
Evolving Neural Controllers for Collective Robotic Inspection

Yizhen Zhang¹, Erik K. Antonsson¹ and Alcherio Martinoli²

¹ Engineering Design Research Laboratory, California Institute of Technology, Pasadena, California 91125, USA

yizhen@caltech.edu erik@design.caltech.edu

² Swarm-Intelligent Systems Research Group, École Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland

alcherio.martinoli@epfl.ch

Summary. In this paper, an automatic synthesis methodology based on evolutionary computation is applied to evolve neural controllers for a homogeneous team of miniature autonomous mobile robots. Both feed-forward and recurrent neural networks can be evolved with fixed or variable network topologies. The efficacy of the evolutionary methodology is demonstrated in the framework of a realistic case study on collective robotic inspection of regular structures, where the robots are only equipped with limited local on-board sensing and actuating capabilities. The neural controller solutions generated during evolutions are evaluated in a sensor-based embodied simulation environment with realistic noise. It is shown that the evolutionary algorithms are able to successfully synthesize a variety of novel neural controllers that could achieve performances comparable to a carefully hand-tuned, rule-based controller in terms of both average performance and robustness to noise.

1 Introduction

Autonomous mobile robots share important characteristics with simple biological systems: robustness, simplicity, small size, flexibility and modularity. Each individual is rather simple with limited local sensing and actuating capabilities, while as a group they can accomplish difficult global tasks in dynamic environments, without any external guidance or centralized control [3].

Design and control of such a robot swarm are difficult mainly because their group behavior is an emergent property of their mutual interaction and that with the environment, which become a distributed dynamical system due to independent parallel actions of different individuals. Since the robots only have partial perceptions based on crude noisy sensors, limited computational capabilities and energy budget, managing the robots to solve a global task under such constraints presents significant technical challenges. This is especially

true because human intelligence is specialized in individuals and centralized control, instead of the collective intelligence shown in nature.

Evolutionary robotics [13] is a new and promising technique for automatic design of such autonomous robots. Inspired by nature, evolutionary robotics makes use of tools such as neural networks and evolutionary algorithms.

Inspired by biological neural networks, Artificial Neural Networks (ANN) have been a powerful computational tool widely applied in science and engineering [8]. They are often used to implement robot controllers because of their light computational requirements and nonlinear basic elements, properties that allow for real-time control and, potentially, modular implementation of complex perception-to-action functions. ANN can be designed and trained using various methods, including those based on evolutionary computation [14, 18]. As opposed to optimization of behavior-based controllers, the key feature of ANN evolution is that, the genotypical searching space is less constrained by ANN models and the resulting phenotypical solution directly shapes the robot behavior as a whole.

Evolutionary algorithms [2, 6, 12], loosely inspired from biological evolutionary processes, have gained considerable popularity as tools for searching vast, complex, deceptive, and multi-modal search spaces with little domain-specific knowledge. In recent years, they have found natural applications in the automatic synthesis of artificial neural networks for intelligent agents [15]. Evolutionary algorithms allow co-evolution of the network architectures as well as the weights within task-specific design constraints. Multiple design objectives can be expressed as fuzzy preferences and aggregated into the fitness function with different weights and trade-off strategies, which can be tuned to evolve solutions with different engineering design trade-offs [1, 16]. As stochastic optimization methods, evolutionary algorithms are also good at working in noisy environments and searching for robust solutions.

Evolution of both feed-forward and recurrent neural controllers are considered in this paper. Only synaptic weights are evolved if the ANN topology is pre-defined; otherwise the network structure and synaptic weights are simultaneously evolved, where the candidate solutions can be modified by adding or deleting hidden neurons, establishing or removing connections between any two neurons, as well as changing values of the synaptic weights.

The efficacy of the evolutionary methodology is demonstrated in the framework of a realistic case study concerned with collective robotic inspection of regular structures. The controllers are optimized in a realistic, sensor-based, embodied simulator, then downloaded to real hardware. If the embodied simulator is faithful enough for the target hardware platform, evolved controllers can be easily transferred to real robots [11]. Homogeneity of the robot team is enforced here to limit the search space, achieve scalability, and bypass the credit assignment problem typically arising in distributed systems consisting of individuals using only local information [7, 17].

The performance of the evolutionary results is compared with that of a hand-coded, rule-based controller carefully tuned for the same task. It will be

shown that the evolutionary algorithm appears powerful and promising for automatic synthesis of novel ANN controllers, requiring little prior domain knowledge or ANN structural information. The evolved solutions can serve as good starting points for further study and optimization by human engineers.

2 Evolutionary Methodology

Based on the well-known principles of genetic algorithms (GA) and evolutionary strategies (ES), our evolutionary optimization loop uses real numbers to encode synaptic weights, variable chromosome length to evolve ANN structures, traditional roulette wheel selection with fitness scaling, and both crossover and mutation genetic operations to improve candidate solutions [1].

2.1 Encoding of Artificial Neural Networks

The ANN synaptic weights are directly encoded as a sequential vector of real numbers, as shown in Fig. 1. The vector (chromosome) length is static if the network structure is *a priori* determined, where a fully connected ANN is usually assumed and the fixed chromosome length can be computed as follows:

$$n_c = \begin{cases} (1 + n_i)n_o & \text{if } n_h = 0 \text{ \& feed-forward} \\ (1 + n_i)n_h + (1 + n_h)n_o & \text{if } n_h > 0 \text{ \& feed-forward} \\ (1 + n_i + n_o)n_o & \text{if } n_h = 0 \text{ \& recurrent} \\ (1 + n_i + n_h)n_h + (1 + n_h + n_o)n_o & \text{if } n_h > 0 \text{ \& recurrent} \end{cases} \quad (1)$$

where n_c , n_i , n_h , n_o and “1” represent the numbers of fully connected weights, inputs, hidden neurons, outputs, and biases, respectively.

When the ANN structure is also evolved, n_h becomes a design variable to be optimized. So the chromosome length must also be variable to accommodate the variable ANN structure and evolve solutions of suitable complexity. To give the algorithm more freedom to search for the appropriate network structures, fewer restrictions are imposed on the number of permissible connections and the variable chromosome length is computed as follows:

$$n_c = \begin{cases} (1 + n_i)n_h + (1 + n_i + n_h)n_o & \text{if feed-forward} \\ (1 + n_i + n_h + n_o)(n_h + n_o) & \text{if recurrent} \end{cases} \quad (2)$$

where n_c represents the maximum possible number of connections, but not all of them must be active. A non-zero real value in the genotype vector represents the weight value of an active connection, while zeros represent inactive (non-existent) connections. Note that the $n_h = 0$ cases of (1) are included in (2).

2.2 Initialization

The population is randomly initialized at the beginning of an evolutionary run. For fixed network structure cases, all the genotype vectors are of the

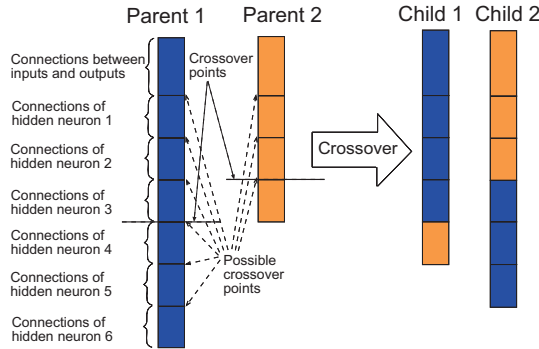


Fig. 1. Illustration of crossover scheme for two chromosomes of different lengths

same length with random values. For variable network structure cases, first n_h is randomly selected from 1 to 10^3 for each individual, then the genotype vector length is computed by (2), and each real number in the genotype vector is set to a random value (active) or zero (inactive) with probability 50%.

The networks initialized this way might contain some useless hidden neurons which do not contribute to the outputs at all. To improve the algorithm efficiency, they are identified and removed from the network by a simple routine after initialization to make the network more concise and relevant.

2.3 Genetic Operations

Crossover and mutation are both used in the evolutionary algorithm here.

In fixed ANN structure cases, standard crossover operators can be directly applied to two real vectors of equal length. In variable ANN structure cases, the crossover must operate on two vectors of different lengths which represent two distinct network structures. It is known that evolutions relying on crossover do not perform well in ANN topology optimization because of their intrinsic disruption feature and the permutation problem [18]. To protect possible modules in the network, crossover points can not be arbitrarily chosen along the whole genotype vector. As shown in Fig. 1, the genotype vector can be grouped into sub-vectors or blocks according to the hidden neurons, and the crossover implemented here only allows block-wise interchange between the parents. The algorithm also tries to best match the sequence of the hidden neuron blocks of before crossover to reduce influence of the permutation problem. Then, from all possible crossover points, as shown in Fig. 1, a random crossover point is selected for both parents and their hidden neuron blocks below the crossover point are exchanged to create two new offspring. For the example shown in Fig. 1, two parents networks of 6 and 3 hidden neurons produce two children networks of 4 and 5 hidden neurons respectively.

³ There is, however, no upper limit for n_h during the evolution.

Mutation is also a powerful tool for creating new network structures and synaptic weights. In fixed structure cases, only Gaussian mutation is used to change values of the synaptic weights. In variable structure cases, two extra types of mutations are introduced. First, a hidden neuron could be added to or removed from the current network configuration. Second, a connection between any two neurons could be turned on or off by switching between non-zero and zero values. When a hidden neuron is added or a connection is switched on, the synaptic weight values are initialized as described in Sect. 2.2.

The crossover and mutation operations could also introduce useless hidden neurons and identical individuals, which are both removed from the network before evaluation. If identical copies are allowed to exist in the population, the power of crossover and pool diversity are reduced, which might cause pre-convergence of the evolution.

3 Case Study: Collective Robotic Inspection

This case study is concerned with the automatic synthesis of ANN-based control algorithms to inspect regular structures using a homogeneous robot swarm. The goal is to design “local” algorithms that coordinate the fine motions of the moving platforms. Sensor uncertainty and vehicle position uncertainty should be taken into account when planning the local motions that carry out the gross motion plan, i.e., the collective effect of the multi-vehicle platform as a whole.

3.1 Application Background

Autonomous robots find a wide variety of applications in the real world to release humans from various chores, especially when the working environment is hazardous or not easily accessible by human beings. For instance, inspection of human occupied space transportation systems and platforms is currently heavily labor intensive, costly and time consuming. Another example could be the inspection of propulsion systems (such as jet turbines), which is usually performed visually by human experts using borescopes, a process which is also time consuming and cost intensive [9]. Therefore it is desirable to have the inspection task performed autonomously by a swarm of miniature mobile robots in these situations. This idea is intellectually appealing and it could find broad applications for general inspection of engineered or natural structures.

3.2 Experiment Setup and Simulation

This paper considers a simple 2-dimensional (2D) scenario, where the objects to be inspected have regular circular shapes (20 cm in diameter), as shown in Fig. 2(a). It is assumed that completely circumnavigating an object is a good

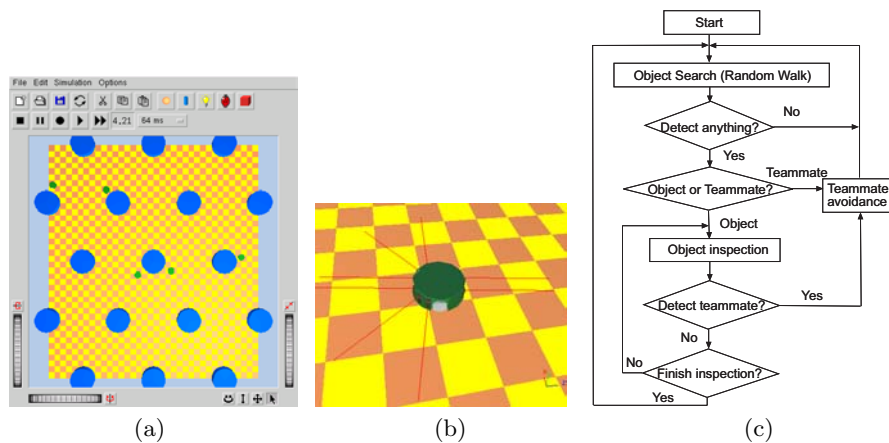


Fig. 2. (a) Top view of structure inspection setup in the embodied simulator: the bigger disks represent the cylindrical objects to be inspected while the smaller dots are the robots; (b) Close-up of the robot model with its distance sensor rays (*solid lines*); (c) The logical diagram of the rule-based hand-coded controller.

emulation of the scanning-for-flaws maneuver. A continuous world scenario without walls is also simulated by wrapping the robots position around to the opposite side when they move out of one side.

The collective performance measure for this scenario depends on the ratio of the inspected object surfaces over a pre-specified time span to all that needed to be inspected in the world. Therefore the maximum performance “1” can be achieved by complete coverage, i.e., fully inspecting all distinct objects in the world, within the time limit. Note that only 12 distinct objects are present in the world shown in Fig. 2(a) under the wrap-around condition.

The simulation scenario shown in Fig. 2 is implemented in Webots⁴ [10], a 3-dimensional (3D), embodied, sensor-based, kinematic simulator. As shown in Fig. 2(b), the simulated robots are miniature unicycle vehicles (5 cm in diameter), like the Khepera⁵ robots. They are equipped with eight distance sensors as ANN inputs with extended sensor range (10 cm). The sensors are assumed to be line sensors characterized by a linear response with the distance: the closer the sensed object the higher the value. Integer sensor values ranging from [0, 1023] with ± 10 white noise are normalized to [0, 1] before feeding to the ANN controller. Both hidden (if any) and output neurons use sigmoid output functions producing outputs in the range of [0, 1]. The ANN has two outputs mapping to the two wheel speeds⁶ of the robots, taking integer values from [-20, 20], with each speed unit representing 8 mm/s in the real world.

⁴ <http://www.cyberbotics.com/products/webots/>

⁵ <http://www.k-team.com/robots/khepera/>

⁶ Either the left and right speeds, or the forward and rotation speeds.

Wheel slippage is also simulated with $\pm 10\%$ white noise on wheel speeds at each simulation step.

The inspection task requires the robot to approach an object to be inspected, inspect it with minimal redundancy and maximum speed, leave it and search for other objects. In collective scenario the robot also needs to avoid teammates. Although one could implement a rule-based hand-coded controller (see Sect. 3.3) or apply behavior-based control algorithms [4], this is nevertheless a non-trivial task for a neural controller reading eight crude distance sensors and controlling two motor wheels directly. Indeed, the controller must not only evolve basic reactive behaviors such as object search, object inspection (i.e., follow the contour of the object) and teammate avoidance but also correctly *sequence* them: for instance, switching from object search to inspection when an object is found and searching for new objects after fully inspecting an object.

3.3 Hand-coded Controller

In order to create a baseline of what level of performance is achievable in this specific case by a traditional design method based on human intelligence, a simple hand-coded controller based on logical rules has been implemented for the same task. It exploits exactly the same sensor inputs and controls the same motor outputs as the evolved ones, and can be used to evaluate and compare with the evolved ANN controller solutions. As shown in Fig. 2(c), the hand-coded controller is based on the robot's distance sensor inputs and some internal timers. There is one key parameter that controls how long the robot keeps inspecting an object. This parameter has been optimized by systematic search in order to get the best performance.

Although it is rather straightforward to implement such a rule-based hand-coded controller for the simple scenario defined in Sect. 3.2, it is not obvious how to complete the same task with a structural ANN controller. Moreover, it might become more intractable and even infeasible to implement such a hand-coded controller for more complex (*e.g.*, inspection of 3D, irregular structures) scenarios, where evolutionary algorithms might be more appealing.

4 Results and Discussions

The evolutionary algorithm was applied to evolve ANN controllers under different configuration settings: feed-forward or recurrent networks with or without a variable number of hidden neurons, as shown in Table 1. The population sizes of the evolutions depend on the dimension of the genotype vector to be optimized. The first two types of neural networks are of fixed simplest topologies with shortest chromosome lengths and smaller pool sizes. The latter two types of neural networks are of variable structures with longer (average) chromosome lengths and larger pool sizes.

Table 1. Different ANN types considered in the evolutionary algorithm

ANN type description	symbol pool size	
feed-forward without any hidden neurons	ffnh	50
recurrent without any hidden neurons	rcnh	50
feed-forward with n_h^* hidden neurons	ffvh	100
recurrent with n_h^* hidden neurons	rcvh	150

*Note $n_h \geq 0$ is a variable number.

For each type of ANN controller synthesis, a series of evolutionary experiments were conducted with different output speed maps and different coefficients in the sigmoid neuron output function. For each evolutionary experiment, 5 evolutionary runs with different random seeds were performed, each lasting for 200 generations.

Due to the noise present in the fitness evaluation of controller solutions in the simulated environment, a noise test of 100 evaluations was applied to each individual in the final population of each evolutionary run as well as the hand-coded controller to get fairer performance comparisons of different controllers (refer to Fig. 6). Different aggregation criteria (minimum, geometric mean, average, etc.) can be used as a measure to estimate the overall performance of each solution from its multiple fitness values. However, only one single evaluation was applied for each individual during the evolutions, because the evaluations in the simulated environment are significantly more computationally expensive than the genetic operations and in this case single evaluation (smallest sample size) is the most computationally effective strategy [5].

In the following sections, the inspection task was approached by three systematic steps: the single robot single object scenario (Sect. 4.1), the single robot multiple objects scenario (Sect. 4.2), and the multiple robots multiple objects scenario (Sect. 4.3).

4.1 Single Robot Single Object (SRSO) Scenario

The global problem of collective robotic inspection of multiple objects in the world can be decomposed to the microscopic interaction between a single robot and a single object, as shown in Fig. 3(a). Here the goal of the robot is to make one and *only one* full circle of the object as soon as possible without any collisions. A short evaluation span of 500 time steps⁷ was chosen here. The robot always started at the same initial position and orientation (facing the object) for all evaluation spans here to reduce noise effects. Walls were included in this scenario to facilitate the development of object avoidance behavior of the ANN controllers during evolutions. Walls can be distinguished from the circular-shaped object by their different sensory patterns.

⁷ Each time step simulates 64 milliseconds in real time.

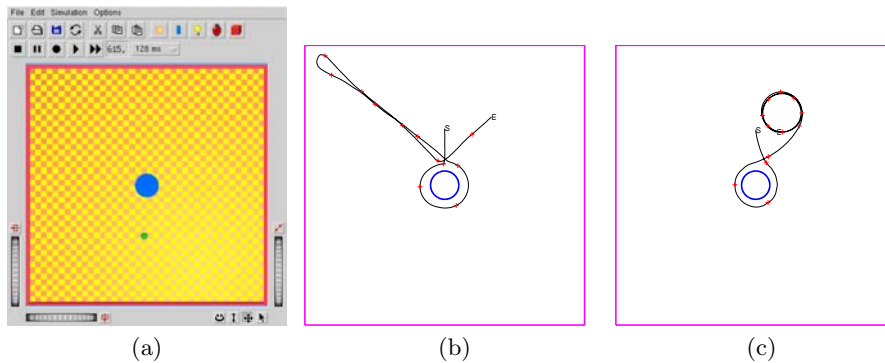


Fig. 3. (a) Screen shot of SRSO scenario; sample robot trajectories for 500 time steps (32 s) using the hand-coded controller (b) and the best evolved controller (c): “S” represents the constant starting point and “E” the ending points, with “+” symbols placed along the trajectories every 40 time steps (2.56 s)

Figure 3 shows the sample robot trajectories of the hand-coded controller and the best ANN controller evolved. It is interesting to note the distinct behaviors of the two controllers shown here. The rule-based hand-coded controller clearly follows the logic shown in Fig. 2(c): it goes directly to the object, walks around it, then leaves and starts random walk. While the evolved one just hangs around in circles after it finishes the inspection task.

It turned out that for this scenario neural controllers that had access to an additional timer input achieved better results than those had not, and were comparable to the hand-coded controller, which also used timers. This implies that timing was a key factor here due to the lack of spatial clues in the world.

4.2 Single Robot Multiple Objects (SRMO) Scenario

A single robot is let to explore the multi-object scenario shown in Fig. 2(a) in the SRMO scenario. Again the goal here is to inspect (circle around) as much as possible the 12 distinct circular-shaped objects in the world. No walls were simulated in this scenario since wrap-around was applied to simulate a continuous world, similar to an unfolded ball. The evaluation span was 2000 time steps here for each ANN candidate controller during evolutions. The robot starts from a random initial position and orientation for each evaluation.

Figure 4 shows sample robot trajectories of the hand-coded and evolved controllers in the SRMO scenario. The difference is quite obvious. The hand-coded controller always tries to make a full circle of each object it finds, then walks away in rather straight lines to search for “new” objects⁸. Different evolved controllers show a variety of different behaviors. The one shown in

⁸ Note that the robot had no clue to figure out whether a newly discovered object was inspected before or not according to Fig. 2(c).

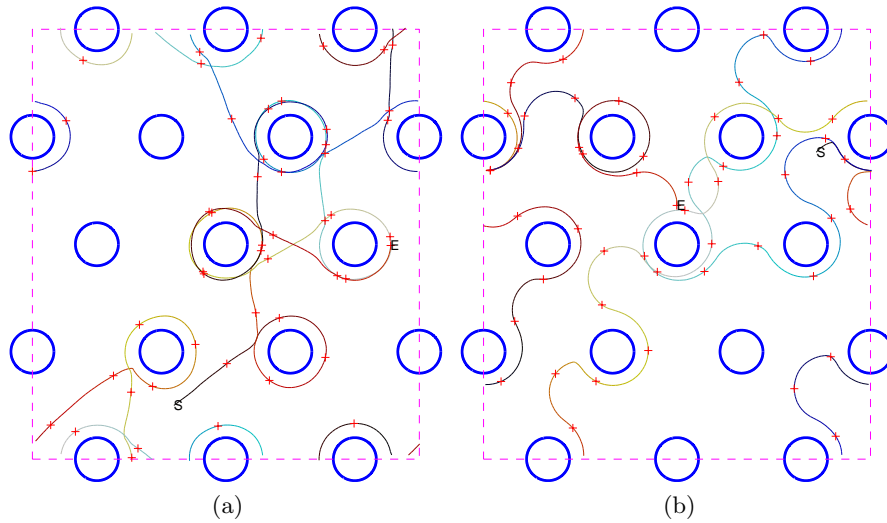


Fig. 4. Sample robot trajectories of the SRMO scenario for 2000 time steps (128 s) using the hand-coded controller **(a)** and the evolved controller **(b)**. The dashed lines delimit the wrap-around boundaries; “S” represents the random initial starting points and “E” the ending points. The trajectories are shown in gradually changing colors with “+” symbols every 40 time steps (2.56 s)

Fig. 4(b) always walks in alternate curves: counterclockwise and fitly curved when inspecting an object while clockwise and less curved otherwise. Most engineered solutions would probably apply the strategy of the hand-coded controller naturally, it is surprising to discover that this evolved ANN-based strategy could work equally well here. In addition, the evolved controllers no longer need additional temporal input here to be comparable with the performance of the hand-coded controller, which still depends on its timers.

However, it might be more difficult for the evolved controllers to achieve complete coverage of all objects. Because it might leave (drift away from) an object before fully inspecting it, it would generally take longer to fully inspect all objects in the world than the hand-coded one. One possible cause might be that the robot could hardly achieve complete coverage within the evaluation span, hence there was no pressure in the evolution to favor complete coverage.

4.3 Multiple Robots Multiple Objects (MRMO) Scenario

Finally a homogeneous team of five robots were employed to collectively inspect the multiple (12) circular objects in the world, as shown in Fig. 2(a). The goal as well as the wrap-around and random initial conditions were the same as those in Sect. 4.2, with an evaluation span of 800 time steps.

Figure 5 shows the sample robot trajectories of the hand-coded and evolved controller for the MRMO scenario. Both behaviors seem to follow the respec-

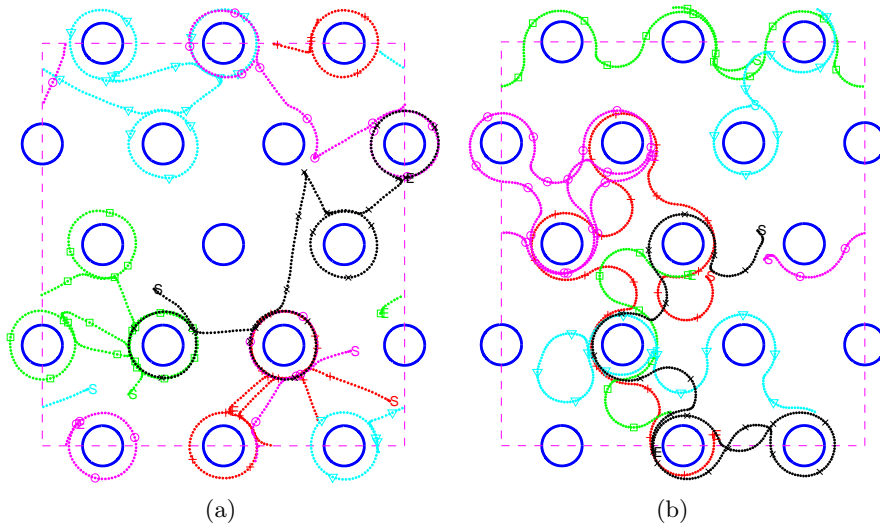


Fig. 5. Sample robot trajectories of the MRMO scenario for 800 time steps (51.2s) using the hand-coded controller (a) and the evolved controller (b). The dashed lines delimit the wrap-around boundaries, but some trajectories beyond the boundaries were kept to enhance the display. “S” represents the random initial starting points and “E” the ending points. Different robots’ trajectories are shown in different colors with different markers placed every 40 time steps (2.56s)

tive strategies discussed in Sect. 4.2. It is noteworthy for the evolution to develop the obstacle (including teammates) avoidance behavior, especially when it was not explicitly defined in the fitness function.

Figure 6 shows the performances of the hand-coded and best evolved controllers under different ANN types (as shown in Table 1) for the MRMO scenario. It is shown that the evolved controllers, especially those with a variable ANN topology, seem to have achieved comparable performances as the hand-coded controller in terms of average performance, and even slightly beat the hand-coded controller in terms of worst performance, appearing more robust to noise. It is remarkable for the evolutionary algorithms to automatically discover robust solutions from only single noisy evaluation of each candidate solution during the evolutions, which verified its extraordinary ability to work in noisy environments.

It is also observed that controllers evolved with variable ANN topologies can generally achieve better results than those with fixed ANN topologies. This demonstrated the power of evolutionary algorithms to synthesize appropriate ANN topologies for a given problem, and evolve the necessary synaptic weights simultaneously. Although the best evolved control strategies have achieve the same level of performance as each other as well as the hand-coded controller, their underlying ANN topologies are completely different, including

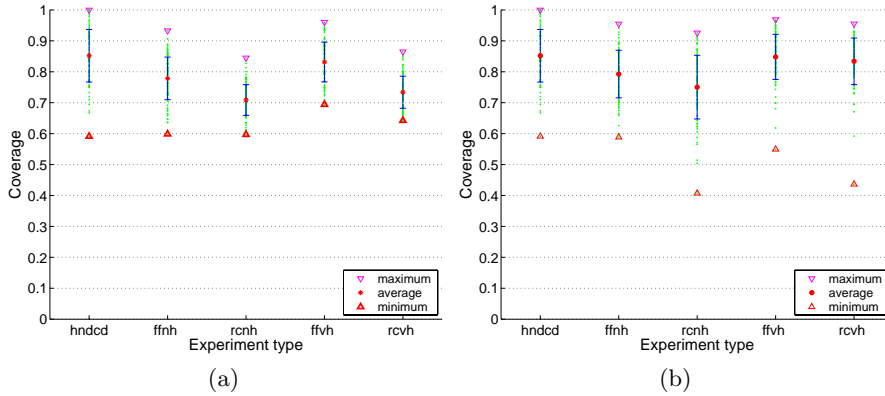


Fig. 6. Coverage values achieved by the hand-coded controller (hndcd) and the best controllers evolved under different ANN architectures (refer to the symbols in Table 1) and selected according to **(a)** minimum and **(b)** average performance for the MRMO scenario shown in Fig. 2(a). Each column shows the coverage values (*the green dots*) obtained by one controller during the 100 evaluations in its noise test and the error bars indicate the standard deviation

both feed-forward and recurrent ANN's with a number of hidden neurons from two to six. They can provide human engineers with diverse alternative candidate solutions that might be difficult to conceive from human intelligence. On the other hand, no significant performance differences were observed between the feed-forward and recurrent ANN's in all scenarios when all other conditions were the same.

It is also noted that the evolutionary algorithm is able to adapt the controller solutions according to the collective or single robot scenarios. Hence the controllers evolved in collective scenarios can achieve better results in collective scenarios than those evolved in single robot scenarios, and vice versa.

5 Conclusion and Future Work

An evolutionary algorithm was applied to automatically synthesize neural controllers for autonomous robots in a noisy simulation environment with little prior knowledge on ANN topologies. The methodology was validated in the framework of a case study concerned with collective robotic inspection of 2D regular structures. It was shown that the best evolved controllers can achieve excellent and robust performances with a variety of different ANN architectures, providing multiple good candidate solutions for human engineers.

In the future, the same synthesis methodology will be applied to more complex and realistic problems such as collective robotic inspection of 3D irregular space structures and/or jet propulsion systems. Implementation and verification of evolved controllers with real robots would also be meaningful.

Acknowledgments

This material is based upon work supported, in part, by NASA Glenn Research Center and by the Engineering Research Centers Program of the National Science Foundation under Award Number EEC-9402726. Martinoli A is currently sponsored by a Swiss NSF professorship.

References

1. Antonsson EK, Zhang Y, Martinoli A (2003) Evolving engineering design trade-offs. In Proc. 15th Int. Conf. on Design Theory and Methodology, ASME. Paper no. DETC2003/DTM-48676.
2. Bäck T (1996) Evolutionary algorithms in theory and practice. Oxford University Press, New York, NY.
3. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: From natural to artificial systems. Oxford University Press, New York, NY.
4. Correll N, Martinoli A (2004) Modeling and optimization of a swarm-intelligent inspection system. In Proc. 7th Symp. on Distributed Autonomous Robotic System (DARS), Springer-Verlag.
5. Fitzpatrick JM, Grefenstette JJ (1988) Genetic algorithms in noisy environments. *Machine Learning*, **3**(2-3):101–120.
6. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, MA.
7. Hayes AT, Martinoli A, Goodman RM (2003) Swarm robotic odor localization: Off-line optimization and validation with real robots. *Robotica*, **21**(4):427–441.
8. Hertz J, Krogh A, Palmer RG (1991) Introduction to the theory of neural computation. Perseus Books, Reading, MA.
9. Martin K, Stewart C (2000) Real time tracking of borescope tip pose. *Image and Vision Computing*, **10**(18):795–804.
10. Michel O (2004) Webots: Professional mobile robot simulation. *Journal of Advanced Robotic Systems*, **1**(1):39–42.
11. Miglino O, Lund HH, Nolfi S (1995) Evolving mobile robots in simulated and real environments. *Artificial Life*, **2**(4):417–434.
12. Mitchell M (1996) An introduction to genetic algorithms. The MIT Press, Cambridge, MA.
13. Nolfi S, Floreano D (2000) Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines. The MIT Press, Cambridge, MA.
14. Nolfi S, Parisi D (2002) Evolution of artificial neural networks. In *Handbook of brain theory and neural networks*, MIT Press, pp. 418–421.
15. Patel M, Honavar V, Balakrishnan K Eds. (2001) *Advances in the evolutionary synthesis of intelligent agents*. The MIT Press, Cambridge, MA.
16. Scott MJ, Antonsson EK (1998) Aggregation functions for engineering design trade-offs. *Fuzzy Sets and Systems*, **99**(3):253–264.
17. Versino C, Gambardella LM (1997) Learning real team solutions. In *Distributed Artificial Intelligence Meets Machine Learning*, Springer-Verlag, pp. 40–61.
18. Yao X (1999) Evolving artificial neural networks. *Proceedings of the IEEE*, **87**(9):1423–1447.