Considerations in the Application of Evolution to the Generation of Robot Controllers

J. Santos[1] , R. J. Duro[2] , J. A. Becerra[1], J. L. Crespo[2], and F. Bellas[1]

[1] Dpto. Computación, Universidade da Coruña, Spain
santos@udc.es, {ronin, fran}@cdf.udc.es
[2] Dpto. Ingeniería Industrial, Universidade da Coruña, Spain
richard@udc.es, crespo@cdf.udc.es

Abstract

This paper is concerned with different aspects of the use of evolution for the successful generation of real robot ANN controllers. Several parameters of an evolutionary/genetic algorithm and the way they influence the evolution of ANN behavioral controllers for real robots have been contemplated. These parameters include the way the initial populations are distributed, how the individuals are evaluated, the implementation of race schemes, etc. A batch of experiments on the evolution of three types of behaviors with different population sizes have been carried out in order to ascertain their effect on the evolution of the controllers and their validity in real implementations. The results provide a guide to the design of evolutionary algorithms for generating ANN based robot controllers, especially when, due to computational constraints, the populations to be used are small with respect to the complexity of the problem to be solved. The problem of transferring the controllers evolved in simulated environments to the real systems operating in real environments are also considered and we present results of this transference to reality with a robot which has few and extremely noisy sensors.

Keywords: Evolutionary robotics, genetic algorithms, evolutionary strategies

Corresponding Author:

Prof. Dr. Richard J. Duro
Grupo de Sistemas Autónomos
Escuela Politécnica Superior
Universidade da Coruña
c/ Mendizábal s/n
15403 Ferrol (La Coruña)
Spain

Phone: 34-981-337400 ext 3281
Fax: 34-981-337410
e-mail: richard@udc.es

Considerations in the Application of Evolution to the Generation of Robot Controllers

J. Santos[1], R. J. Duro[2], J. A. Becerra[1],J. L. Crespo[2], and F. Bellas[1]

1 Dpto. Computación, Universidade da Coruña, Spain

santos@udc.es, {ronin, fran}@cdf.udc.es

[2] Dpto. Ingeniería Industrial, Universidade da Coruña, Spain

richard@udc.es, crespo@cdf.udc.es

Abstract

This paper is concerned with different aspects of the use of evolution for the successful generation of real robot ANN controllers. Several parameters of an evolutionary/genetic algorithm and the way they influence the evolution of ANN behavioral controllers for real robots have been contemplated. These parameters include the way the initial populations are distributed, how the individuals are evaluated, the implementation of race schemes, etc. A batch of experiments on the evolution of three types of behaviors with different population sizes have been carried out in order to ascertain their effect on the evolution of the controllers and their validity in real implementations. The results provide a guide to the design of evolutionary algorithms for generating ANN based robot controllers, especially when, due to computational constraints, the populations to be used are small with respect to the complexity of the problem to be solved. The problem of transferring the controllers evolved in simulated environments to the real systems operating in real environments are also considered and we present results of this transference to reality with a robot which has few and extremely noisy sensors.

1. Introduction

In the late eighties and early nineties, some authors, such as Harvey et al. [7], Cliff et al. [4] and Beer and Gallagher [2], have proposed artificial evolution as means to automate the design procedure of behavior based robot controllers. Many authors have taken up this issue and have developed different evolutionary mechanisms and strategies in order to obtain controllers for the autonomous operation of robots in structured and unstructured environments. Most of the algorithms employed were specific versions for the problems in hand, and, in the literature, there are very few data which can be taken as general or even helpful for contemplating the design of ANN based robot controller evolvers.

The most usual structures employed for the implementation of behavior controllers are Artificial Neural Networks (ANNs) [2][7][15][18]. ANNs form a powerful distributed processing model with very interesting characteristics for their use in autonomous robotics, such as their fault tolerance, noise tolerance (crucial property because of the presence of noise in real environments) and the possibility of using the traditional connectionist learning algorithms for obtaining the ANN in ontogenetic time or the possibility of combination of evolution with learning. Another advantage is derived from the fact that the weights and nodes are low level syntactic primitives, well below the semantic level [3], and, as Nolfi et al. [15] argue, the primitives considered by the evolutionary process must be of the lowest level possible so that undesirable selections produced by a human designer are avoided.

Evolution of ANNs as robot controllers imply a series of problems: on one hand, the evolution of ANN based structures is very prone to epistasis problems; on the other hand, the evaluation of

the controllers is necessarily very indirect, as one does not know how good a controller is until the robot has interacted with the environment for a relatively long period of time. As a consequence, the computational load falls mainly on the simulation of the live of each individual in its environment, generally forcing the designer to work with small populations of individuals so as to make the evolutionary process as efficient as possible. This obviously leads to premature convergence problems, where, because of the small number of individuals in the populations, it is very easy for one individual to dominate and lead to a population consisting of replicas of this individual.

Some authors, such as Salomon [18], have employed the evolutionary strategy alternative because it aids in problems where the level of epistasis, and consequently, of deceptivity is high. That is because the fitness of a gene within the chromosome is strongly dependent on the value of other genes and, in this case, where the chromosomes represent the weights or other distributed parameters of ANNs, there is usually a large correlation between the genes in terms of fitness. The general trend, however, has been to make use of the classical Genetic Algorithm (Gas) [4][6][7][16] or small variations of it, and most authors have concentrated mostly on different aspects of the adaptation of the evaluation function, without really going into other parameters affecting the evolutionary algorithm.

In this paper we are going to address some of the solutions adopted by our group for the successful generation of controllers for real robots operating in real environments and which have been integrated in *SEVEN* (Simulation and EVolution ENvironment), our general purpose automatic robot controller generator. To reach these conclusions, several experiments were

carried out in the generation of controllers for three tasks considering different values for the parameters that influence the evolutionary process.

2. Experimental Setup

In the examples we present throughout this article, the robot employed is a “Rug Warrior”. It is a small (18.5 cm diameter), simple and cheap circular robot. It has two DC motors, two infrared emitters and one binary receiver, two photosensors, one pyrosensor, three bumper sensors and two optical encoders. The sensors and actuators are very low quality, very noisy and imprecise, which is a plus when trying to obtain robust behaviors in the hardest possible conditions. The response of these sensors and actuators employed in the simulator were obtained by measuring the real response of the real sensors and actuators under different conditions such as different levels of ambient light, etc.

As indicated in the introduction, for the purpose of evolving the controllers to be used in the real robot, we have made use of *SEVEN* where simulation and evolution have been integrated in the same environment. *SEVEN* allows running complete evolutionary strategies [19][17], genetic algorithms [8], or a combination of both. The evolutionary algorithm can evolve a single low level behavior, a high level behavior that makes use of the low level behaviors we provide it with or co-evolve both simultaneously, as explained in [1]. The parameters of the evolutionary algorithm used, except for those under study, were the same for all the experiments. Tournament selection, nonlinear mutation function, and replacement of all the parents by the offspring, except for the best 3%, were contemplated. The encoding scheme employed for the ANNs that implement the controllers is just a direct genotypic representation of the phenotype in terms of

synaptic weights and/or delays and the characteristics of the nodes, such as slope or bias of the activation functions. Regarding the fitness criteria we employed a global energy based fitness criteria, although in one of the sections we comment on the merits of different possibilities.

The behaviors obtained in simulation with the evolutionary methodology must work in the real environment. Usually, there is loss of performance in this transference and, to minimize this "reality gap", noise is traditionally applied to sensors and actuators. In addition, we have applied in our simulations three different types of noise, we have called generalization noise, systemic noise and temporal noise. These types of noise in simulation permit traversing the reality gap and obtaining the same behaviors in the real robot and environment as the ones obtained in simulation.

The experiments have considered three types of behaviors, light catching, wall following and homing. The light catching behavior is the easiest one. The robot must use its light sensors (two of them) in order to follow and catch a target and the light sensors are quite good. This does not mean that this behavior is the simplest in terms of the fitness function, as the only information the evolutionary algorithm employs in order to obtain the goodness of a controller is if it caught the light. The wall following behavior is more difficult, especially for the Rug Warrior robot, as it must make use of its binary unreliable infrared sensors to implement the behavior and, in general, in order to achieve valid solutions the networks employed must be able to take into account temporal aspects, as will be explained later on. Anyway, in terms of the fitness function, it is the behavior with the largest amount of information of what happened during the robot life. Finally, the homing behavior consists in the robot finding and reaching home, represented by a flashing light, while avoiding traps, represented by fixed lights. This behavior necessarily

implies the consideration of temporal aspects and is the most complex one for the robot to evolve.

Regarding the networks evolved, they all consisted of two inputs (whether from IR or light sensors), six nodes in the hidden layer and two outputs. All of them included trainable delays in the synapses between the input layer and the hidden layer. This was done so as to preserve the same dimensionality of the search space for the three problems, and thus, the complexity of the problem is related to the fitness function and the error surface it generates. As we will show later, this rule was broken in just one case where we wanted to simplify the search space in order to make a point and we took out the delays.

The complexity of the problem is determined, on one hand, by the size of the chromosomes to be evaluated, that is, the dimension of the solution space and, on the other, by how abrupt the error surface determined by the fitness function is. Thus, if we have a smooth gently curving solution space, with solutions which are gradually reached even when starting from points far from them, then the problem presents less complexity than if the optimal solutions are located in very narrow peaks surrounded by areas that provide very little information on their location.

In the following sections we are going to present and discuss the different experiments. It is evident that the results presented are just a selection from the hundreds of experiments carried out, but they do represent the behaviors that were consistently obtained from the whole series. We have basically run experiments modifying the distribution of the initial population in the search space, the division of the population into disjoint races, modifying the crossover probability (Pc), considering different evaluation strategies and several simulation conditions.

3. Distribution of the Initial Population in the Solution Space

Generally, the starting population of an evolutionary process is initiated randomly. Thus, if the number of individuals is not very large, the individuals may cluster around certain locations in the search space, with no individuals searching other areas. This is a typical problem of evolutionary robotics, where the number of individuals is usually small due to computing constraints.

To address this problem, we have considered different distribution strategies. The initial individuals were generated as a random distribution, as a diagonal distribution or as a grid. In order to obtain a diagonal distribution the individuals in the population were evenly spaced along one of the hyper-diagonals in the solution hyperspace, consisting of one dimension per gene in the chromosomes representing the controllers. A grid distribution was obtained by evenly spacing the individuals as a hyper-grid in the solution hyperspace.

Analyzing these three distributions, it can be observed that the diagonal distribution is the one that provides the maximum variety in gene space. That is, as all the chromosomes lie along a hyper diagonal of the search space and thus, for a given gene, say gene number 2, the initial values are different in all the chromosomes. Consequently, when crossover takes place, a large portion of the search space can be evaluated through crossover. On the other hand, in a perfect grid distribution, even though initially it seems that the search space is explored more thoroughly, from the point of view of initial gene values for a given gene the number is much smaller than for a diagonal distribution. A chromosome only differs in one gene from its

neighbors. In addition in the case of a perfect grid like distribution crossover will only lead to the same chromosomes over and over, as any recombination of chromosomes in the population will lead to chromosomes that are already part of the population, and only through mutations will we ever be able to explore other portions of the search space. Finally, a random distribution could be taken, on average, as an imperfect grid like distribution, thus presenting less variety than a diagonal distribution but, on average, more than a grid.

Several different tests were carried out for the generation of wall following, light catching and homing behaviors using a Rug Warrior robot and considering different population sizes, between 20 individuals and 800 individuals. The results clearly show that when the populations are large with respect to the complexity of the problem all the distributions perform equally well. When the populations are small as compared to the complexity of the solution space, then it becomes more optimal to make use of initial population distributions, such as the diagonal one, which provide a more structured exploration as an initial guess.

Figure 1a displays the average of the best individual´s fitness for 10 evolutions of the homing behavior. The population consisted of 1 race of 400 individuals, we made use of a tournament window of 100 individuals which were evaluated 32 times and Pc=0.5 (crossover probability). The thick line corresponds to a diagonal initial distribution, the thin line to a random distribution and the remaining line to a grid type distribution. It is clear in this figure that a diagonal distribution allows the evolutionary process to perform better on average. In fact, if we take a look at figure 1b, where we represent the best evolutions obtained for each of the distributions, it can be seen that a diagonal distribution requires less than half the number of generations than the random distribution in order to achieve comparable fitness values. This is so because a diagonal

distribution for small populations uses the small number of chromosomes available in a more efficient manner to obtain as much information as possible on the solution space. In fact, this information is rapidly compiled into an efficient solution through crossover. The same behavior was obtained in many different trials for the behaviors commented, although, in less complex behaviors, the difference is smaller. On the other hand, in figure 1a it can also be seen that the grid distribution performs worse than the random one. This can be attributed to the fact that this type of distribution, as commented before, makes very poor use of crossover.

Another aspect to be taken into account is that the initial population may be taken as a single unit to be evolved or may be divided into races. In the second case, some individuals of each race (usually the best) migrate every so many generations to the other races. This provides for a protection against premature convergence. If one of the races prematurely converges to a super-individual, migrating good, usually different, individuals from other races increases the variety of individuals in this race. In addition, if the races are created in such a way that they encompass disjoint areas of the search space, the search performed is initially more efficient than in the case of a single population. Clearly, after a few generations, some subpopulations may contain individuals outside their assigned search spaces due to mutations but, in general, this procedure implies performing a more exhaustive search of the solution space and thus tends to prevent premature convergence. Obviously, if a large single population finds the right track to the best solution fast, it has more resources in terms of individuals to follow this track, and thus the optimal solution is reached faster. On the other hand, the probability of finding this right track is much enhanced in the case of having different races.

In this sense, as shown in figure 2 for the case of an evolutionary process with 800 individuals representing light catching controllers, the evolution using 8 different races (represented by a thick line) is slower than the evolution corresponding to the same 800 individuals taken as a single population (thin line). Figure 2a corresponds to the average of ten evolutionary processes of the light catching behavior using a diagonal distribution with 32 evaluations for each individual and Pc=0.5. In the case of 8 races, the tournament window was 30 individuals and in the case of a single population the tournament window was 200 individuals. Figure 2b represents the best evolutions obtained in the same conditions in both cases.

Figure 3b displays the difference between best and worse evolutions for the case of 8 races and figure 3a provides the same information for the evolution without using a race scheme. These last two figures clearly show that, despite the slower evolution in the case of races, the process is much more constant, with less oscillations and hardly any case in which a satisfactory solution is not found. This can be deduced from the spread of evolutions determined by the limits provided by the best and worst evolution, in other words, the area bounded by them. These results are obviously valid if we maintain selective pressure constant; otherwise, if the selective pressure is increased, having more races would not provide any evolutionary advantage.

4. Mutation and Crossover

Two of the most studied operators with a bearance on evolutionary algorithms are the mutation operator and the crossover operator. Here, we have considered their relative influence on the results provided by the behavior generator. In the tests performed, and in order to maintain consistency, the sum of the crossover probability (Pc) and the mutation probability (Pm) is 1.

In evolutionary strategies there are several ways of approaching mutation. It is possible to fix the maximum mutation beforehand or to include it in the evolutionary process. The first solution may slow down the process either due to low values of the mutation parameter, which implies that it may take a long time to reach the global maximum if it is far in the solution space from the initial individuals, or to large values as in the case when one is close to the global maximum and most of the offspring stray away from it, hindering convergence. The second solution does not present these problems, but it increases the dimensions of the search space, which in some cases may slow down the evolutionary process. For these reasons, we have chosen an intermediate solution, a non-linear probability mutation function. This way, in most cases mutation consists in the addition or subtraction of a small amount, but sometimes large values are obtained, leading to long trajectories in the search space, which accelerates evolution in the first few generations and permits avoiding local maxima or minima.

Evolutionary strategies do not usually implement the crossover operator. However, we wanted to study the effect of including it up to the point where the whole process would become a Genetic Algorithm. We did not think that it was appropriate to completely reject it, as this operator, even in deceptive problems, sometimes produces better individuals and is a good method in order to avoid local maxima or minima as long as the selection method employed does not imply an excessive selective pressure (as can be the case of tournament selection, the one which we have employed). To evaluate its influence on processes with the objective of generating ANN based robot controllers, we have run several evolutions with no crossover, that is, an evolutionary strategy without including the mutation probability in the chromosome, 0.5 crossover probability, and a crossover probability of 1 (genetic algorithm) for the different

problems described. All those offspring generated through crossover were mutated with a mutation probability of 0.02 using random mutation so as to simulate as closely as possible a traditional GA.

The resulting data shows that it is usually a crossover probability of 0.5 the one that provides the most reliable results and the faster evolutions. Figure 4a clearly displays this fact. In this figure we provide results considering 10 evolutions of the light catching behavior, a diagonal distribution, 1 race of 400 individuals, a tournament window of 30 individuals and 32 evaluations for each individual. The evolutions with a crossover probability of 0 (thin line), that is, evolutionary strategies, are much faster in their evolution characteristics than genetic algorithms (normal line), but often get stuck in local minima when using small populations (which is generally the case of evolutions for obtaining robot controllers). Consequently, they do not perform, on average, as well as genetic algorithms. These are much slower in reaching the solutions required of them when populations are small, as they do not include in the initial populations all the genetic data necessary for the crossover operator to recombine it into a final optimal solution and must wait for the mutation operator to generate the appropriate building blocks before recombining them. This recombination capability allows this type of algorithms to avoid becoming stuck in local minima, but at the cost of a much slower evolution. The final results provided by the genetic algorithm are usually better when the complexity of the problem is small, that is, when the sparse sampling of the solution space provides information that is easy to recombine into better solutions. To make the problem simpler we have chosen two possibilities, to consider a different behavior where the robot has more information as it moves around than in light catching behavior, or to consider a simpler encoding of the problem. This is the case displayed in figure 4c (diagonal distribution, 1 race of 20 individuals, tournament

window of 5 individuals and 8 evaluations for each individual, wall following behavior) and, especially, in figure 4d (diagonal distribution, 1 race of 20 individuals, tournament window of 5 individuals and 32 evaluations for each individual, light catching behavior), where the problem is made much simpler by considering ANNs without delays, with just simple synaptic weights.

As a conclusion of this section, we must state that, in general, our results show that the best solution when contemplating these small populations is to apply the combination of an evolutionary strategy and a genetic algorithm. This permits a faster evolution through the use of a better mutation operator and avoiding local minima through crossover.

4. Fitness Evaluation

Whatever the evolution mechanism employed, the controllers corresponding to the different individuals must be evaluated and their fitness obtained. This fitness corresponds to how well the robot performs in an environment during its lifetime. Two different perspectives are possible from the point of view of evolutionary robotics: a local perspective or a global perspective. The first one is to establish for each step of the robot life a goodness of its actions in relation to its goal. The final fitness will be the sum of the fitness values corresponding to each step. This strategy presents many drawbacks as, except in toy problems, it is very difficult to decide beforehand the goodness of each action towards a final objective.

The global approach, on the other hand, implies defining a fitness criterion based not on the goodness of each particular action in the robot life, but on how good the robot was at achieving its final goal. That way, the evolutionary algorithm has a lot more freedom for discovering the

best final controller. This approach presents several problems, but the main one is the credit-apportioning problem. The final fitness value does not directly reflect the goodness of each action taken by the robot to achieve this final fitness, and thus a lot of information that could accelerate the evolution is lost.

There are two main ways in which global fitness can be obtained: external and internal. By external we mean a fitness assigned by someone or some process outside the robot. This evaluator has a lot more information than the robot and can thus produce all kinds of fitness values. This approach is very efficient in some cases where the designer wants a particular task, in a very well defined environment, done. It is not so adequate when these conditions are not met, that is, when the environment is partially unknown or there is noise in the perception of it, or when the robot controller is being evolved in one environment (simulated or not) and employed in another. An extreme example of external global evaluation of the robot behavior is presented by Lund et al. [11]. In this work, as they conclude that it is not clear how to construct a mathematical fitness function for their problem, the authors apply an interactive genetic algorithm for the development of Lego robot controllers and morphology. They make use of neural networks in order to implement the controllers and let a group of children evaluate them. The main limitation they encountered was that of development time due to the interaction characteristics of the evaluation strategy.

The other possible approach is to employ an internal representation of fitness. This is, employ clues in the environment the robot is conscious of and that it can use in order to judge its level of fitness without the help from any external evaluator. A concept often used in order to implement this approach is that of internal energy, which increases or decreases according to a set of rules

or functions related to robot perceptions. The final fitness of the robot can be given by the level of energy at the end of its life. The only intervention of the human designer here is in assigning the relationship between energy increases or decreases and the robot perceptions.

The two main alternatives, that is, local and global fitness functions, are considered in the work of Floreano and Mondada [5]. The authors employ a Khepera robot, and define a fitness function in terms of each movement for a navigation behavior and for a homing task but considering at the same time the recharge level of its battery in its home and the rate of discharge when away from it. They also consider grasping, but this behavior is defined only in terms of the number of balls grasped. The drawback of the last approach, as commented by the authors is the time required, specially if the evolution is carried out in the real robot as the authors do.

We provide in this section some results using these considerations for obtaining robot behavior controllers in the form of ANNs. In order to do this, we will include some examples using one of the most ubiquitous behaviors in the autonomous robotics literature: the wall following behavior. The behavior consists in the robot finding and following the walls of an enclosure at the highest possible speed; minimizing the distance to the wall it is following and avoiding collisions.

The behavior is usually implemented in robots where the sensors employed in this task provide values in a range that is large enough for the robot to be able to distinguish between when it is approaching a wall or going away from it. The biggest problem found when obtaining these behaviors is caused by the presence of noise in the sensors. Also, in our case, the binary nature of the Rug Warrior's infrared sensors makes impossible to decide when we are

approaching or going away from an object without taking into account the previous instants. Because of this, we use input neurons that detect transitions in the sensed values and that permit that differentiation.

Several evolutionary processes were run to obtain robot controllers for a wall following task, making use of the different approaches to the fitness function mentioned in the previous paragraphs. A comparison of the resulting controllers for the same task is presented in figures 5 and 8. In the right side of figure 8 we display, as a reference, the best case of an energy based internal fitness function. In this case fungi-like food was attached to the bricks of the walls. When the robot reached a brick it would eat the food and the fungi would slowly grow back. The fitness of the individual is calculated as its final energy after 1000 iterations in the world averaged over eight lives. The operation of the robot is quite satisfactory, it has evolved a behavior that really allows it to follow walls in a very efficient manner considering that its infrared sensors (which are the only ones it can use for this task) are binary and very noisy.

In figures 5a, b, and c we display the operation of behavior controllers evolved using the same conditions as before but, in these cases, with different external fitness functions. The general definitions of these functions are:

Left:

```
Fitness = 0
For I = 1 to n_steps
        If sensing_with_IR
                Fitness = fitness + MAX(vel_left_motor,vel_right_motor)/max_vel_motor
```

Middle:

```
Fitness = 0
For I = 1 to n_steps
```

```
        If sensing_with_IR
                Fitness = fitness + (vel_left_motor + vel_right_motor)/(2*max_vel_motor)
```

Right:

```
Fitness = 0
For I = 1 to n_steps
        If sensing_with_IR
                Fitness = fitness + MIN(vel_left_motor,vel_right_motor)/max_vel_motor
```

If we only employ as a fitness function the number of wall sensing events with the IR sensors, the robot evolves to a controller that goes in a straight line to a wall and then, when it senses the wall, it stops and it remains stopped, continuously sensing the same brick. On the other hand, if we make use of the speed of the wheels of the Rug Warrior, the behaviors obtained are different depending on how these values are combined.

If we consider the maximum speed of the two motors, the robot evolves a behavior that takes it to a wall and, once it has reached it, performs 360º turns (figure 5a). This motion is very easy to obtain in the evolutionary process and very hard to improve on, because with this motion there are never any collisions and when the robot is near a corner it is sensing almost continuously.

Through an evaluation by means of the average speed of the two motors, the controller obtained implies that the robot tries to sense the wall performing very soft turns. Doing that without crashing is not easy for the Rug Warrior. In fact, when the robot looses the wall, it goes in a straight line as fast as possible trying to find another, as shown in the central part of figure 5.

Finally if we consider the minimum speed, the problem is very similar. Here, the robot shown in figure 5c, tries to follow the wall in the straightest line possible and, periodically, turns towards the wall to verify that it is still “there”.

From these results it is clear that to establish a local fitness function that allows the robot to perform the behavior we desire is not simple at all, it is usually a question of trial and error with very unpredictable results. On the other hand, internal energy based functions allow the robot to obtain by itself the desired behavior simply using clues from the environment. It must be mentioned that using environmental clues may lead, in some cases, to a much more difficult evolution, but this difficulty is compensated by the fact that the final behaviors obtained are usually much better in real robots.

5. Noise in Simulation

For a simulation to be appropriate for the transference of behaviors developed using it to the real world, it must meet some criteria, such as those established by Jakobi et al. [10], which usually imply handling different levels and types of noise. In the evolutionary robotics literature several authors have proposed different types of noise that should be applied to the simulated sensors, actuators or environment. Traditionally, the noise applied to the simulated sensors and actuators consisted in small random variations [12][13] or variations with a gaussian distribution [6] in the values sensed or applied. Another type of noise used, termed “conservative noise” by the authors [13], consists in simulating small alterations in the positions of the objects perceived by the sensors, which can be translated into large variations in the sensed values depending on the sensor model. This type of noise simulates the different behaviors exhibited by the sensors

depending on “differences in the illumination of the objects, shadows, or because of slight physical differences between objects of the same type” [13], and it is especially useful with some types of sensors such as infrared sensors. For example, with this type of noise, the authors find the best match in terms of fitness decay from the simulation for obstacle avoidance while moving as fast as possible behavior with the Khepera robot.

The authors in [14] introduce another variant of noise in the simulation: whether or not to introduce noise depending on the value of a random variable. If a change was to be made, it was accomplished by doubling the values of the action that the output variables generated. The authors find the best match between simulation and real performances with 22% noise level (the robot actions were doubled 22% of the time) for a wandering behavior with a LEGO robot.

In [9] different levels of noise (zero noise, observed noise and double observed noise) were applied in their simulation of the Khepera robot for obstacle avoidance and light following. Their conclusion is that “simulation to situation correspondence seems to be maximized when the noise levels of the simulation have similar amplitudes to those observed in reality”. The authors define the observed noise as a gaussian distribution of noise with standard deviation equal to that empirically derived from experiments. Thus, if too little or too much noise is injected in the simulation, then, according to the authors, “different classes of behaviors become available which, while acquiring high fitness in simulation, necessarily fail to work in reality”.

These concepts have been applied to the simulations employed in this work for the evolution of controllers for the Rug Warrior robot. But, in addition to the traditional random noise applied

to values sensed and actuator commands, our simulations include the following three types of noise:

- Generalization noise. It consists in randomly changing some characteristic of the robot or the environment (such as the orientation of a sensor) and maintaining this change for one whole life of the robot. This is necessary, for instance, if the robot is working on battery power, as different levels of battery charge imply different speeds for the wheels of the robot when presented with the same command.
- Systemic noise. It is a specially important type of noise that has to do with defects or particular traits in the operation of the real robot. It was implemented through the inclusion of variations of traits in different lives of the robot.
- Temporal noise. This type of noise is used in order to obtain behaviors that are tolerant to variations in the time elapsed between events. It consists in modifying the temporal position of the events the robot perceives relative to each other.

Obviously, noise makes obtaining a given behavior slower and more difficult. It may cause the environment to be perceived very differently in each evaluation of the robot, forcing it to obtain compromise conservative solutions.

In order to test these premises we again make use of a wall following behavior. In figure 6 we display the behavior obtained in simulation under different conditions. In the first case (6a), we have only included random noise (15%) in the sensors and in the actuators. Even though these levels of noise allow the robot to operate correctly in environments that present up to the same levels of noise the operation in environments closer to the real one in terms of noise (6c) is

deficient. The behavior of figure 6b was obtained evolving the robot with an environment that considers all types of noise except the temporal one: random noise in sensors and actuators, generalization noise (variations in the orientations of the sensors and on the charge of the battery) and systemic noise by which the left wheel of the robot would operate at random speeds down to 10% below its specified value.

From an evolutionary point of view, to evolve a controller when the environment is so noisy becomes more difficult, as can be seen in figure 7. This is because depending on the particular noise parameters of each robot's life, this robot can be evaluated to better or worse, and thus, more robust compromise solutions to the problems it finds must be obtained.

The robot performs the behavior adequately (6b), but the operation is not smooth as a consequence of the high levels of noise present. Figure 6d displays the operation of the robot on the closest environments to real life operation we have implemented, where in addition to the types of noise previously specified, we have added temporal noise. It can be clearly seen that the robot performs the behavior we require but, as it was evolved without considering temporal information, we see that there are some types of curves, whatever the number of hidden layers employed, that the robot could not handle in a satisfactory manner.

In figure 8 we also consider the effect of the handling of temporal information by the robot controller. As we have said, we use input neurons in a feedforward neural controller that detect transitions in the sensed values, that is, habituation neurons. These neurons are activated if there is a change in its inputs and when there is no change, the output slowly decays. In the left part of figure 8 we show the evolution and obtained behavior if we do not consider temporal

information in the neural controller of the Rug Warrior. We include in this behavior the need of finding a wall to follow in order to make it more interesting. Obviously, this makes obtaining the behavior much more difficult. Thus, when the robot is not sensing anything it must select a trajectory that leads it to a wall but this trajectory cannot be a tight turn (closed circle), or it would never find the wall. When the robot leaves a wall that it is following and is not sensing anything, it must perform a tight turn in order to return to the wall. Obviously, these two conditions are not compatible and the consequence is that a robot controlled by a regular feedforward neural network usually presents very poor wall following behavior, generally bouncing off walls. In the bottom left part of figure 8 we show how the fitness (the energy at the end of its life, that is, the fungi the robot has eaten) achieved a value that is lower than the two cases that consider temporal information in the form of habituation neurons at the inputs of the neural controller.

The central figure shows the behavior and evolution when habituation neurons were used, but without temporal or any type of generalization or systemic noise, only with random noise in sensors and actuators. This behavior is better than the previous one, because now the robot controller can discern when it goes away from a wall and where the wall is, allowing it to follow the walls in a closer manner. The problem is that the behavior thus obtained cannot traverse the reality gap to behave adequately in the real world, where all those types of noise are present. For the controller in the right part of figure 8 we have now introduced, in the simulation, all the types of noise previously commented, including temporal noise. In this case, the evolution is slower and the fitness moves towards a value that is always lower than in the previous case. The reason is that the neural controller must achieve compromise solutions to consider all the noise present, which causes the environment to be perceived very differently in each evaluation of the robot. In

addition, as noise is random, the evaluation conditions for two individuals in a population may not be fair, one individual may find itself in more difficult conditions than another. Also, the best individual in one generation may obtain a much lower fitness value in the next if the environments, due to noise, are perceived in a very different manner. This problem will be especially relevant in the first few generations, and to prevent the algorithm from prematurely converging to a sub-optimal solution, the number of evaluations of each individual must be made much larger until a certain level of evolution has been achieved.

Finally, to test the capacity of the inclusion of all the types of noise considered in the simulation to traverse the reality gap, in figure 9 we display the behavior of a compound wall following behavior (monolithic wall following and escape from collisions) in the simulated and real environment. The monolithic wall follower was coevolved with a high level controller, which switches between that wall following controller and one that escapes from collisions. In this case the simulation works with all types of noise, including temporal noise, and now the behavior is robust enough to perform its task correctly in the real environment.

6. Conclusions

In this work we have studied the impact of different parameters of an evolutionary algorithm on the performance of ANN based robot controllers obtained using it. These parameters included the crossover probability, going from an evolutionary strategy to a completely genetic algorithm, the distribution of the initial populations, both in terms of the initial values for the genes making up the chromosomes and in terms of the distribution of the population into different races. The results indicate that for cases where the populations consist of reduced numbers of individuals,

which is usually the case when evolving robot controllers due to computational constraints, if the complexity of the problem is low, purely genetic algorithms obtain the most reliable results, but when this complexity increases, the best results are obtained through a controlled mixture of genetic algorithms and evolutionary strategies.

Regarding the distribution of the initial individuals, it has turned out that when populations are small, a diagonal distribution accelerates evolution with respect to a random one, providing a more efficient initial evaluation of the search space. Finally, the division of the population into different races, even though it usually makes evolution slower, provides a mechanism for improving the reliability of the evolutionary process. In fact, when an adequate number of races are used, the difference in performance of several runs of the evolutionary process becomes very small.

We have also considered the possibility of using different types of fitness functions. The best results for obtaining robot controllers in uncertain environments and using robots with relatively unreliable sensors have been achieved by internally defined global energy functions. That is, it has been deemed better to leave clues in the environment that would help the evolutionary process evaluate the best strategy to achieve the desired objective than to evaluate each action of the robot an outside source.

In addition, we have considered different types of noise that applied to simulations permit the behavior controllers thus obtained to operate without any problem in the case of real robots in real environments. That is, these types of noise, such as temporal noise or systemic noise permit traversing the reality gap between the simulated and real behaviors.

Summarizing, through the appropriate selection of certain parameters of the evolutionary processes involved in the generation of ANN based controllers for autonomous robots, the resulting controllers can be obtained with a high level of quality reducing the computational complexity and cost while achieving the best possible results. All these aspects make the evolutionary robotics methodology an efficient alternative for the automatic generation of real robot controllers.

Acknowledgements

This work was funded by the Xunta de Galicia under project PGIDT99PXI10503A.

References

[1] J. A. Becerra, J. Santos, and R. J. Duro, Progressive Construction of Compound Behavior Controllers for Autonomous Robots Using Temporal Information, in: *Lecture Notes in Artificial Intelligence*, Vol. 1674, Springer-Verlag, Berlín, 1999, pp. 324-328.

[2] R. D. Beer and J. C. Gallagher, Evolving Dynamical Neural Networks for Adaptive Behavior, *Adaptive Behavior* 1(1):91-122 (1992).

[3] D. J. Chalmers, Subsymbolic Computation and the Chinese Room, in: J. Dinsmore (Ed.), *Symbolic and Connectionist Paradigms: Closing the Gap*, 1992.

[4] D. Cliff, P. Husbands, and I. Harvey, Evolving Visually Guided Robots, in: J-A. Meyer, H. Roitblat, and S. Wilson (Eds.), *From Animals to Animats 2*, MIT Press Bradford Books, Cambridge, MA, 1993, pp. 374-383.

[5] D. Floreano and F. Mondada, Evolutionary Neurocontrollers for Autonomous Mobile Robots, *Neural Networks* 11:1461-1478 (1998).

[6] J. C. Gallagher, R. D. Beer, K. S. Espenschied, and R. D. Quinn, *Application of Evolved Locomotion Controllers to a Hexapod Robot*, Technical Report CES-94-7, Dept. of Computer Engineering and Science, Case Western Reserve University, 1994.

[7] I. Harvey, P. Husbands, and D. Cliff, Issues in Evolutionary Robotics, in: J-A. Meyer, H. Roitblat, and S. Wilson (Eds.), *From Animals to Animats 2*, MIT Press, Cambridge, MA, 1993, pp. 364-373.

[8] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michingan Press, Ann Arbor, 1975.

[9] P. Husbands, I. Harvey, D. Cliff, A. Thompson, and N. Jakobi, The Artificial Evolution of Control Systems, in: I. C. Parmee (Ed.), *Proceedings of the 2nd International Conference on Adaptive Computing in Engineering, Design and Control 96*, University of Plymouth, 1996.

[10] N. Jakobi, P. Husbands, I., and Harvey, Noise and the Reality Gap: the Use of Simulation in Evolutionary Robotics, in: *Lecture Notes in Artificial Intelligence*, Vol. 929, Springer-Verlag, 1995, pp. 704-720.

[11] H. H. Lund, O. Miglino, L. Pagliarini, A. Billard, and A. Ijspeert, Evolutionary Robotics – A Children's Game, in: *Proceedings of IEEE 5th International Conference on Evolutionary Computation*, 1998.

[12] L. Meeden, An Incremental Approach to Developing Intelligent Neural Network Controllers for Robots, *IEEE Transactions on Systems, Man and Cybernetics Part. B: Cybernetics* 26(3):474-485 (1996).

[13] O. Miglino, H. H. Lund, and S. Nolfi, Evolving Mobile Robots in Simulated and Real Environments, *Artificial Life* 2(4):417-434 (1995).

[14] O. Miglino, K. Nafasi, and C. Taylor, Selection for Wandering Behavior in a Small Robot, *Artificial Life* 2:101-116 (1995).

[15] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada, How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics, in: R. Brooks and P. Maes (Eds.), Proceedings of Fourth International Conference on Artificial Life, Cambridge, MA, MIT Press, 1994.

[16] S. Nolfi, Using Emergent Modularity to Develop Control Systems for Mobile Robots, *Adaptive Behavior*, 5(3/4):343-363 (1997).

[17] I. Rechenberg, *Cybernetic Solution Path of an Experimental Problem*, Library Translation 1122, Royal Aircraft Establishment, UK, 1965.

[18] R. Salomon, The Evolution of Different Neuronal Control Structures for Autonomous Agents, *Robotics and Autonomous Systems* 22:99-213 (1997).

[19] H. P. Schwefel, *Evolutionsstrategie und Numerische Optimierung*, Ph. D. Thesis, Technische Universität Berlín, 1975.

Figure Captions

Figure 1: a) Average of best individual fitness, b) best individual fitness of 10 evolutions of the homing behavior. One race of 400 individuals, tournament window of 100 individuals, 32 evaluations for each individual and Pc=0.5. Thick line: diagonal distribution. Thin line: random distribution. Remaining line: grid distribution.

Figure 2: a) Average of best individual fitness b) Best individual fitness of 10 evolutions of the light catching behavior. Diagonal distribution, 32 evaluations for each individual and Pc=0.5. Thick line: 8 races of 100 individuals each and a tournament window of 30 individuals. Thin line: 1 race of 800 individuals and a tournament window of 200 individuals.

Figure 3: Best individual fitness and worst individual fitness of 10 evolutions of the light catching behavior. Diagonal distribution, 32 evaluations for each individual and Pc=0.5, a) 1 race of 800 individuals, tournament window of 200 individuals b) 8 races of 100 individuals, tournament window of 30 individuals.

Figure 4: Average of best individual fitness of 10 evolutions of the light catching and wall following behavior. Thick line: Pc=0.5. Normal line: Pc=1.0. Thin line: Pc=0.0, a) 1 race of 400 individuals, light catching behavior, b) 1 race of 100 individuals, light catching behavior, c) 1 race of 20 individuals, wall following behavior, d) 1 race of 20 individuals, light catching behavior. In this last case, the ANN was evolved without delays between nodes.

Figure 5: Operation of the three robot controllers evolved according to local fitness functions defined in the text.

Figure 6: Behavior of controllers evolved and tested with different noise conditions.

Figure 7: Evolution of the fitness of the best individuals each generation.

Figure 8: Wall following behavior. Left: without temporal information, center: with temporal information but without temporal/systematic noise, and right: with temporal information and temporal/systematic noise.

Figure 9: Compound wall following behavior on the simulated and real robot.

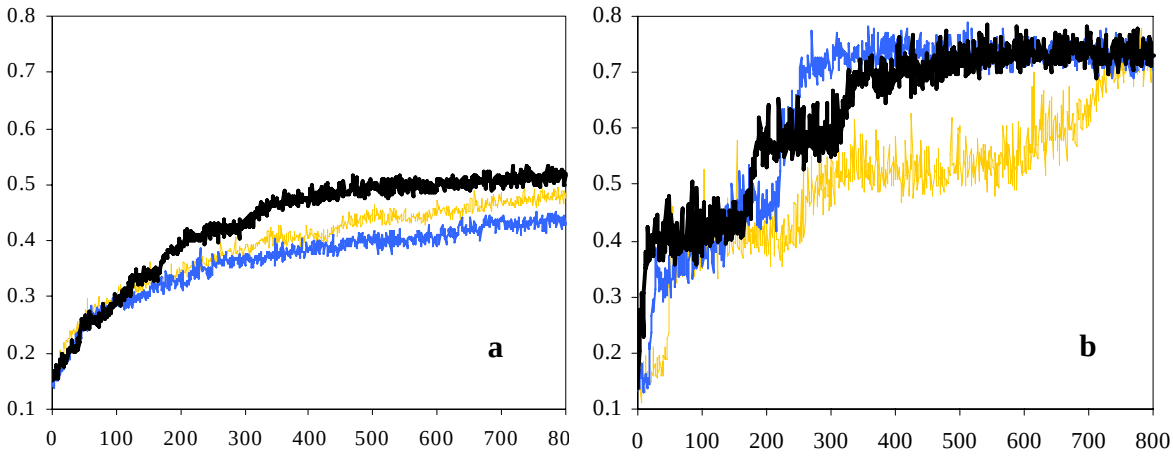
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
100
200
300
400
500
600
700
80
a
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
100
200
300
400
500
600
700
800
b

Figure 1

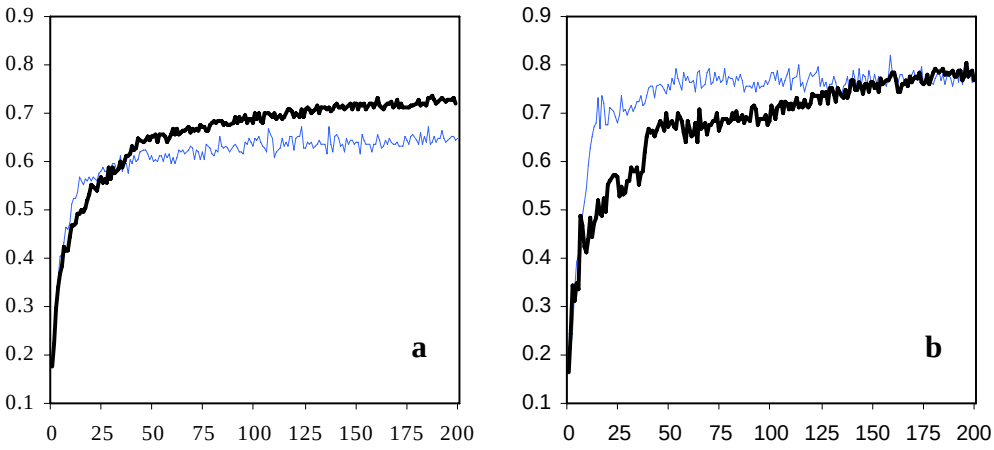

Figure 2

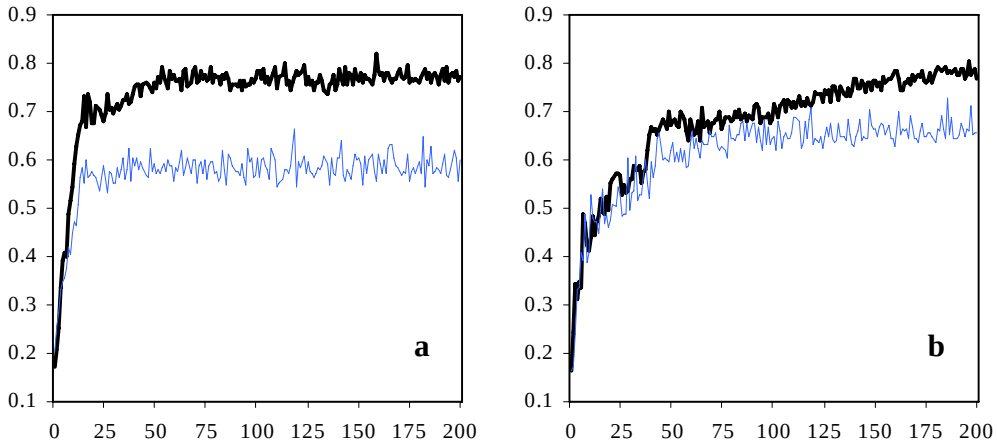

Figure 3

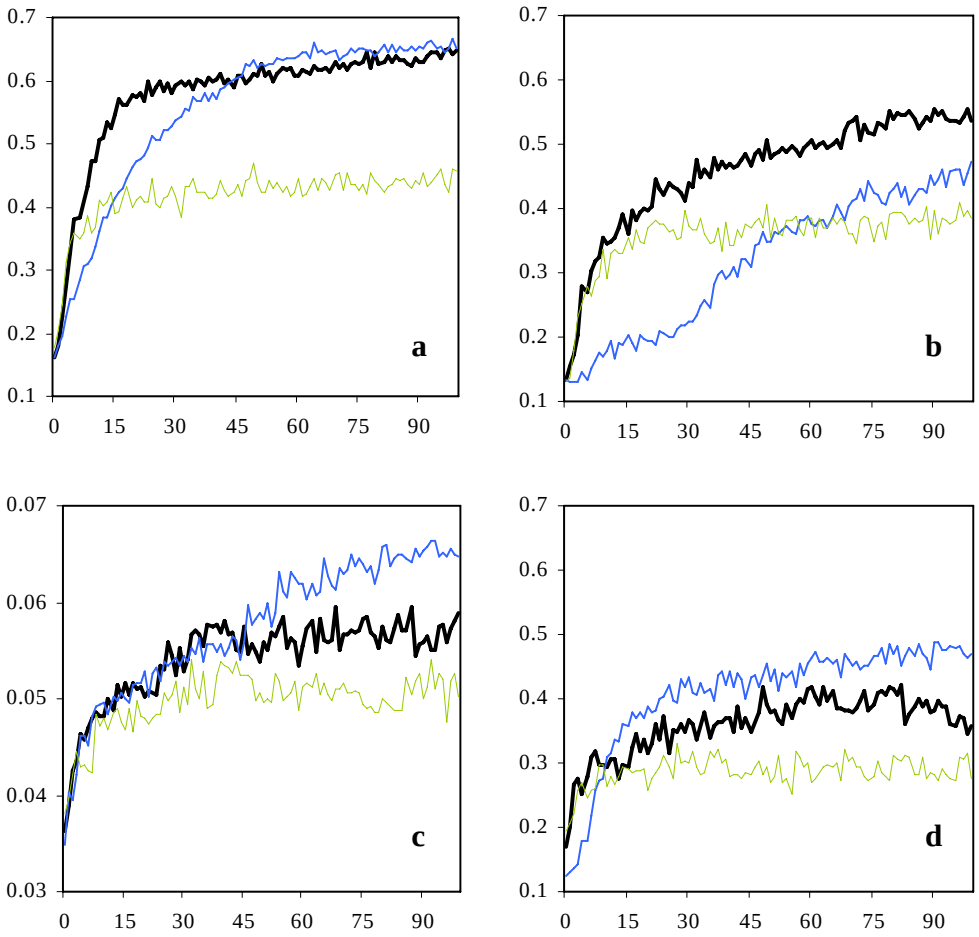

Figure 4

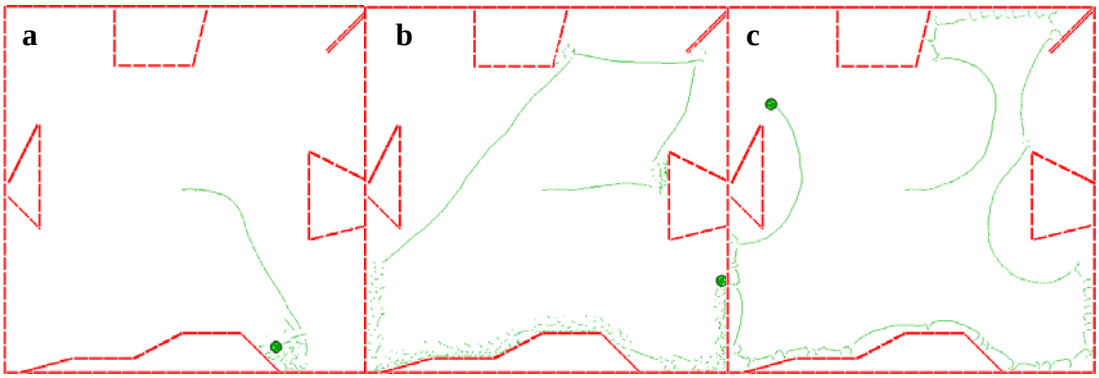

Figure 5

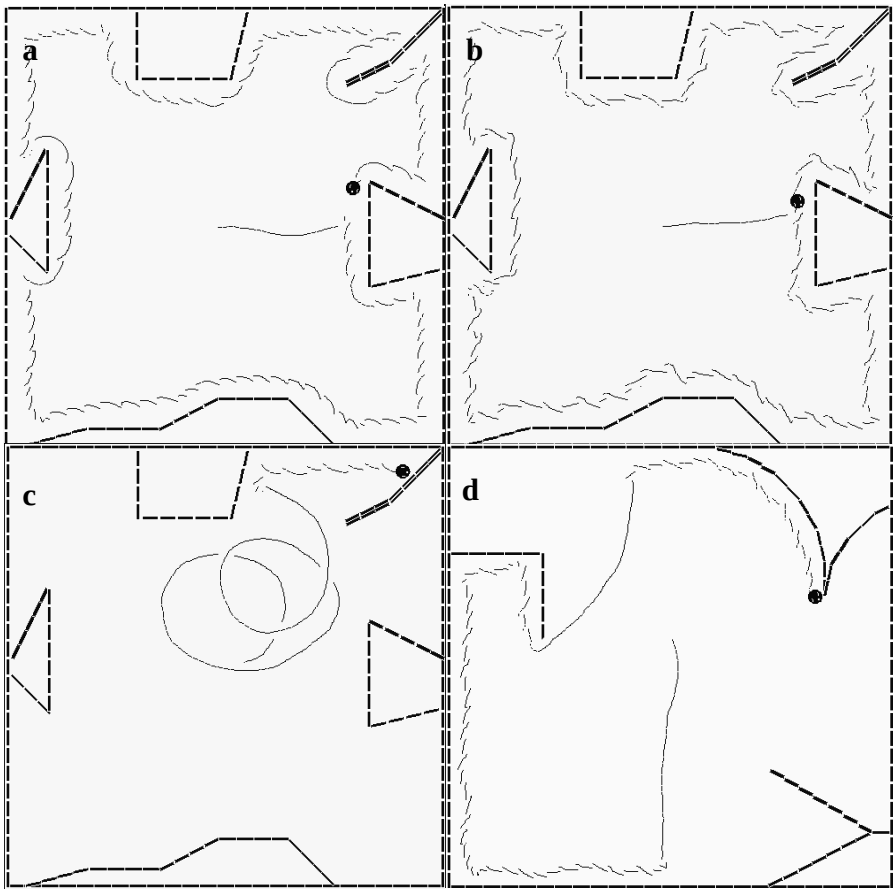

Figure 6

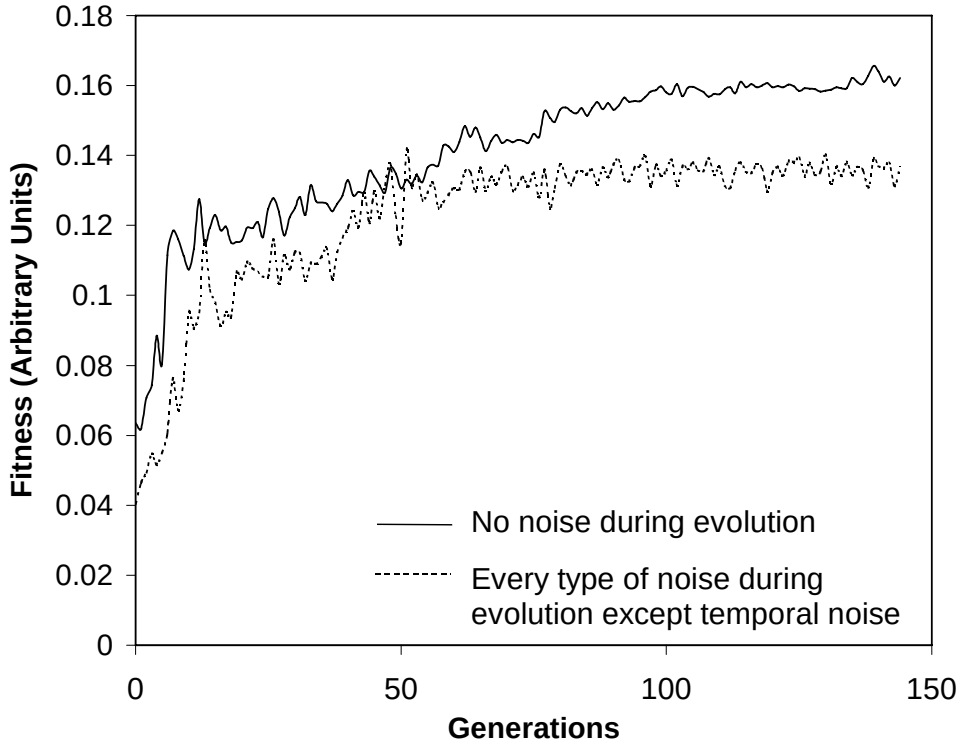
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
Fitness (Arbitrary Units)
0
50
100
150
Generations
No noise during evolution
Every type of noise during evolution except temporal noise

Figure 7

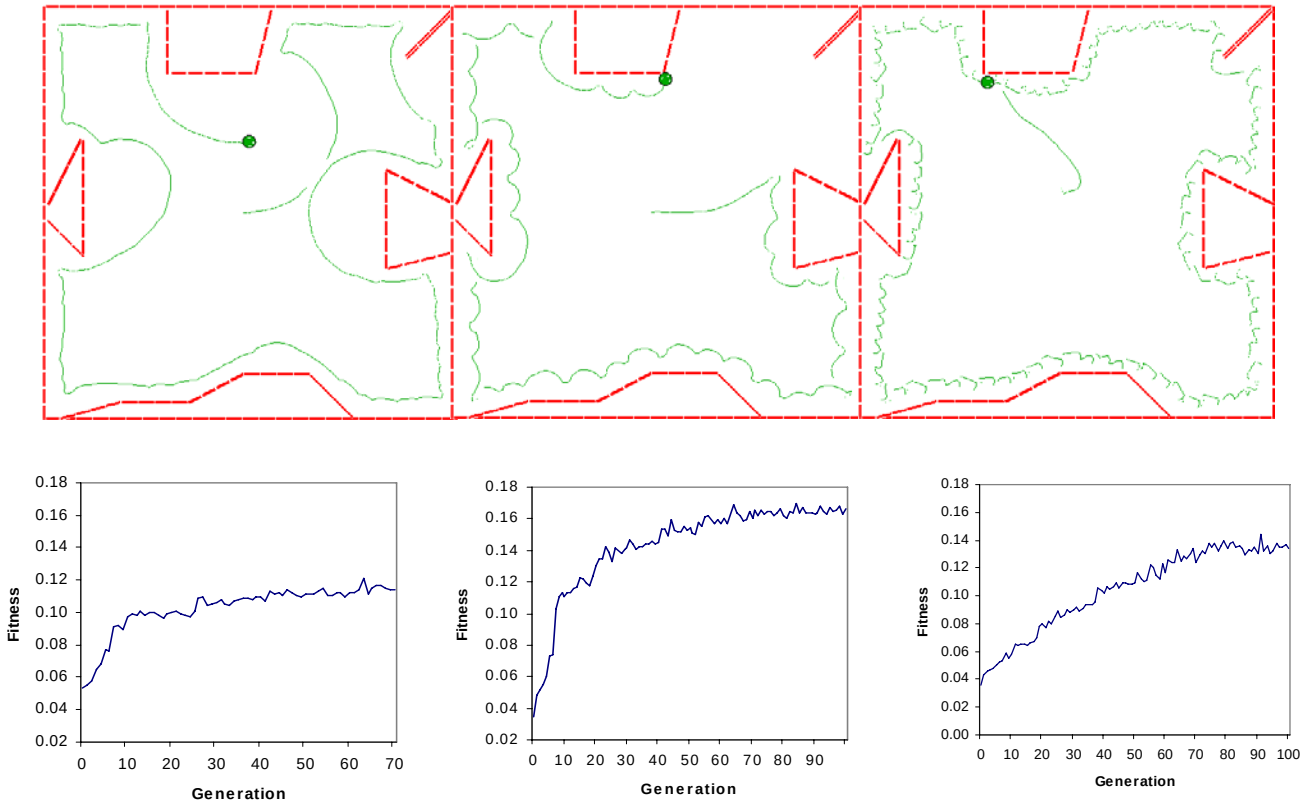
Fitness
Generation
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0 10 20 30 40 50 60 70
0 10 20 30 40 50 60 70 80 90
0 10 20 30 40 50 60 70 80 90 100

Figure 8

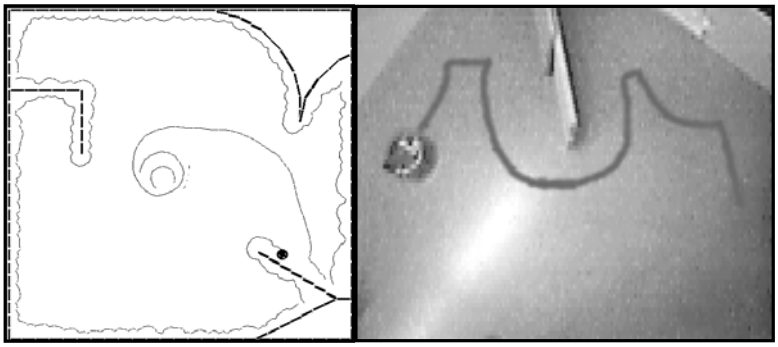

Figure 9