Evolving Neural Controllers for Visual Navigation

Verena V. Hafner and Ralf Salomon
{vhafner,salomon}@ifi.unizh.ch
Artificial Intelligence Laboratory, Department of Information Technology, University of Zurich
Winterthurerstr. 190, 8057 Zurich, Switzerland

Abstract – Biological evidence strongly suggests that insects utilize visual cues for their navigation tasks. This paper discusses the evolution of a simple controller for visual homing by means of evolutionary algorithms. The application is representative for a class of (real world) problems, for which the choice of the fitness function is non-trivial, since the data are not known in advance. For this class of problems, recombination has a much higher influence on the convergence than previously assumed. We show how convergence rates comparable to those of neural network learning algorithms can be achieved.

I INTRODUCTION

Compared to robots, animals exhibit quite impressive navigation capabilities, especially in uncertain and dynamically changing environments. Most animals utilize visual cues for their navigation tasks, for which the pertinent literature proposes several biologically-motivated models [12].

Among these models, both the Average Landmark Vector model[1] and the snapshot model [3] as well as variations thereof have received particular attention, especially for insects with their obviously limited brain capacities. Insects tend to be prey, and it was thus evolutionary beneficial for them to have a close-to-360° panoramic view of their local neighborhood. Some insects take ("store") a snapshot at their home position, and "carry" this snapshot along their way wherever they move to. By utilizing an external compass direction and merely comparing the current view with the stored home-position's snapshot, the insect is able to derive the direction to its home location, also called homing vector [3]. The desert ant *Cataglyphis,* for example, keeps track of the global compass direction by using the polarization pattern of the blue sky [7]. The models mentioned above require that when comparing the two views, they both need to be aligned to the same direction. It is not necessary that this direction is alway the same; instead, it may vary between different comparisons.

From a computational point of view, the snapshot model has several advantages. For example, the insect does not need to store any (global) map of its environment nor does it know anything about coordinates or global positions; the insect is able to find its target by simply following a constantly updated homing vector. An answer to the question "Where am I?" is therefore not necessarily a prerequisite for the decision "Which direction do I have to take to get home?" (for further details, see the discussion in [4]).

[1]Since the Average Landmark Vector model is not in the focus of this paper, the interested reader is referred to [8].

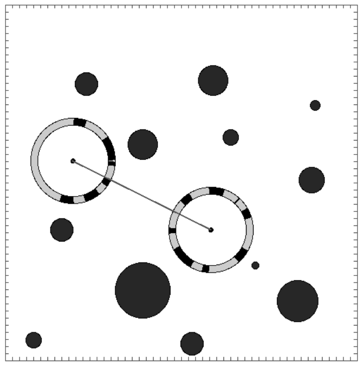

Figure 1: This figure shows a virtual world with a bunch of landmarks, varying in size and position. All landmarks are actually 3-dimensional cylinders and illustrated as black circles. The figure also indicates a robot at two positions with a connecting line, called homing vector. For the sake of comprehensibility, the robot is represented by its significantly blown-up 360° view. For further details, please refer to the text.

Figure 1 shows an appropriate simulation framework for robot experiments. In this environment, the landmarks are randomly-distributed cylinders, which vary in both size and position and are illustrated as black circles. This figure also shows one robot at two different locations with a straight line indicating its homing vector; actually, it does not matter, which of the two is the home position. Figure 1 represents the robot by its 360° view, which is drastically blown up for comprehensibility reasons. In these views, the black segments at the circle's periphery represent the projections of the landmarks onto the robot's center. For illustration purposes, Fig. 2 shows a physical realization of a panoramic camera.

In biologically-inspired approaches, neural networks are often used to determine the homing vector by merely comparing the current view with the snapshot taken at the robot's home position. This approach, however, requires a series of some substantial processing steps. Section II covers these issues including an appropriate learning algorithm, called the delta rule.

The experiments using neural network learning techniques were successful. In addition to these experiments, this paper investigates to which extent evolutionary methods are able to solve this task efficiently. As Section III illustrates, first experi-

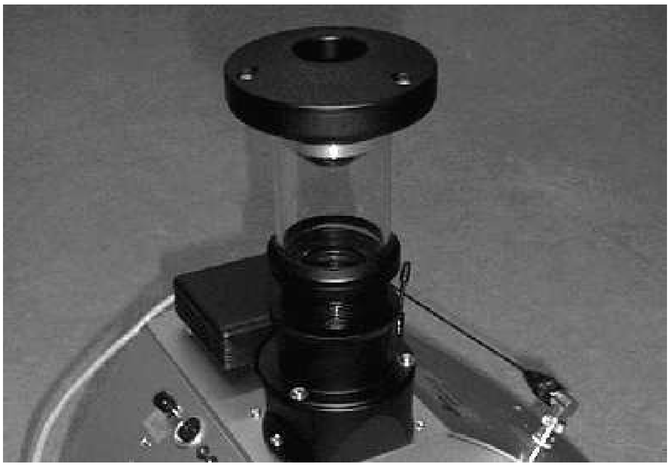

Figure 2: A robot with a panoramic camera mounted on the top. The actual camera is mounted at the bottom and "looks" upwards to the tip of a parabolic mirror, which in turn provides the panoramic scene. In this panoramic image, the horizon is somewhere between the circle's center and its periphery. Some real-world, panoramic images can be found in [5].

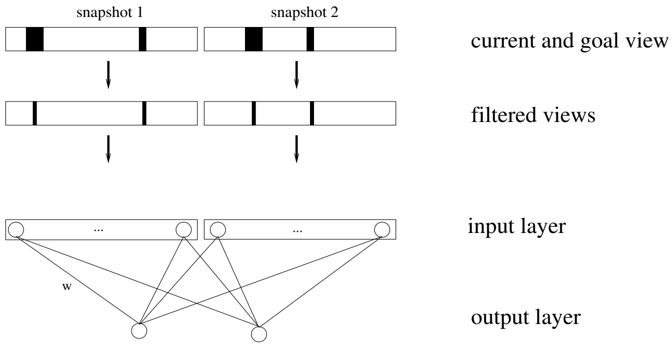

Figure 3: The controller receives two views, one from the current and one from the home position. Each landmark appears as an area of black pixels. The first processing step reduces each landmark to only one pixel located at the landmark's center. In the subsequent step, a neural network feeds the condensed views to an output layer without any hidden layer involved. The output layer consists of only two neurons, and essentially does a view-comparison of some sort, which results in the x and y-values of the homing angle. Further details can be found in the main text.

ments with some standard evolutionary algorithms were rather discouraging. The algorithms either got stuck at poor solutions or required orders of magnitude longer than the delta rule. This might be due to the fact that the previously chosen learning setup [5] is quite unusual for evolutionary methods. Section III discusses the differences and implications in more detail.

This paper investigates various options of how to not only significantly speed up the evolutionary methods but also substantially improve their accuracy on the given problem. To this end, Section IV summarizes the algorithms used and Section V then discusses some of the results. It turns out that some well-chosen procedures (including their parametrization) achieve results comparable to neural network learning techniques. Finally, Section VI concludes with a brief discussion.

II BACKGROUND

This section presents a brief summary of previous research [5], which includes some essential design questions, preprocessing required for the image handling, and results obtained with a simple learning algorithm, called the delta rule [11].

A Neural Network Architecture

Figure 3 presents the main components of the robot's controller. The top layer consists of the two views, i.e., the current view and the home position snapshot. The first preprocessing stage reduces every landmark (i.e., a series of continuous black pixels) to *only one* pixel at its center position. In other words, this preprocessing step encodes for each landmark's angular position and neglects the landmark's width. Note that the distance to the landmark is not available. In theory, this model assumes that all landmarks have an equal distance to the robot. Experiments indicate though [5] that this model also works quite well, if this assumption is not fulfilled.

In the next preprocessing step, the condensed (filtered) views are fed into the input layer of a neural network. The input layer consists of two regions, each responsible for one view. The activation ("0" and "1" encode for white and black, respectively) of the input layer is fed into the output layer without any hidden layer involved. The output layer consists of only two neurons, o_1 and o_2. Each output neuron o_i calculates the weighted input sum as follows:

$$o_i = f\left(\sum_{k=1}^{n} i_k w_{ik}\right) , \qquad (1)$$

$$f(x) = \tanh(x) , \qquad (2)$$

with i_k denoting input neuron k, o_i denoting output neuron i, n denoting the total number of input units, w_{ik} denoting the weight connecting input neuron i_k with output neuron o_i, and $f(x)$ denoting the non-linear transfer function.

The two output neurons o_1 and o_2 encode for the homing direction relative to the robot's orientation. The values are given as $o_1 = \cos(\varphi)$ and $o_2 = \sin(\varphi)$, with φ denoting the homing angle. An additional motor controller – whether in simulation or real-world – is responsible to launch appropriate motor commands. For a deeper discussion on other neural architectures, the interested reader is referred to [5].

B The Learning Rule

For one-layer perceptrons like the one presented in Fig. 3, the literature offers a simple learning procedure, called delta rule.

$$w_{ik} = w_{ik} + \eta(t_i - o_i)i_k , \qquad (3)$$

with w_{ik} denoting the weight connecting input unit i_k with output unit o_i, t_i denoting the expected target value at output unit o_i, and η denoting the learning rate.

In order to be successful, the delta rule requires a sufficiently small learning rate η. Similar to the step size in evolutionary methods, the learning rate significantly influences the runtime behavior of the learning procedure. I.e., too small a value results in a very long training time whereas too large a value may cause convergence problems. In order to achieve both fast conversion and stable results, the following procedure was chosen in [5]: after every learning cycle, the learning rate $\eta_{t+1} \leftarrow \eta_t/\beta$ is divided by the factor $\beta > 1$.

C Training Data

The previously-reported simulation experiments [5] took place in the artificial environment already shown in Fig. 1. As the boarders of the figure indicate, the x and y-coordinates were divided into 50 by 50 tics. After removing those points that lay within a landmark, 2269 points were left for training. In every cycle, the learning procedure constructed a training pattern by the following two-step procedure:

1. Choose two positions $(x/y)^c \neq (x/y)^t$, called current and target, from the remaining 2269 points.

2. Since the robot should operate in *all* orientations equally well, the procedure randomly chooses an angle α, and rotates the two views at $(x/y)^c$ and $(x/y)^t$ accordingly. Please remember that for the model to work successfully, the orientation of both views needs to be aligned. The rotation angle α is chosen anew for each training pattern.

An angular resolution of 4° results in 90 pixels per view. Consequently, the input layer of the neural network features $2{\times}90 = 180$ neurons with a total of $2{\times}180{+}2{=}362$ connections including 2 bias weights.

D Simulation Results

Figure 4 shows some representative results taken from [5]. The delta rule was working in online mode, i.e., the learning procedure applied one weight update (eq. (3)) immediately after choosing *one* training pattern (see subsection II.C). In the figure, graphs (a) and (b) show averages over 10 runs with the settings $\eta_0 = 0.1, \beta = 1.01$ and $\eta_0 = 0.5, \beta = 1.001$, respectively. It can be seen that the choice of η and β strongly influences both the speed and quality of the network mapping.

For illustration purposes, graph (c) shows just a single run of the averaged graph (b) but shifted up 40 error points. These graphs clearly show that the delta rule converges at a final error of about 20. These graphs, especially graph (c) also indicate that the network output is left with a quite high noise level, even at its very end.

Please note that a reasonable choice of η and β requires significant experimentation time. Further architectures, learning procedures, coding schemes, etc. have been investigated in

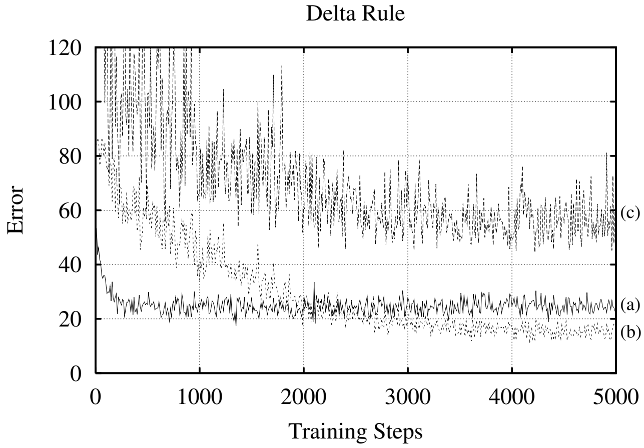

Figure 4: Some delta-rule results taken from [5]. Legend: (a): $\eta_0 = 0.1, \beta = 1.01$, averaged over 10 runs, (b): $\eta_0 = 0.5, \beta = 1.001$, averaged over 10 runs, and (c): $\eta_0 = 0.5, \beta = 1.001$, one single run shifted by error=40.

[5]. In addition, theoretical analyses have shown that the optimal values of the weights w_{ik} are $w_{1k} = c \sin 2\pi k/n$ and $w_{2k} = c \cos 2\pi k/n$ [5].

III PROBLEM DESCRIPTION

In order to be comparable, this paper adopts the training method used in previous research [5]. Since neural network training and standard evolutionary methods differ in several aspects, the "naive" application of the latter more or less fails; in comparison to the delta rule, standard evolutionary algorithms either stop at very poor error values or take way too much time. Some graphs can be found in Section V, particularly in Fig. 5. This section discusses the aforementioned differences as well as the resulting consequences and challenges in more detail.

First of all, both neural network training and artificial evolution would normally operate on relatively fixed training and test sets[2]. For the given environment (Fig. 1), a complete training set consists of $2269 \times 2268 \times 90 = 463{,}148{,}280$ different patterns. One major problem is to select a reasonably small training set that contains all relevant patterns and that has a redundancy as small as possible.

It is more or less impossible for the designer to construct such a set of training patterns. In the previously-chosen setup, this problem was successfully tackled by using online learning. In this mode, the delta rule simply picks a training pattern at random and directly applies a weight update (eq. (3)). From the equation, it is evident that the training procedure updates only those connections w_{ik} that show significant grey values (i.e., the change is proportional to the input activity, and thus vanishes for white pixels) at their inputs i_k. Please note that this paper refers to simulation experiments, which use only black

[2]Some neural network training procedures exchange patterns between those two sets or sparsely apply other random variations [11].

and white pixels. Over time, the procedure presents most or all patterns and thus, gradually evolves all connections w_{ik}.

The usage of non-stationary training sets, however, imposes at least two difficulties on the behavior of evolutionary algorithms. First, fitness-based selection plays a key role in all evolutionary algorithms. In the problem at hand, the individual's fitness value represents its network mapping quality. Since the training sets vary among different individuals, the fitness comparisons are actually be done on different problem instances; loosely speaking, this is a bit comparing apples and oranges. Second, by way contrast to the delta rule, standard implementations of evolutionary algorithms would modify connection weights w_{ik} *regardless* of their inputs i_k. In a particular fitness evaluation, for example, certain weight changes might be (almost) fitness-neutral, since in the randomly chosen views, the corresponding input activity is zero (or very low in real-world experiments). In the next generation, however, a simple copy of this individual might experience a drastic fitness change – whether positive or negative – since it might sense many black pixels at the particular position.

The second major problem is to find an adequate value for the number of patterns an evolutionary algorithm may present to each individual, often called candidate solution. If this value is too small, the determined fitness value may not be representative for the evaluated network. Increasing this value may alleviate this problem. It would also decrease statistical derivations. However, a value too large would consume too much time for each fitness evaluation and would therefore unnecessarily increase the algorithm's runtime. As Section V shows, this number can be set to relatively small values given an algorithm's suitable parametrization.

IV METHODS

The term evolutionary algorithms refers to a class of heuristic population-based search procedures that incorporate random variation and selection, and provide a framework that mainly consists of genetic algorithms, evolutionary programming, and evolution strategies. Despite their historical roots and differences, all evolutionary algorithms maintain a population of μ individuals, called parents. In each generation g, an evolutionary algorithm generates λ offspring by copying randomly selected parents and applying variation operators, such as mutation and recombination. It then assigns a fitness value (defined by a fitness or objective function) to each offspring. Depending on their fitness, each offspring is given a specific survival probability. For a broad overview, the reader is referred to [2].

A Evolution Strategies

Since all weights w_{ik} are real-valued parameters, it is straight forward to resort to evolution strategies[3] and the breeder genetic algorithm [10]. In their simplest form, evolution strategies maintain one global step size σ for each individual, and they typically apply mutations to all n parameters, i.e., $p_m = 1$, as follows

$$w_{ik} \leftarrow w_{ik} + \sigma N(0,1) \ , \tag{4}$$

with $N(0,1)$ denoting normally-distributed random numbers with expectation value 0 and standard deviation 1. Each offspring inherits the step size from its parent (with $\sigma_0 = 0.01$), and prior to mutation, the inherited step size is modified by lognormally-distributed random numbers[4] $\exp(N(0,1))$. This simple evolution strategy is denoted as (μ,λ)-ES or (μ+λ)-ES for short; the first notation indicates that the new parents are selected only from the offspring (i.e., no elitism), whereas the latter also considers *all* parents from the previous generation (i.e., μ-fold elitism). In addition, some evolution strategies also feature various recombination operators such as discrete and intermediate recombination [2]. This paper applies only uniform recombination, which exchanges corresponding weights w_{ik} between two offspring with probability p_r=0.5.

B The Breeder Genetic Algorithm

The breeder genetic algorithm is a genetic algorithm variant particularly tailored to real-valued parameter optimization. It encodes each parameter w_{ik} as a floating-point number, features various crossover operators, and mutates each parameter with probability $p_m = 1/n$ by adding or subtracting small random numbers. In the present investigation, the breeder genetic algorithm uses the same crossover operator as the evolution strategies. The mutation operator is as follows [10]:

$$w_{ik} \leftarrow w_{ik} \pm A2^{-ku} \ , \quad u \in [0,1) \ , \tag{5}$$

with "+" and "-" being selected with equal probability, A denoting the mutation range, k denoting a precision constant, and u being a uniformly-distributed random number. $A = 0.01$ and $k = 16$ were used in all experiments. This algorithm is denoted as (μ,λ)-BGA for short.

C The Fitness Function

According to the previous attempts [5], the fitness function is:

$$f = \frac{1}{2}\frac{100}{P}\sum_{p=1}^{P}\sum_{i=1}^{2}(t_i - o_i)^2 \ , \tag{6}$$

with P denoting the number of patterns used for *one* fitness evaluation. This fitness definition contains two scaling factors. The term 1/2 is adopted from standard neural network learning procedures, such as backpropagation [11], whereas the other term 100/P provides a normalization expressed in percentage. The values t_1 and t_2 are derived from the sine and cosine components of the homing direction, which is determined by the positions of the two randomly-chosen views.

Eq. (6) suggest that the runtime required for simulating one generation is proportional to λ and P. This should be taken into account when comparing performance figures.

[3]Evolutionary programming might be used as well, but the differences to evolution strategies seem negelgible at the application at hand.

[4]Constant factors, such as 1.5, 1.0, and 1/1.5, might work as well.

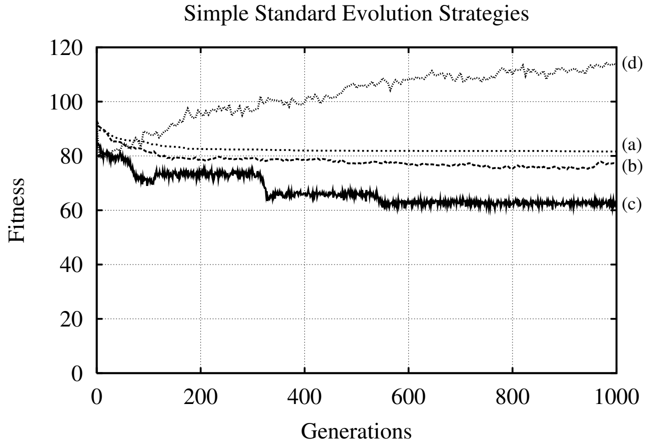

Figure 5: This figure shows the behavior of four different standard evolution strategies with different parametrization. Legend: (a): (1+20)-ES with 2000 patterns, (b): (1,20)-ES with 2000 patterns, (c): a single run of a (1,20)-ES with 2000 patterns, and (d): (1,20)-ES with 200 patterns.

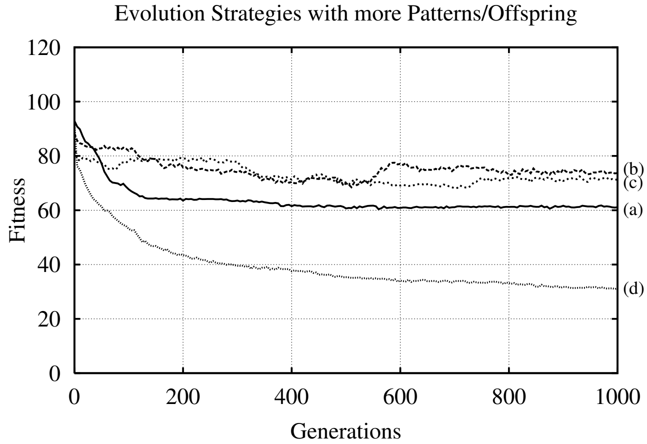

Figure 6: Legend: (a): (1,20)-ES with 10000 patterns, (b): (1,40)-ES with 2000 patterns, (c): (1,80)-ES with 2000 patterns, and (d): (60,200)-ES with 2000 patterns. It can be seen that in comparison to Fig. 5, the improvements are rather small, especially in light of the significantly increased runtime.

V RESULTS

This section presents some results when using evolutionary algorithms to evolve neural networks for visually-guided homing. Unless otherwise stated, all figures represent averages over 10 independent runs.

Fig. 5 presents some results obtained with simple standard methods when applied naively. Despite of showing only four different experiments, Fig. 5 shows some general behaviors. First of all, graphs (a) and (b) clearly indicate that a (μ+λ)-ES performs slightly worse than the corresponding (μ,λ)-ES. At first glance, this might be a bit surprising, since a "+"-strategy does not accept any deterioration. But in the context of noisy fitness evaluations, given here by non-stationary training sets, such "+"-strategies would hold on parental individuals that received fitness values too good just by accident. More generally, the utility of "+"-strategies in noisy environments is rather small, and therefore not further considered in this paper.

Furthermore, graph (c) in Fig. 5 indicates that a single run with small populations might exhibit a significant standard deviation. Last but not least, graph (d) of the same figure shows what happens, if the number of patterns presented to each population member becomes too small. A comparison of graphs (a) and (d) suggests that the minimal number of patterns per individual should be in the order of 2000 for a simple (μ,λ)-ES. That means that in each generation, $20 \times 2000 = 40,000$ patterns are processed. In other words, a (μ,λ)-ES must be considered a failure in comparison to the delta rule, since the final fitness values are four times worse and the number of patterns required is about four orders of magnitude larger.

Figure 6 indicates the effect of increasing the number of presented patterns (graph (a)) and increasing the number of offspring (graphs (b) to (d)). It can be clearly seen that moderately increasing the number of offspring (graphs (b) and (c)) does not result in any worthy improvement. Only a drastic increase of the population size (graph (d)) results in a final mapping quality that is partially in the region of the delta rule. The resulting runtime, however, is unacceptable in comparison to the delta rule. The figure also shows that reducing the noise level (graph (a)) by increasing the number of patterns presented to all population members has some noticeable effect. But again, this action leads to a significantly longer runtime.

The next series investigates to what extent the introduction of recombination can yield a reasonable speedup. Figure 7 clearly shows that the usage of recombination results in both a remarkable speedup and a final accuracy comparable to the delta rule. The obtainable speedup is due to two effects. First, the number of generations required is significantly smaller. Second, recombination allows for drastically reducing the number of patterns to be presented to each offspring. In the experiments shown in the figure, only 100 patterns were used. In contrast to the experiments reported in Fig. 5, this reduction in the number of patterns does not lead to a deterioration but a significant speedup. Furthermore, it may be of interest, that an increase in the number of parents (while keeping the number of offspring at λ=100.) leads to an improved mapping quality; however, above μ=35, the mapping quality gets worse again (not shown). Figure 7 also shows that increasing the population size to a (60,200)-ES further improves the final fitness.

Figure 8 shows some results obtained with the breeder genetic algorithm. To a large extent, the performance graphs resemble previously-discussed figures and are thus not further discussed here. A comparison of the (60,200)-ES (Fig. 6, graph (d)) with the (60,200)-BGA (Fig. 8, graph (d)) suggests that without recombination, a small mutation rate ($p_m = 1/n$) does not suffice, but that all parameters should be mutated.

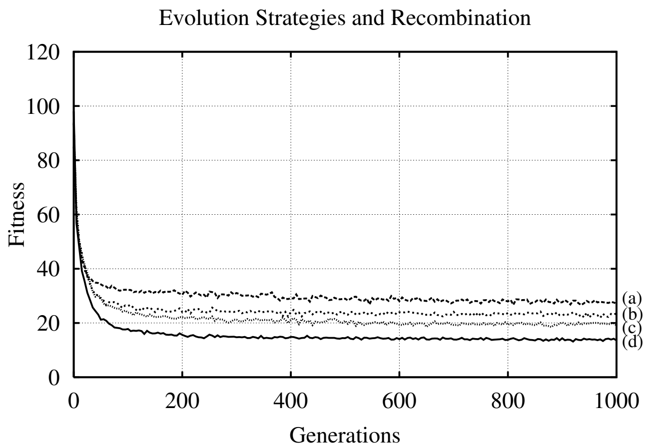

Figure 7: Legend: (a): (15,100)-ES, (b): (25,100)-ES, (c): (35,100)-ES, and (d): (60,200)-ES, all with uniform recombination and 100 patterns presented. This figure shows a remarkable speedup and a final accuracy comparable to the delta rule (neural network learning procedure).

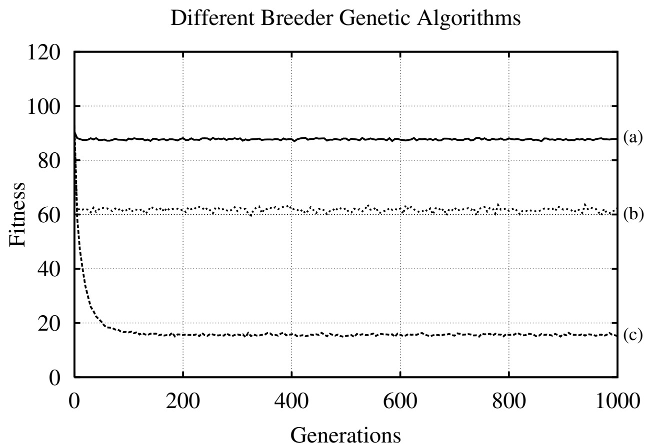

Figure 8: Legend: (a): (1,20)-BGA with 2000 patterns; (b) and (c): (60,200)-BGA with 100 patterns, but with (c) featuring uniform recombination. The performance graphs resemble previous ones.

VI DISCUSSION

This paper has discussed the evolution of neural networks for visually-guided navigation. It has been shown that on this task, the "naive" application of standard evolutionary algorithms may be considered a failure, since their results were considerable worse than those obtained by neural network learning.

One challenge was to appropriately transfer the previously-chosen online learning mode to evolutionary algorithms. Only by using relatively large populations sizes and featuring recombination was it possible to achieve comparable results. This result contrasts a bit common observations. While it is well known [6, 1] that evolution strategies may be accelerated towards a local optimum by recombination, this effect is normally considered moderate; here, recombination on large populations is the key factor. One possible explanation for the strong influence of recombination on the performance could be the following: Depending on how often the corresponding pixel k for a weight w_{ik} has been activated for a certain number of patterns, the weight is closer to its optimal value. By using recombination, new individuals have a chance of getting both higher and lower numbers of weights with many activated corresponding pixels, with the first ones being more likely to be chosen for further generations.

The approach presented here has the disadvantage that the number of patterns presented to each offspring was "hand-coded." Additional experiments have shown that the algorithms can be further accelerated by even fewer patterns. However, at a certain point, the number of patterns is too small to reasonably reflect the individual's fitness; it is easier for the algorithms to "find" a set of good test patterns than to evolve a good network. Future research will be devoted to the development of an adaptation scheme for the number of test patterns.

REFERENCES

[1] D. V. Arnold and H.-G. Beyer, "Performance Analysis of Evolution Strategies with Multi-Recombination in High Dimensional R^N Search Spaces Disturbed by Noise," *Theoretical Computer Science*, 2000

[2] T. Bäck, U. Hammel, and H.-P. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 1:1, pp. 3-17, 1997

[3] B. A. Cartwright and T. S. Collett, "Landmark learning in bees," *Journal of Comparative Physiology*, Vol. 151, pp. 521-543, 1983

[4] M. O. Franz and H. A. Mallot, "Biomimetic robot navigation," *Robotics and Autonomous Systems*, Vol. 30, pp. 133-153, 2000

[5] V. V. Hafner and R. Möller, "Learning of Visual Navigation Strategies," *European Workshop on Learning Robots (EWLR), Prague,* 2001

[6] T. Jansen and I. Wegener, "On the Analysis of Evolutionary Algorithms — A Proof That Crossover Really Can Help," *Proceedings of the 7th Annual European Symposium on Algorithms (ESA '99)*, 184-193, 1999

[7] D. Lambrinos, M. Maris, H. Kobayashi, T. Labhart, R. Pfeifer, R. Wehner, "An Autonomous Agent Navigating with a Polarized Light Compass," *Adaptive Behavior*, Vol. 6, pp. 131-161, 1997

[8] D. Lambrinos, R. Möller, T. Labhart, R. Pfeifer, and R. Wehner, "A mobile robot employing insect strategies for navigation," *Robotics and Autonomous Systems*, Vol. 30, pp. 39-64, 2000

[9] R. Möller, "Insect Visual Homing Strategies in a Robot with Analog Processing," *Biological Cybernetics, special issue: Navigation in Biological and Artificial Systems*, Vol. 83, pp. 231-243, 2000

[10] D. Schlierkamp-Voosen and H. Mühlenbein, "Strategy Adaptation by Competing Subpopulations," *Parallel Problem Solving from Nature (PPSN III)*, 199-208, 1994

[11] R. Rojas, *Neural Networks, A Systematic Introduction*, Springer, 1995

[12] O. Trullier, S. I. Wiener, A. Berthoz, and J.-A. Meyer, "Biologically based artificial Navigation systems: review and prospects," *Progress in Neurobiology*, Vol. 51, pp. 483-544, 1997