Is Situated Evolution an Alternative for Classical Evolution?

M.C. Schut and E. Haasdijk and A. Prieto

***Abstract*—In this paper we present an evolutionary method that can deal with the specific problem requirements of adaptivity, scalability and robustness. These requirements are increasingly observed in the areas of pervasive and autonomic computing, and the area of collective robotics. For the purpose of this paper, we concentrate on the problem domain of collective robotics, and more specifically on a surveillance task for such a collective. We present the *Situated Evolution Method* as a viable alternative for classical evolutionary methods specifically for problem domains with the aforementioned requirements. By means of simulation experiments for a surveillance task, we show that our new method does not lose performance in comparison with a classical evolutionary method, and it has the important design and deployment advantage of being adaptive, scalable and robust.**

I. Introduction

The research in this paper is inspired by the advent of real-life *surveillance* robots. Typically, such robots are deployed nightly in environments where they patrol and report any unexpected encounters. We envisage a system where deploying (additional) robots for surveillance requires little or no parameter setting, uploading of maps or programming of routes: a number of robots is simply set loose in some arena they have to monitor and they adapt their routing strategies autonomously as circumstances dictate (*adaptivity*). The system should be scalable to large numbers of robots and large environments without incurring undue overhead (*scalability*). Removing robots (e.g., through random failure, mishaps or even sabotage) should not impact the performance of the collective too severely and should certainly not break the system. On the other hand, adding robots should increase system performance. All this without having to modify any aspect of robots already present. Finally, the system should not have a single point of failure to ensure resilience to failure or sabotage (*robustness*).

These requirements point to a design where robots adapt autonomously on the basis of only local information. This stipulation conflicts with most traditional evolutionary algorithms where some central authority evaluates, selects and replaces individuals. Therfore, we propose autonomous *situated evolution* as an evolutionary algorithm that does comply with the need for purely local control, hopefully without sacrificing system performance in comparison to more traditional evolutionary robotics implementations.

This paper describes an initial implementation of an autonomous evolutionary algorithm (called *situated evolution*) that addresses these issues. To determine whether we can address the issues outlined above without (significant) loss of performance, we compare its performance to that of a traditional evolutionary method by means of simulation.

Department of Computer Science, Faculty of Sciences, VU University Amsterdam, The Netherlands {mc.schut,e.haasdijk}@few.vu.nl, abprieto@udc.es.

The research objective of the work described in this paper is to investigate whether situated evolution can be used as a viable alternative to classical evolution. We investigate this within the context of the presented problem domain: collective robotic surveillance. As mentioned above, our problem domain prohibits us from using a traditional evolutionary method delivering the necessary adaptivity, scalability and robustness. While there are many variants of evolutionary methods used in robotics (as discussed below in Section II), these methods do not suffice in terms of the beforementioned requirements.

This paper is structured as follows. In the following Section, we discuss related work on variants of evolutionary methods that have been used in our problem domain. In Section III, we explain our *situated evolution* method. Then, in Section IV we demonstrate this method in the abovementioned case study of surveillance robotics. Finally, Section V concludes and provides pointers for future work.

II. Background

Most evolutionary robotics approaches implement off-line robot programming, not on-line adaptivity (although the evolutionary process can be set up to deliver adaptive controllers). The (simulated or physical) robots are placed in an arena, their performance vis-a-vis some task is measured centrally and some central selection scheme is applied to select parents that produce the next iteration in the search for a good controller. The final result is a (set of) fixed controller(s) that do not adapt any further—at least, not through evolution. Nolfi and Floreano extensively describe such systems in [6].

For the evolutionary algorithm to provide continuous adaptivity, however, it has to be *embedded in the robots themselves* and executed *as they perform their tasks* in real life, preferably without interrupting those tasks (i.e., on-line evolution). Crucially, the evolutionary algorithm must employ a selection algorithm that does not need any global calculations if it is to be scalable and robust. Also, as there are typically multiple robots in the environment, it seems a waste not to exploit the inherent possibilities for collective adaptation, i.e., not to have the robots exchange information to help each other adapt.

The first attempt at truly autonomous embedded evolutionary robotics that we are aware of is presented by Watson et al. as *embodied evolution* [3], [13]. Their Probabilistic Gene Transfer Algorithm addresses all three issues mentioned above, but it does presuppose a known theoretical maximum fitness, which limits its applicability. Wischmann et al. and

978-1-4244-2959-2/09/$25.00 © 2009 IEEE

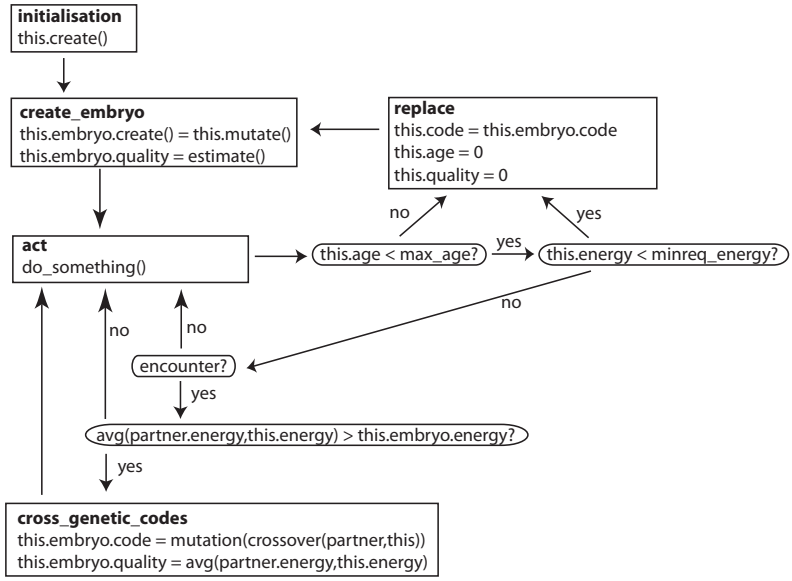

Fig. 1. A flow-chart showing the SE procedure.

Nehmzow implemented variants of this scheme [14], [5], although the latter introduces a mating mode during which the robots actively seek each other out to mate, interrupting their work on the actual task at hand.

The initial work on embodied evolution also inspired a number of implementations akin to what parallel evolutionary algorithm literature would term *island-based algorithms*. These implementations employ time-sharing to evaluate an on-board pool of individuals [11], [2], [7]. Individuals migrate from one robot to another using local communication. These implementations carry the benefit that the evolutionary algorithm works even in a single robot. Similar approaches –in the sense that a complete evolutionary algorithm runs locally within an individual robot– are described by Walker et al. and Haroun Mahdavi and Bentley [12], [4].

Simões and Dimond propose an algorithm where individual robots use radio to transmit their fitnesses and genetic material; the best individuals survive and mate to create new controllers with which to reprogram the remainder of the population [9]. Thus, this implementation has all population members fully connected (by radio), contradicting the scalability requirement.

III. Situated Evolution

The situated evolutionary algorithm that we present here is our first implementation of an embedded evolutionary algorithm that addresses all three issues mentioned above. In this algorithm, the robots adapt continuously through evolution as they perform their task. The robots share and incorporate information to learn from each other's experiences autonomously without recourse to any centralised instance or repository. Furthermore, we avoid the additional memory requirements of running a complete evolutionary algorithm within each single robot to minimise the requirements posed by the algorithm. Contrary to embodied evolution as implemented in, e.g., [13], the algorithm presented here does not require any a priori known maximum fitness.

In this algorithm, the robots have a *virtual energy level* that indicates how successful they are at the task at hand. Energy is replenished when the robot performs a task well –it can be thought of as accumulated reinforcement learning rewards– and decays over time. In the evolutionary algorithm, it plays the role of fitness: individuals are rated according to their energy level when they are being considered as mates. Thus, the higher the accumulated energy, the higher the robot's fecundity.

If the energy level reaches some minimum, the robot 'dies': it replaces its controller with an embryonic controller –the cached result of mating– as described below. As also noted in [10], this is in effect a form of *tournament selection* – De Jong and Sarma [1] note that the only traditional algorithm with this feature is tournament selection.

The SE *procedure* is shown in Figure 1 (and the pseudocode of this procedure instantiated for the surveillance task is shown in Figure 2). At initialisation, the agent is created together with its embryo. The embryo is given an estimated fitness value. After that, the agent continuously walks around, performing its task (here: cell coverage). Each iteration, it checks its age (if it is too old, it dies) and fitness (if it is too low, it dies). If it encounters another agent, then the fitnesses of 1) the possible offspring (average of fitnesses of two parents) and 2) the embryos are compared. Depending on this comparison, either no interaction takes place, or the embryo is replaced by a recombined version of the two parents, and its quality value estimate is updated accordingly. The estimate is set to be half of the original agent quality if the embryo is a mutation of the original agent. Note that it

```
INITIALISATION() :- {
FOR EACH watcher
  Random generation of the controller_genetic_code;
  Random generation of the embryo_genetic_code;
  prequality = 0;
  quality = 10000;
  T_last_mating = 0;
}

ACT() :- {
FOR EACH watcher
  Cover cells in range;
  Refresh quality;
  Sense obstacles (obs) and agents (agn);
  Refresh trajectory memory (trm);
  angleVariation = ArtNeuralNetwork(obs,agn,trm);
  position = position + speed * AngleVector;
  age++;
  IF (age < maturity || quality < 1)
    THEN quality = 1;
  IF closest_watcher_distance < range_meeting
    THEN MEETING (closest_watcher);
  IF (quality < 0 || age > 1000)
    THEN REPLACE();
}

MEETING(watcher2) :- {
IF ((AVERAGE(quality,watcher2.quality) > prequality)
   && (T_last_mating < ActualTime - 10))
  THEN embryo_genetic_code =
       XOVER_MUTATION(genetic_code,
                      watcher2.genetic_code);
       prequality = AVERAGE(quality,watcher2.quality);
}

REPLACE() :- {
controller_genetic_code = embryo_genetic_code;
embryo_genetic_code = MUTATION(embryo_genetic_code);
quality = prequality / 2.0;
prequality /= 2.0;
}
```

Fig. 2. Pseudocode of the SE procedure.

is important that the estimate is less than the original agent quality to avoid maintaining ad infinitum genetic codes of embryos wrongly estimated with high quality. If the embryo is created from the combination of two agents, it is the average of the quality of their qualities.

IV. Case Study

As mentioned, the case study in which we evaluate the SE method presented above, concerns a *surveillance task* in an enclosed area. Figure 3 shows a screenshot of the simulation software that we programmed, using the WaspBed simulation [8]. There are a fixed number of surveillance agents (indicated by black dots) that have a particular field-of-vision (shown by a circular area in front of the agent). There are immovable obstacles in the environment (indicated by black blocks) over which the agents cannot pass. The different shades indicate the degrees of *cell coverage* – the darker a cell is, the longer it has not been observed.

In this Section, we present a precise task description, experimental design and setup, followed by the obtained results and analysis, linking back to the earlier presented objectives.

A. Task Description

The task involves to surveil the arena shown in Figure 3 as good as possible with a group of robot agents. Our

Fig. 3. Screenshot of the simulated arena.

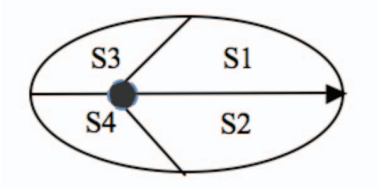

Fig. 4. Vision of the robots, where Sx refers to a particular section.

implementation of this task is that all cells in the arena have a value that increases while it is not observed (i.e., not within the vision of an agent) and is set to 0 when it is observed. Following, we give short descriptions of the agents, the environment and performance measurement.

Agents – each agent continuously moves throughout the arena covering cells. An agent has at all times a *position* and an *orientation angle*. Its fixed physical parameters are its speed, range of vision (forward and backward), and vision angle. These parameters are the same for all agents in all simulations and are not subject to change. The controller parameters (which *are* able to change) are the position, the heading and the fitness level. Agents age and mate, and they receive rewards for cell coverage. An agent's maximum age is 1,000 iterations. Its maturity age is 100 time steps: before reaching the maturity age, the energy level cannot go below 1, so they cannot die. This transient time was included for agent to potentially recover from bad initial circumstances.

Velocity: the agent's moving speed is fixed at 5 pixels per iteration. Its angular velocity is decided by the agent's controller (see below) and is within the range (-100, 100) arc degrees per iteration.

Memory: each agent has a *trajectory memory*, which is an array of length 100 where the last 100 positions of the agent are stored. From this, the *module* and *angle* between the positions where the agent was n times steps ago and the current position are calculated as follows. Let the agent be at time i at location (x_i, y_i) and at time $i-n$ at location (x_{i-n}, y_{i-n}). Then angle = $arctan[(y_i - y_{i-n})/(x_i - x_{i-n})]$

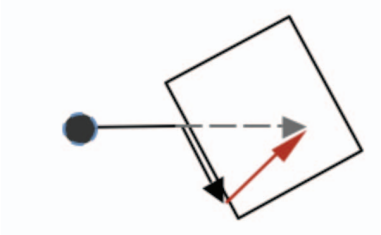

Fig. 5. The concept of an 'angle' as used for obstacle avoidance.

and module = $sqrt[(x_i - x_{i-n})^2 + (y_i - y_{i-n})^2]$, where both are normalised within $[-1, 1]$.

Controller: the control system for each agent is a Radial Basis Function Network (RBFN) [15], with 7 neurons in the input layer, 6 hidden neurons in 1 hidden layer and 1 output neuron. It is typical of an RBFN that the neurons in the hidden layer have a non-linear activation function. In addition to the weights of the connections between the input, hidden and output nodes ($7 \times 6 + 6 \times 1 = 48$) each hidden node has an amplitude parameter. This results in 54 (evolvable) values in this network. From the 7 input neurons, 5 are for the input sensors and 2 for the trajectory memory (the module and angle). From the 5 input sensors, 1 is for detection of hits against obstacles (the variation between the estimated trajectory and the real trajectory when hitting against an obstacle, see Figure 5) and 4 for the detection of the number of agents within each of the four sectors of vision (see Figure 4).

Vision: figure 4 shows the vision range to detect (and cover) cells, which is an ellipse with one focus in the position of the watcher and another focus on the major axis following the orientation angle[1]. The distance between these *foci* is 40 pixels, and the sum of distances from every point within the vision range to both foci is less than 60 pixels. The line dividing s1/s3 and s2/s4 is at 45 degrees from the major axis. The vision range to detect other close agents is the same as the cells vision range but 5 times longer, so 25 times larger. 200 pixels between foci and 300 the sum of the distances to foci to be detected. However, this bigger vision range is not shown in Figure 3 in the interest of clarity.

Evolution: the genetic code contains 55 REALS: 54 for the parameters of the agent's NN (as explained above), and 1 for the length (N) of the trajectory memory. We use a crossover of real numbers to obtain the new chromosome from the two original ones. Each of the genes of the child chromosome is based on the genes of its parents, modified with a given Gaussian-like density probability function (specifics can be found in the software, which is available from the first author's website.). Additionally, we use a probability of 0.05 per chromosome for random mutation as well.

Environment – the environment size is 1,000 $\times$ 700 pixels. It consists of a grid with cells of 20 $\times$20 pixels. At the borders, there are 20 pixels wide boundaries.

[1]Thus these two foci define the ellipse.

Cells: each grid cell has a value that grows over time. An agent can 'see' cells, which results in the involved cell values being reset.

Obstacles: these are rectangular shaped objects with fixed positions throughout the simulation. The heights and widths of the obstacles are randomly determined at the start of a simulation run (either 50, 100 or 150 pixels). Obstacles are placed randomly within the scenario (the left-up corner position is selected randomly with steps of 50 pixels.) Obstacles can also be used deterministically to create paths and areas and are created randomly (size and position) at the beginning of each simulation run.

Performance measurement – each cell has a so-called *coverage value* that expresses how well the agents have covered this cell: if this value is high, it was a long time since the cell was observed and vice versa. The coverage value grows over time according to a sigmoid function. The task of the agents is to cover all cells as good as possible – in other words, to minimise the coverage values of all cells. If an agent observes a cell, then it receives the cell coverage value as its reward and the coverage value is reset to 0.

We have thus two ways to measure performance: on the agent level, we can monitor the *virtual energies* of the agents to see how well the agents are doing; on the system level, we can measure the total *coverage value* of all the agents together. In the experiment below, we are interested in the system performence (i.e., total coverage value) and consider the agent's energy values as an intermediate performance indicator used for the system's evolution.

B. Classical Evolution

In the experiments, we compare situated evolution with 'classical' evolution. Here, we briefly explain our 'classical' evolution. In classical evolution (CE), a best individual agent emerges over a number of epochs. At first, N populations are created with homogenous agents. Each epoch consists of a number of N simulation runs, in which each population is put in the task environment and run for a fixed amount of time. After finishing a run, the total performance of the population is measured. Based on this performance, the original N agents are recombined and mutated, leading to the next generation, with which the next epoch runs. In the pseudocode in Figure 6, we show in detail what happens in CE.

C. Experiments

We conducted two series of experiments (one for situated evolution (SE) and one for classical evolution(CE) in the surveillance problem domain (as described above). In this Section, we present the experimental design, setup, results and analysis.

1) Design and Setup: In each of the two series, we let the respective method (SE or CE) evolve for 30,000 timesteps. In each series, we conduct 25 experimental runs and we report on the average values of these runs (when presenting the results below). In each run, we have a robot collective consisting of 20 robots.

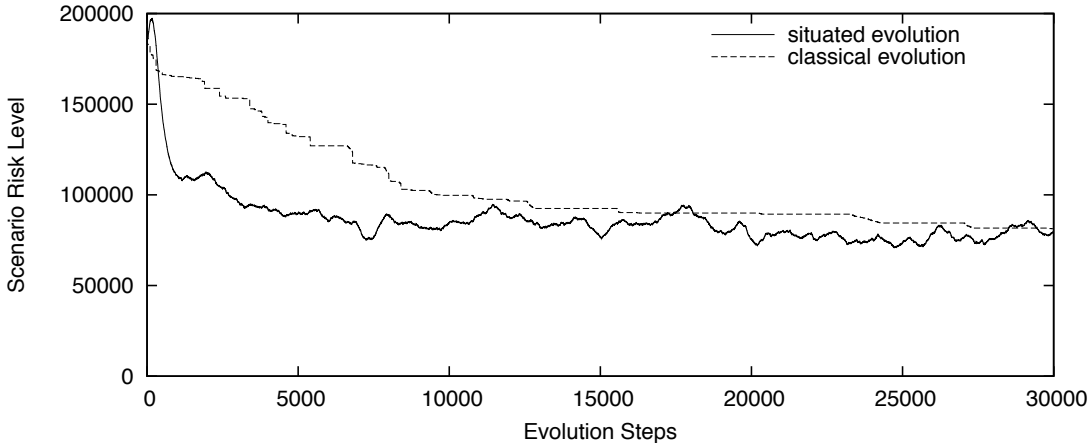

Fig. 7. Obtained results.

```
INITIALISATION() :- {
FOR EACH individual
  Random generation of the controller_genetic_code;
global_quality = 200000;
}

EVOLUTION() :- {
FOR m = [100,200,400,800]
  FOR EACH individual
    FOR EACH watcher
      watcher.genetic_code = individual.genetic_code;
      DURING m timesteps
        FOR EACH watcher
          Cover cells in range;
          Sense obstacles (obs) and agents (agn);
          Refresh trajectory memory (trm);
          angleVariation = ArtNeuralNetwork(obs,agn,trm);
          position = position + speed * AngleVector;
      Refresh (individual.)global_quality(scenario quality);
    FOR n = 1 to 20 (number of individuals)
      parents = TOURNAMENT(individuals, global_quality,
                           size = 2);
      individuals_aux[n].genetic_code =
      XOVER_MUTATION(parents[0].genetic_code,
                     parents[1].genetic_code);
    individuals = individuals_aux;
}
```

Fig. 6. Pseudocode of the CE procedure.

Our performance measure is the sum (over all the locations in the arena) of the before-mentioned *coverage values*: we call this sum the *scenario risk level*. The aim for the robot collective is to minise this sum.

Our aim with conducting these experiments is to see whether situated and classical evolution deliver comparable results in terms of performance (hence, the question in the title whether situated evolution can be an *alternative* for classical evolution).

The setup of the SE series is simple: we initialise the robot collective, where each robot runs the SE method as explained above and we let the simulation run for the maximum numer of timesteps. The setup of the CE series is more complex. First, each of the 20 individuals in the population is tested during a given number of timesteps, where this number of time steps is gradually increased [100, 200, 400 and 800] in order to speed up convergence. Then, after testing we obtain a fitness value – if this value is lower than the current fitness value for that timestep then we replace the robot.

2) Results: Figure 7 shows the obtained results from both the SE and CE series. Here, we plotted the development of the performance measure (*scenario risk level*) over time (*evolution steps*). Note that we deliberately did not include standard deviations in the graphs; statistical analysis showed that there is no significant difference between the two graph lines.

3) Analysis: In Figure 7 we most importantly see that there is no significant difference between the measured performance of the SE and CE methods. With respect to the objective of this paper defined above (to see whether situated evolution could be a viable alternative for classical evolution) we can thus say that SE does not outperform CE but it also does not perform worse. Considering the beforementioned advantages of situated over classical evolution (i.e., adaptivity, scalability and robustness), we consider this a valuable and worthwhile result.

Besides obtaining this main result, we made some more observations. Firstly, we observed a very rapid convergence of the SE robots: they all tend to rapidly develop towards similar controllers. We attribute this to the "tournament size" that may have been set too large. As robots wander through the arena and 'evolve' upon encountering other agents, we discovered that the number of robots encountered before reproduction is almost the whole population. This is very large, especially in comparison with traditional EAs. Secondly, for both methods, we see that the performance increases over time, but only comes half way (from 200,000 to 100,000). Finally, we see that the CE graph line is more 'stable' (less fluctuation) than the SE graph line. This can be easily explained by the fact that the CE method is is centrally coordinated, which makes it easy to regulate (for example, giving the desirable "anytime behaviour" that traditional EAs have). By distributing the evolutionary process, one loses such regulation. However, overall we can still say that in both

lines we see the same overall trends (increasing performance over time).

V. Conclusions

The research that we present here comprises a first step on the long road towards our ultimate goal of *continuously and autonomously self-adapting ensembles of robots*. The goal that we set ourselves in this particular paper is modest, especially in comparison to this goal: to show that the fully autonomous strategy of situated evolution is a viable strategy with comparable, if not better, results than 'traditional' off-line evolutionary robotics.

Although the implementation of situated evolution in this paper must be considered a first step and can be improved in many ways, not least through thorough tuning of the –in some cases new and unexplored– parameters, we showed that this implementation is already capable of achieving results comparable to those of 'traditional' evolutionary robotics (admittedly, without tuning the parameters of that technique, either). We confidently look forward to further advances on the road towards adaptive robotics through true situated evolution with its inherent benefits of continuous, robust and autonomous adaptivity.

Concerning future work and in terms of concrete steps towards our beforementioned ultimate goal are: a detailed analysis of the new (impact of) parameters such as sample size and maximum age; extending this research to more complex controllers for more sophisticated strategies; and, finally, to leverage possibilities for collaboration and combining individual (e.g., reinforcement) learning with evolutionary adaptation.

Acknowledgements

Part of this work was made possible by the European Union FET Proactive Intiative: Pervasive Adaptation funding the Symbrion project under grant agreement 216342; and by the MEC of Spain through project DPI2006-15346-C03-01.

References

[1] Kenneth De Jong and Jayshree Sarma. On decentralizing selection algorithms. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 17–23, San Francisco, CA, 1995. Morgan Kaufmann.

[2] S. Elfwing, E. Uchibe, K. Doya, and H.I. Christensen. Biologically inspired embodied evolution of survival. In Zbigniew Michalewicz and Robert G. Reynolds, editors, *Proceedings of the 2008 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation*, volume 3, pages 2210–2216, Hong Kong, June 2008. IEEE Press.

[3] S. Ficici, R. Watson, and J. Pollack. Embodied evolution: A response to challenges in evolutionary robotics. In J. L. Wyatt and J. Demiris, editors, *Proceedings of the Eighth European Workshop on Learning Robots*, pages 14–22, 1999.

[4] Siavash Haroun Mahdavi and Peter J. Bentley. Innately adaptive robotics through embodied evolution. *Auton. Robots*, 20(2):149–163, 2006.

[5] Ulrich Nehmzow. Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. In C.G. Prince, Y. Demiris, Y. Marom, H. Kozima, and C. Balkenius, editors, *Proceedings of The Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, number 94 in Lund University Cognitive Studies, Edinburgh, UK, August 2002. LUCS.

[6] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. The MIT Press, Cambridge, MA. / London, 2000.

[7] Anderson Luiz Fernandes Perez, Guilherme Bittencourt, and Mauro Roisenberg. Embodied evolution with a new genetic programming variation algorithm. *icas*, 0:118–123, 2008.

[8] Abraham Prieto, Francisco Bellas, Pilar Caamano, and Richard J. Duro. A complex systems based tool for collective robot behavior emergence and analysis. In Emilio Corchado, Ajith Abraham, and Witold Pedrycz, editors, *3rd International Workshop on Hybrid Artificial Intelligence Systems*, volume 5271 of *Lecture Notes in Computer Science*, pages 633–640, 2008.

[9] Eduardo D. V. Simões and Keith R. Dimond. Embedding a distributed evolutionary system into population of autonomous mobile robots. In *Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference*, 2001.

[10] Robert E. Smith, Claudio Bonacina, Paul Kearney, and Walter Merlat. Embodiment of Evolutionary Computation in General Agents. *Evolutionary Computation*, 8(4):475–493, 2000.

[11] Y. Usui and T. Arita. Situated and embodied evolution in collective evolutionary robotics. In *Proceedings of the 8th International Symposium on Artificial Life and Robotics*, pages 212–215, 2003.

[12] Joanne H. Walker, Simon M. Garrett, and Myra S. Wilson. The balance between initial training and lifelong adaptation in evolving robot controllers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(2):423–432, 2006.

[13] Richard A. Watson, Sevan G. Ficici, and Jordan B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, April 2002.

[14] Steffen Wischmann, Kristin Stamm, and Florentin Wörgötter. Embodied evolution and learning: The neglected timing of maturation. In *Advances in Artificial Life: 9th European Conference on Artificial Life*, LNAI, in press. Springer-Verlag, in press.

[15] Paul V. Yee and Simon Haykin. *Regularized Radial Basis Function Networks: Theory and Applications*. John Wiley, 2001.