurations in Figure 1. Each execution has a different starting position and orientation of the robot, randomly generated. The probability of reaching the goal is summarized by quadrant in Table V for *fixed* experiments and Table VI for *coevU* experiments.

The comparison of Tables V and VI shows some important results. In *fixed* experiments, the probability of reaching the goal is related with the quadrant in which the robot starts off, but it also depends on the objects configuration used. Thus, the second quadrant is the best to start moving in the objects configuration

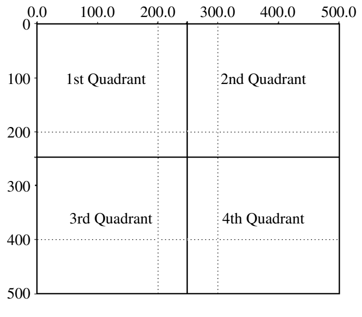


Figure 11. Quadrant decomposition of an object environment.

Table V. Probability to reach the goal, by quadrant, of the best controller evolved in experiment *fixed* number 10.

	Validation object configuration					
	2	3	4	5	6	Average
1st Quadrant	0.5	1.5	14.8	4.7	49.5	14.2
2nd Quadrant	56.8	11.4	17.9	37.2	14.0	27.5
3rd Quadrant	12.8	41.8	7.0	5.2	70.2	27.4
4th Quadrant	47.6	7.0	1.4	10.2	8.9	15.0

Table VI. Probability to reach the goal, by quadrant, of the best controller evolved in experiment *coevU* number 2.

	Validation object configuration					
	2	3	4	5	6	Average
1st Quadrant	95.5	97.9	97.7	91.0	95.4	95.5
2nd Quadrant	98.8	95.4	98.5	96.5	96.9	97.2
3rd Quadrant	95.6	99.6	96.5	93.6	97.7	96.6
4th Quadrant	99.7	98.8	96.3	95.7	97.7	97.6

2, but it is bad in the objects configuration 10. This circumstance absolutely changes in controllers evolved with UC. Table VI shows that the differences of the probability of reaching the goal between objects configurations and quadrants are irrelevant. These controllers have a general behavior; they reached the goal from all starting positions.

Figures 12 and 13 show the executions performed to calculate the results shown in Tables V and VI. This graphical representation allows us to extract some qualitative conclusions. In Figures 12 and 13, filled dots show the starting positions from which the robot reached the goal. On the contrary, if the robot did not get the goal position, an empty dot in the starting position is plotted. The selected objects configurations are 2, 3, 4, 5, and 6 of Figure 1.

It can be observed in Figure 12 that a region exists, different in every environment, in which the probability of reaching the goal is highest. In the best case, Figure 12(e), in this objects configuration, the evolved controller has the highest probability of reaching the goal and corresponds to the environment used in the learning process. The preferred starting position was also the same as the one used in the learning process (left-bottom). With a slight modification of the starting position, the robot is not able to reach the goal. This result indicates the poor generalization reached due to the overadaptation problem. The analysis of *fixed* type experiments shows two main problems: the

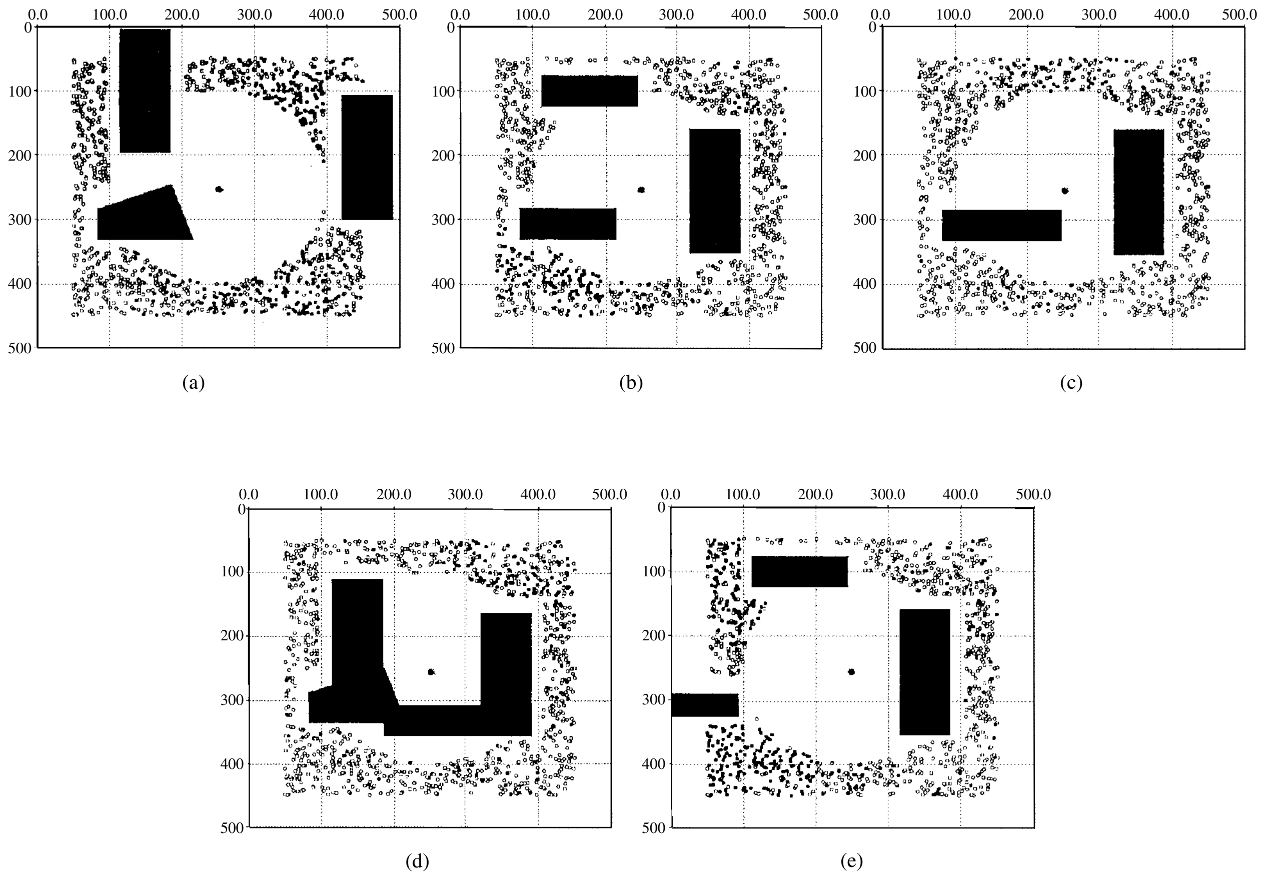


Figure 12. Validation of the best controller obtained in *fixed* experiments.

overadaptation problem and the quality of the solutions that depends on the training examples set. Thus, the necessity of a new evolutive algorithm, the UC, is justified.

On the other hand, following the UC method, the learned controller shows a similar behavior in all validated environments. In this way, there is no overadaptation of the controller; the performance of the controller is not significantly better in the training environment. In Figure 13, it can be appreciated how the robot, the best controller of experiment 2 evolved with *coevU*, reaches the goal from almost all starting positions, independently of the region from which it starts. UC evolves a general controller by using just one object configuration. This is a very different result if compared with the best controller obtained in the *fixed* experiment.

Taking a good look at the path followed by the best controller obtained in both kinds of experiments. This conclusion can be reaffirmed. In Figures 14(a) and 14(b) the paths of the robots have been plotted. This can be used as a qualitative measure of the navigation skills of the best controller obtained in the different types of experiments.

In Figure 14(a) the object avoidance is very poor and the controller crashes into the wall and the objects. Furthermore, only twice the goal was reached under similar circumstances; there are two objects on the right and a corridor between them. This was the configuration used in the learning process. The learning method has enabled the controller to reach the goal with minimal time and path; the solution is optimal to the learning situation. When the starting conditions are changed slightly, then the controller is not able to navigate successfully. It is easy, just by seeing the behavior of the robot, to deduce which objects configuration was used in the learning process: the objects environment where the robot got the highest success. This proves that the generalization level achieved was quite poor.

Figure 14(b) shows that the object avoidance is quite good, but the path obtained is not optimal. In

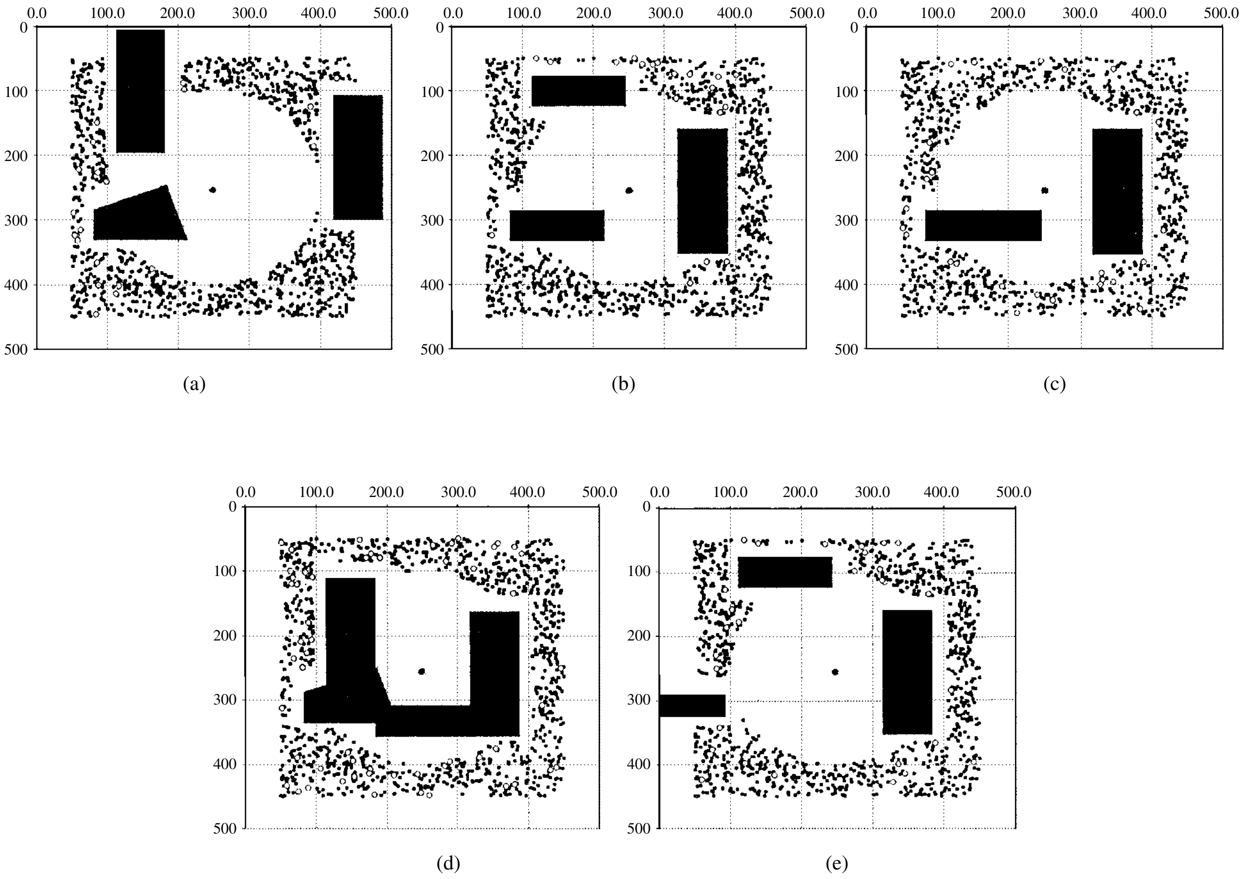


Figure 13. Validation of the best controller obtained so far in *coevU* experiments.

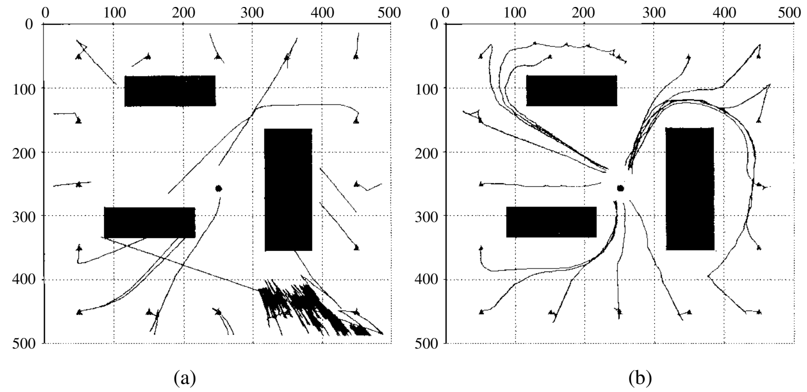


Figure 14. Path followed by the best robot controller. (a) *fixed* (b) *coevU*. Small triangles show the robots' starting positions (the orientation is not drawn) and the robots' paths are represented with lines.

many cases, there are shorter paths to avoid the objects. This is an expected result; the method and the fitness function do not include any specific information to increase the evolution pressure in order to obtain optimal solutions. The controller has achieved right associations between the sensorial inputs and wheel velocities, the proximity sensors provide avoidance of walls and obstacles, and the goal distance sensors permit the guidance to the goal. With these navigation skills, the robot can move and find the goal in every objects environments.

6. CONCLUSIONS

This article has presented the application of a new co-evolutive method, Uniform Co-evolution, to the evolution of Khepera controllers implemented as feed-forward neural networks without hidden layers. The results presented herein show that general behaviors are more likely to be generated through Uniform Co-evolution than through $(\mu + \lambda)$-ES. The solutions obtained with classical ES were overadapted to the training examples. The Uniform Co-evolution evolves a general obstacle-avoidance behavior as the result of a simultaneous evolution of robot controllers and environments. The co-evolution fulfills an incremental evolutionary process, smoothing the convergence rates.

Future research will be devoted to study of the generality of the Uniform Co-evolution method and the incorporation of the domain information to enhance the robot's navigation skills in order to obtain optimal solutions.

REFERENCES

1. R.A. Brooks, Intelligence without representation, Artif Intell 47 (1991), 139–159.
2. S. Ishikawa, A method of autonomous mobile robot navigation by using fuzzy control, Adv Robot 9:(1) (1995), 29–52.
3. V. Matellán, C. Fernández, and J.M. Molina, Genetic learning of fuzzy reactive controllers, Robot Autonomous Syst 25:(1–2) (1998), 33–41.
4. J.M. Molina, A. Sanchis, A. Berlanga, and P. Isasi, An enhanced classifier system for autonomous robot navigation in dynamic environments, Intell Automat Soft Comput 6:(2) (2000), 113–124.
5. A. Sanchis, J.M. Molina, P. Isasi, and J. Segovia, RTCS: A reactive with tags classifier system, J Intell Robot Syst (in press). 1999.
6. F. Mondada and P.I. Franzi, Mobile robot miniaturization: A tool for investigation in control algorithms, Proc 2nd Int Conf on Fuzzy Systems. San Francisco, CA, 1993.
7. Rechenberg, Evolution Strategy: Nature's way of optimization. Optimization: Methods and applications, possibilities and limitations (Lecture Notes in Engineering), H.W. Bergmann (Editor), Springer, Bonn, 1989, pp. 106–126.
8. H.P. Schwefel, Numerical optimization of computer models, John Wiley, New York, 1981.
9. O. Miglino, H. Hautop, and S. Nolfi, Evolving mobile robots in simulated and real environment, Artif Life 2 (1995), 417–434.
10. R. Axelrod, The evolution of cooperation, Basic Books, New York, 1984.
11. R. Axelrod, Evolution of strategies in the iterated prisioner's dilemma, Genetics algorithms and simulated annealing, Davies (Editor), Morgan-Kaufman, San Francisco, 1989, pp. 32–41.
12. K. Lindergren and M.G. Nordahl, Artificial food webs, Artificial life III, Addison-Wesley, Reading, MA, 1994, 73–103.
13. W.D. Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure. Artificial life II, C.G. Langton (Editor), Addison-Wesley, Reading, MA 1991, pp. 313–324.
14. J. Paredis, Coevolutionary computation, Artificial life 2, Addison-Wesley, 1996, pp. 355–375.
15. C.D. Rosin and R.K. Belew, New methods for competitive coevolution, Evolutionary Computation 5 (1997), 1–29.
16. P.J. Angeline and J.B. Pollack, Competitive environments evolve better solutions for complex tasks, Proc Fifth Int Conf on Genetic Algorithms, S. Forrest (Editor), San Mateo, CA, Morgan-Kaufmann, 1993, pp. 264–270.
17. L. Sommaruga, I. Merino, V. Matellán, and J.M. Molina, A distributed simulator for intelligent autonomous robots, 4th Int Symp on Intelligent Robotic Systems-SIRS96, Lisboa, Portugal, 1996.
18. P.J. McKerrow, Introduction to robotics, Addison-Wesley, Reading, MA, 1991.
19. Berlanga, A. Sanchis, P. Isasi, and J.M. Molina, Neural networks robot controller trained with evolution strategies, Proc 1999 Congress on Evolutionary Computation, CEC99, 1999.
20. V. Braitenberg, Vehicles: Experiments in synthetic psychology, MIT Press, Cambridge, MA, 1984.
21. D. Goldberg, Genetic algorithms in search, Optimization and machine learning, Addison-Wesley, New York, 1989.